МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ПрАТ «ПВНЗ «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра інформаційних технологій

ДО ЗАХИСТУ ДОПУЩЕНА

Зав. кафедри _____

д.е.н., доц. С.І. Левицький

МАГІСТЕРСЬКА ДИПЛОМНА РОБОТА

ТЕХНОЛОГІЯ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ ПРО НАДАННЯ ПОСЛУГ РІЕЛТОРА

Виконав

ст. гр. ІПЗ–111м _____ І.О. Суботін

Науковий керівник

к.т.н., доц. _____ Ю.С. Резніченко

Запоріжжя

2023 р.

ПРАТ «ПВНЗ «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра інформаційних технологій

ЗАТВЕРДЖУЮ
Зав. кафедри
д.е.н., доцент Левицький С.І.
_____
03.10.2022 р.

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

ст. гр. ІПЗ–111м, спеціальності 121 «Інженерія програмного забезпечення»
ОП «Інженерія програмного забезпечення»
Суботіна Ігоря Олександровича

1.Тема: Технологія підтримки прийняття рішень про надання послуг ріелтора затверджена наказом по інституту № 02-16 від 03.10.2022 р.

2. Термін здачі студентом закінченої роботи: 12.01.2023 р.

3. Перелік питань, що підлягають розробці:

1. Виконати аналіз сучасного стану сфери надання ріелторських послуг
2. Виконати аналіз методів підтримки прийняття рішень
3. Розробити рекомендаційний алгоритм для підтримки прийняття рішень про надання послуг ріелтора
4. Розробити базу даних рекомендацій з урахуванням критеріїв, що впливають на вибір користувача системи надання послуг ріелтора
5. Розробити рекомендаційний мікросервіс та інтегрувати його до системи надання послуг ріелтора

4. Календарний графік підготовки магістерської дипломної роботи

№ етапу	Зміст	Терміни виконання	Готовність по графіку %, підпис керівника	Підпис керівника про повну готовність етапу, дата
1.	Формулювання теми магістерської дипломної роботи (збір практичного матеріалу за темою магістерської дипломної роботи)	20.10.22		
2.	І атестація І розділ магістерської дипломної роботи	27.10.22		
3.	ІІ атестація ІІ розділ магістерської дипломної роботи	17.11.22		
4.	ІІІ атестація ІІІ розділ магістерської дипломної роботи, висновки та рекомендації, додатки, реферат, перевірка програмою «Антиплагіат»	29.12.22		
5.	Доопрацювання магістерської дипломної роботи, підготовка презентації, отримання відгуку керівника та рецензії	10.01.23		
6.	Попередній захист магістерської дипломної роботи	12.01.23		
7.	Надання магістерської дипломної роботи на кафедру	за 3 дні до захисту		
8.	Захист магістерської дипломної роботи	19.01.23		

Дата видачі завдання: 03.10.2022 р.

Керівник магістерської дипломної роботи _____ Ю.С. Резніченко
(підпис) (прізвище та ініціали)

Завдання отримав до виконання _____ І.О. Суботін
(підпис студента) (прізвище та ініціали)

РЕФЕРАТ

Магістерська дипломна робота містить 97 сторінок, 2 таблиці, 27 рисунків, 2 додатки, 1 лістинг, 31 бібліографічне посилання.

Магістерську дипломну роботу спрямовано на розробку рекомендаційного алгоритму для надання послуг ріелтора та розробку на його основі рекомендаційного мікросервісу для системи підтримки прийняття рішень про надання послуг ріелтора.

Об'єктом дослідження є методи підтримки прийняття рішень. Предметом дослідження є критерії та алгоритми формування релевантних рекомендацій.

У першому розділі проведено аналіз сучасного стану сфери купівлі/продажу нерухомості. Виявлено різноманітні критерії, що впливають на прийняття користувачами системи надання послуг ріелтора. Обґрунтовано доцільність розробки рекомендаційного алгоритму для інтеграції до системи надання послуг ріелтора.

У другому розділі проведено аналіз методів підтримки прийняття рішень. Наведено огляд та переваги використання у розробці рекомендаційного мікросервісу наступного стеку технологій: Java, Spring, Maven, MongoDB, AWS Redshift, AWS SQS, Swagger, GraphQL.

В третьому розділі проведено огляд існуючих програмних аналогів. Виконано проектування рекомендаційного мікросервісу. Описано програмну реалізацію рекомендаційного мікросервісу як для розробника, так і для користувача.

РЕКОМЕНДАЦІЙНИЙ АЛГОРИТМ, РЕКОМЕНДАЦІЙНИЙ МІКРОСЕРВІС, JAVA, SPRING, MAVEN, MONGODB, AWS REDSHIFT, AWS SQS, SWAGGER, GRAPHQL

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

Слово / словосполучення	Скорочення	Умови використання
База даних	БД	У тексті

ВСТУП

Більшість сучасних web-сайтів використовують системи підтримки прийняття рішень (СППР) для того, щоб задовольнити потреби користувачів. Такі СППР генерують різні комплексні пропозиції, рекомендації (наприклад, пропонують супутні товари, звертають увагу на людей зі схожими уподобаннями). Рекомендаційні механізми СППР обробляють величезні обсяги інформації для позначення потенційних переваг користувачів, для визначення найпридатніших результатів пошуку, для прогнозування потенційно цікавого контенту. Рекомендаційні механізми удосконалюють взаємодію між користувачем та web-сайтом. Замість статичних даних вони надають динамічні змінні: рекомендації генеруються індивідуально під кожного користувача, ураховуючи різні характеристики його активності на даному web-ресурсі, а іноді й інтегрують дані, що надходять від інших відвідувачів.

Ринок купівлі-продажу нерухомості відіграє істотну роль у житті суспільства. Мільйони різних варіантів житлових об’єктів у всьому світі виставлено на продаж, ще більше запитів на придбання нового житла. Діяльність певної категорії людей так чи інакше пов’язана з реалізацією бажання з купівлі або продажу житла. Клієнт звертається до компанії або приватного ріелтора, якого йому порадив друг, колега або побачивши рекламу в Інтернеті тощо. У результаті виникає безліч проблем та труднощів, пов’язаних з навичками агента, специфікою житла тощо. Доцільним рішенням є підбір агентів залежно від інтересів клієнта, з урахуванням його власних критеріїв та пріоритетів. Сервіс для покупців/продавців житла, який вчасно інформує, спрямовує приймати кращі рішення та мати кращий досвід взаємодії з питань купівлі/продажу.

Найчастіше розвиток рекомендаційних механізмів полягає в удосконаленні алгоритмів прийняття рішень. Мета цього процесу − давати відвідувачам web-сайтів якомога найповніші та найточніші рекомендації, що задовольняють їхні запити. Для цього математичні алгоритми, що лежать в

основі рекомендаційних механізмів, повинні постійно навчатися. Отже набуває чинності машинне навчання [1] та інтелектуальний аналіз даних [2]. Спрощено схему можна описати таким чином: сервіс web-сайту надає користувачу набір рекомендацій, далі розробник отримує від нього зворотний зв'язок, аналізує дані на відповідність інтересам користувача, перенавчає математичну модель, потім знову пропонує рекомендації. Далі це відбувається циклічно.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність теми роботи

Обсяг світового ринку нерухомості у 2021 році оцінювався в 3,69 трильйона доларів США, і очікується, що з 2022 по 2030 рік він зростатиме на 5,2% у середньому річному темпі. Очікується, що протягом прогнозованого періоду ринок зростатиме здоровими темпами, у зв’язку зі зростанням населення та бажанням мати особистий простір у будинку. Станом на 2021 рік площі комерційної нерухомості вважалися найважливішим елементом, що стимулює розширення галузі [3].

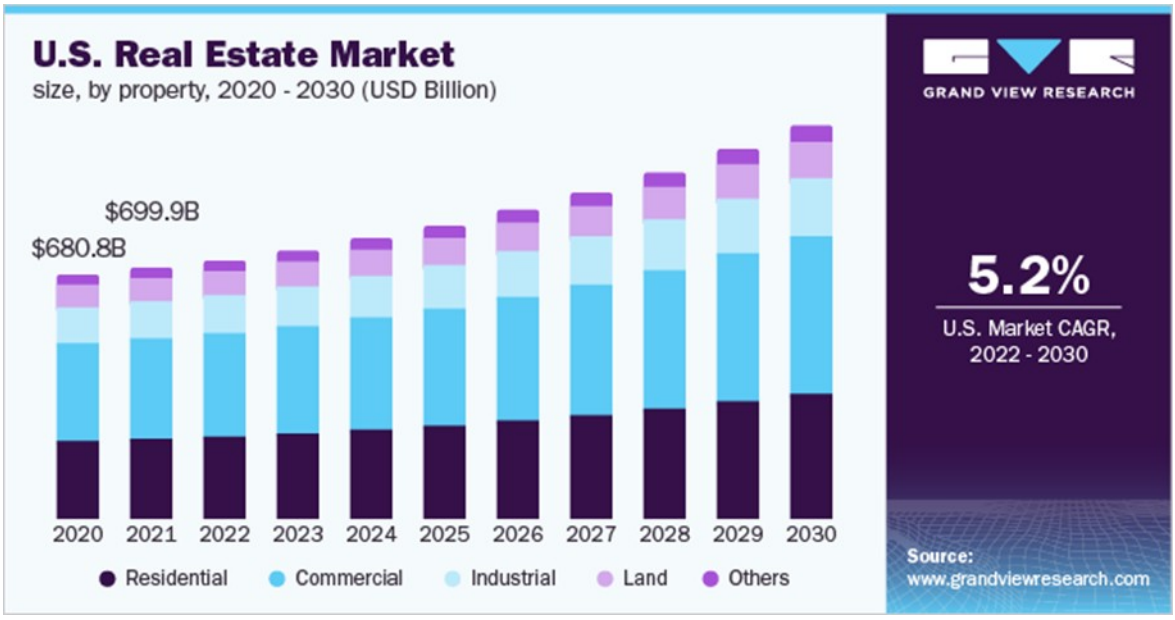


Рисунок 1.1 – Діаграма прогнозованого зростання ринку нерухомості

Пандемія COVID-19 негативно вплинула на зростання ринку нерухомості. Вплив пандемії був помітний у перші кілька місяців кризи, особливо з точки зору роздрібної торгівлі, завдяки жорсткому карантину та обмеженням пересування. Блокування, введені в різних регіонах, призвели до

затримки нових будівельних проектів та призвели до повільного зростання промисловості [4].

Незважаючи на величезне скорочення продажів житла внаслідок пандемії, активність у сфері нерухомості почала відновлюватися, повертаючись до рівня до пандемії. Потенційні покупці почали активізувати пошук та купівлю будинків, що сприяло зростанню ринку нерухомості. За даними Національної асоціації ріелторів, незавершені продажі в метрополітенах США, які в квітні 2020 року впали більш ніж на 30%, до серпня 2020 року зросли майже на 30% [5].

Крім того, вплив Інтернету підвищив обізнаність споживачів щодо онлайн-послуг з нерухомості. Ключові гравці пропонують різноманітні послуги, такі як кімнати для прямих трансляцій, щоб отримати частку ринку. Наприклад, за даними Alibaba, понад 5000 агентів з нерухомості з майже 100 місць у Китаї застосували метод прямої трансляції в кімнатах, що дозволяє покупцям житла досліджувати будинки та укладати угоди вдома [6].

Житлова нерухомість домінувала на ринку з часткою доходу 35,5% у 2021 році. Зростання відбувається переважно за рахунок міленіалів, оскільки останніми роками вони більш схильні до володіння житлом. Наприклад, згідно зі звітом Homeownership від Apartment List, рівень володіння житлом серед міленіалів зріс до 47,9% у 2021 році з 40% у 2022 році [7].

Прогнозується, що CAGR (Compound Annual Growth Rate, сукупний середньорічний темп зростання) комерційної нерухомості складе 5,1% з 2022 по 2030 рік [8]. Ринок процвітає винятковими темпами в результаті зростання туристичного сектора.

З точки зору бізнеса виявлено гарний показник зростання попиту. Розлянемо доцільність розробки рекомендаційного алгоритму замість традиційного використання бази даних ріелторських послуг.

По-перше, виявлено збільшення коефіцієнта конверсії. Ретельно підібрана рекомендація дозволяє підвищити рівень конверсії. За даними джерела Barilliance приблизно на 8% [9].

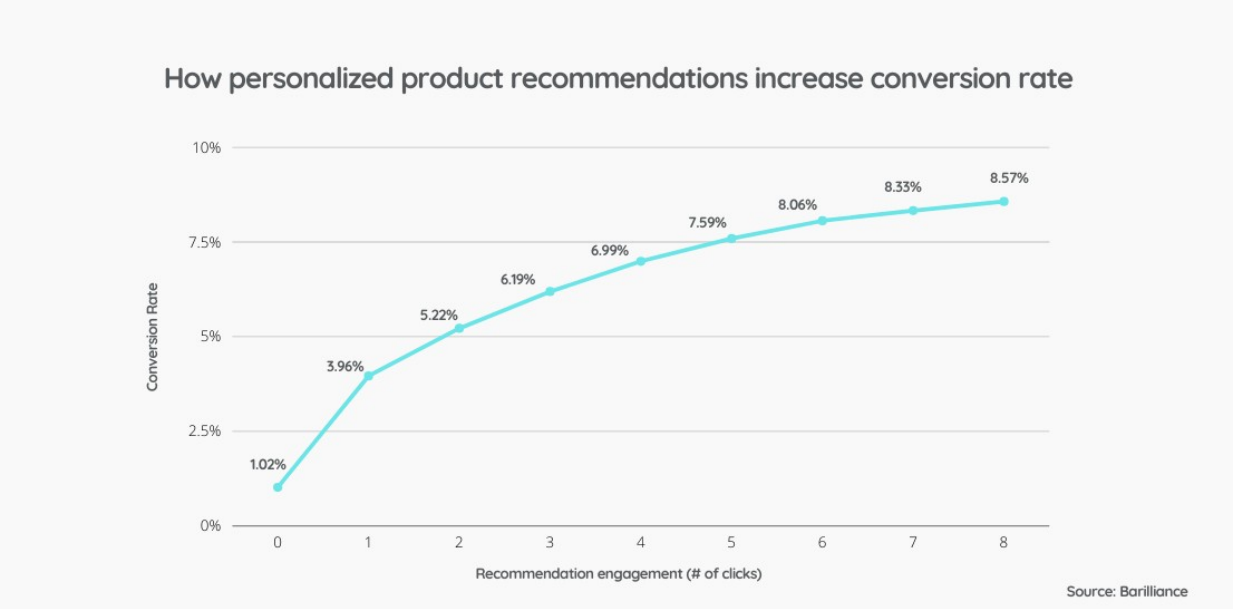


Рисунок 1.2 – Діаграма збільшення конверсії в залежності від рекомендацій

По-друге, виявлено зростання доходу. Більш цілеспрямований підхід до орієнтування клієнтів на правильний шлях купівлі призводить до значного збільшення продажів. Дослідження Gartner свідчать про те, що інтелектуальні механізми персоналізації дозволяють отримати на 15% більше доходу. Крім того, за статистикою McKinsey, рекомендації щодо продукту використовуються для збільшення доходу на 5-15% [10].

За сучасними даними кожному успішному бізнесу доцільно мати стратегію рекомендації продуктів/послуг своїм клієнтам. Однак розробити комплексну стратегію для створення різних етапів та різних типів структур рекомендацій продукту та ефективно реалізувати цю стратегію – складне завдання.

По-третє, виявлено економію часу та ресурсів. Ручний пошук у базі даних ріелторів, що відповідають певним критеріям є складним та трудомістким.

Завдяки системі надання послуг ріелтора клієнт може швидко отримати необхідну інформацію та на її основі отримати якісні послуги. Також надається можливість скоротити базу операторів – менеджерів з продажу, бо більшість роботи виконує система надання послуг ріелтора.

Таким чином, web-ресурс, що здатен навчатися та використовує запатентований механізм рекомендацій для домовласників, які вивчають та оцінюють безліч способів продажу будинку, є корисним та ефективним. Рекомендаційний механізм, заснований на машинному навчанні, дозволяє ураховувати безліч критеріїв, пов’язаних з окремими об'єктами нерухомості, місцевими ринками, перевагами окремих продавців тощо. На основі підібраних запитань про будинок, головні цілі у процесі продажу тощо, запатентована технологія оцінює кращих місцевих агентів з нерухомості, інституційних покупців, моделі дисконтних агентів тощо, щоб підібрати варіант, який найкраще допоможе у досягненні цілей користувачів. Служба також рекомендує особистого консьєржа, щоб допомогти домовласникові в процесі продажів.

«Продаж будинку − одне з найважливіших фінансових − і часто найбільш емоційних − рішень, які приймають люди, та, оскільки це може статися тільки один або два рази протягом життя, це не те, що з часом стає другою натурою», − говорить М. Вудс, президент SOLD.com. − «Ми прагнемо допомогти домовласникам зрозуміти всі доступні їм варіанти продажу своїх будинків, захищаючи при цьому їх цінний капітал, та будемо служити надійним та неупередженим ресурсом протягом усього процесу продажу будинку». «Оскільки існує безліч нових моделей, доступних в домашніх торгових площах на додаток до традиційної моделі брокера по нерухомості, існує гостра потреба в послузі, яка враховувала б усі доступні варіанти», − продовжив М. Вудс, − «Заповнити цю прогалину у знаннях, об'єднуючи як традиційні, так і нові методи продажу будинку під одним дахом та надаючи безкоштовні та

неупереджені рекомендації, персоналізовані для кожного власника будинку» [11].

1.2 Мета та задачі роботи

Метою даної магістерської дипломної роботи є розробка рекомендаційного алгоритму для надання послуг ріелтора та розробка на його основі рекомендаційного мікросервісу для системи підтримки прийняття рішень про надання послуг ріелтора.

Для досягнення мети необхідно виконати наступні задачі:

1. Виконати аналіз сучасного стану сфери надання послуг ріелтора.
2. Виконати дослідження методів підтримки прийняття рішень.
3. Розробити рекомендаційний алгорим для підтримки прийняття рішень про надання послуг ріелтора.
4. Розробити базу даних рекомендацій з урахуванням критеріїв, що впливають на прийняття рішень користувачем системи надання послуг ріелтора.
5. На основі рекомендаційного алгоритму розробити рекомендаційний мікросервіс.
6. Інтегрувати рекомендаційний мікросервіс до системи надання послуг ріелтора.

Об'єктом дослідження є методи підтримки прийняття рішень.

Предметом дослідження є критерії та алгоритми формування релевантних рекомендацій.

У даній магістерській дипломній роботі було використано методи підтримки прийняття рішень та методи системного аналізу, методи об’єктно-орієнтованого проектування та web-програмування.

1.3 Наукова новизна

У даній магістерській дипломній роботі запропоновано нову технологію підтримки прийняття рішень про надання послуг ріелтора.

Наукова новизна отриманих результатів полягає у тому, що:

- ураховано критерії, що впливають на прийняття рішення користувачем системи надання послуг ріелтора;
- розроблено рекомендаційний алгоритм для підтримки прийняття рішень з метою підвищення швидкості та якості обслуговування;
- попередні результати реалізовано у рекомендаційному мікросервісі, який інтегровано до системи надання послуг ріелтора.

1.4 Плановане практичне значення

Практичне значення результатів, отриманих під час написання даної магістерської дипломної роботи, полягяє у програмній реалізації можливості надання релевантних рекомендацій під час роботи з системою надання послуг ріелтора.

Рекомендаційний мікросервіс має на меті формування цілісної, неупередженої рекомендації, яка підвищить швидкість та якість надання ріелторських послуг.

Рекомендаційний мікросервіс ураховує безліч різних фвкторів, наприклад, стан ринку, який визначив продавець, особисті переваги та цілі продавця, оцінки найкращих місцевих агентів з нерухомості, інституційних покупців, моделей дисконтних агентів, агентів зі знижками та фіксованою платою, моделей торгівлі тощо.

1.5. Висновки до першого розділу

У результаті аналізу сучасного стану сфери надання ріелторських послуг обгрунтовано доцільність розробки рекомендаційного алгоритму для підтримки прийняття рішень про надання послуг ріелтора та розробки на його основі рекомендаційного мікросервісу. Виявлено різноманітні критерії, що впливають на прийняття рішення користувачами системи надання ріелторських послуг. Сформульовано мету та задачі даної магістерської дипломної роботи. Описано наукову новизну та планове практичне значення даної магістерської дипломної роботи.

РОЗДІЛ 2
МЕТОДИ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ ТА ІНСТРУМЕНТИ РОЗРОБКИ

2.1 Java та Spring

Мовою програмування було обрано мову Java [12-14], як найстабільнішу та найбільш підходящу мову для бекенд-рішення високого навантаження. Також на цій мові вже написані інші мікросервіси існуючої системи. Це об'єктно-орієнтована мова програмування зі строгою типізацією, заснована на класах та інтерфейсах з підтримкою наслідування - Java швидка, безпечна і надійна. Широко використовується для розробки додатків під різні операційні системи, в центрах обробки даних, ігрових консолях, наукових суперкомп'ютерах і т. д. Платформа Java - це набір програм, які допомагають програмістам ефективно розробляти і запускати додатки для програмування Java. Він включає в себе механізм виконання, компілятор і набір бібліотек. Це набір комп'ютерного програмного забезпечення та специфікацій.

Розглянемо важливі можливості Java:

1. «Напишіть код один раз та запустіть його практично на будь-який обчислювальній платформі».
2. Мова не залежить від платформи. Деякі програми, розроблені на одній машині, можуть виконуватися на іншій машині.
3. Мова, призначена для створення об'єктно-орієнтованих програм.
4. Багатопоточна мова з автоматичним управлінням пам'яттю.
5. Мова, створена для розподіленої середовища Інтернет.
6. Мова полегшує розподілені обчислення, орієнтовані на мережу.

JDK (Java Development Kit) - це середовище розробки програмного забезпечення, що використовується для створення аплетів і додатків Java.

Розробники Java можуть використовувати його в Windows, macOS, Solaris та Linux. JDK допомагає їм створювати та запускати програми Java. На один комп'ютер можна встановити кілька версій JDK.

Розглянемо основні особливості JDK:

- JDK містить інструменти, необхідні для написання програм Java, і JRE для їх виконання;
- включає в себе компілятор, спосіб запуску додатків Java, засіб перегляду аплетів тощо;
- компілятор перетворює код, написаний на Java, в байт-код;
- інструмент запуску додатків Java відкриває JRE, завантажує необхідний клас і виконує його основний метод.

- це механізм, який забезпечує середовище

Розглянемо основні особливості віртуальної машини Java (JVM):

- забезпечує незалежний від платформи спосіб виконання вихідного коду Java;
- має безліч бібліотек, інструментів та фреймворків;
- підтримує можливість працювати на будь-якій платформі та заощадити час;
- поставляється з компілятором JIT (Just-in-Time), який перетворює вихідний код Java в машинний мову низького рівня.

JRE - програма, призначена для роботи з іншим програмним забезпеченням, що містить бібліотеки класів, клас завантажувача та JVM.

Розглянемо основні особливості JRE:

- містить бібліотеки класів, JVM та інші допоміжні файли, не включає ніяких інструментів для розробки Java, таких як відладчик, компілятор тощо;
- використовує важливі класи пакетів, такі як бібліотеки math, swing, util, lang, awt та runtime;
- підтримує запуск Java-аплетів.

Розглянемо основні концепції фреймворку Spring [15-17]:

1. Інверсія управління: в цьому принципі об'єкти не залежать від інших об'єктів, але знають про їх абстракціях для подальшої взаємодії.
2. Упровадження залежностей: в цій концепції один незалежний об'єкт використовує інші об'єкти за допомогою інтерфейсів. Залежність передається в момент створення. DI може бути реалізований за допомогою установників або передачі параметрів в конструкторі.
3. Аспектно-орієнтоване програмування: це допомагає розрізняти наскрізні функції в додатку.

Архітектура Spring Framework надає 20 модулів, які можна використовувати в залежності від вимог додатка.

Core і Bean забезпечує фундаментальну частину фреймворка, включаючи IoC і DI.

Контейнер ядра (Core Container) можна умовно розбити на кілька субконтейнерів:

1. Модуль Core надає всі основні компоненти Spring Framework. Він включає в себе функції Inversion of Control і Dependency Injection.
2. Модуль Bean пропонує BeanFactory, який являє собою складну реалізацію фабричного шаблону.
3. Модуль Context побудований на міцній основі, що надається модулями Core і Beans, і є середовищем, яке допомагає вам отримати доступ до будь-яких певних і налаштованих об'єктів.
4. Spring Expression Languages (SpEL) модуль пропонує мову виразів для зміни і запиту графа об'єкта під час виконання.

Рівень доступу до даних і інтеграції (Data Access/Integration) складається з модулів JDBC, ORM, OXM, JMS і Transaction:

1. ORM: модуль ORM забезпечує узгодженість/переносимість коду незалежно від технологій доступу до даних. Він буде заснований на концепції об'єктно-орієнтованого відображення.
2. JDBC складається з рівня абстракції JDBC, необхідного для роботи з СУБД.
3. OXM: Object XML Mappers допомагає перетворювати об'єкти в формат XML і навпаки.
4. Модуль JMS пропонує такі функції, як створення і використання повідомлень.
5. Transaction: цей модуль пропонує декларативний і програмний метод управління для реалізації унікальних інтерфейсів і для всіх типів POJO (простий старий об'єкт Java).

Елемент Spring Web:

1. Web: цей модуль використовує слухачів сервлетів і контекст веб-додатка. Він також пропонує функцію веб-орієнтованої інтеграції і функціональність для завантаження файлів з декількох частин.
2. Web-servlet: цей модуль зберігає реалізацію на основі MVC для веб-додатків.
3. Web-Socket: модуль пропонує засновану на WebSocket і двосторонній зв'язок між клієнтом і сервером в веб-додатках.
4. Web-Portlet: цей модуль також називається модулем Spring-MVC-Portlet. Він пропонує портлет на основі Spring і копіює всі функції модуля веб-сервлетів.
5. AOP: мова АОП - корисний інструмент, який дозволяє розробникам додавати в додаток корпоративні функції.
6. Instrumentation: цей модуль пропонує інструментарій класів і реалізації завантажувача. Він використовується для певних серверів додатків.

7. Test: цей модуль забезпечує підтримку тестування компонентів Spring за допомогою інструментів TestNG або JUnit. Він пропонує послідовну завантаження Spring ApplicationContexts і кешування цих контекстів.

Фреймворк Spring MVC пропонує архітектуру model-view-controller, що пропонує компоненти, які допомагають створювати гнучкі і слабо пов'язані веб-додатки. Шаблон MVC дозволяє розділяти різні аспекти програми, пропонуючи при цьому слабкий зв'язок між цими елементами. Дизайн MVC також дозволяє розділити бізнес-логіку, логіку уявлення і логіку навігації. Він також пропонує елегантне рішення для використання MVC в Spring Framework за допомогою DispatcherServlet.

Переваги використання:

1. Spring дозволяє розробникам розробляти програми корпоративного класу за допомогою POJO.
2. Пропонує шаблони для Hibernate, JDBC, JPA і т. д., щоб уникнути написання довгого коду.
3. Надає абстракцію для Java Enterprise Edition (JEE).
4. Можна організувати мультимодульність. Так що, якщо кількість пакетів і класів є значним, вам потрібно тільки вказати, що вам потрібно, і ігнорувати інші.
5. Пропонує декларативну підтримку транзакцій, форматування, перевірки, кешування тощо.
6. Додаток, розроблений з використанням Spring, просте, оскільки код, що залежить від середовища, переміщений в цю структуру.

2.2 Apache Maven

Створення програмного проекту зазвичай складається з таких завдань, як завантаження залежностей, розміщення додаткових jar-файлів в шляху до класів, компіляція вихідного коду в двійковий код, виконання тестів, упаковка

скомпільованого коду в артефакти для розгортання, такі як файли JAR, WAR і ZIP, і розгортання цих артефактів на сервер додатка або в репозиторій. Apache Maven [18] автоматизує ці завдання, зводячи до мінімуму ризик здійснення помилок людьми при створенні програмного забезпечення вручну і відокремлюючи роботу щодо збирання і упаковки нашого коду від роботи зі створення коду.

Конфігурація проекту Maven виконується за допомогою об'єктної моделі проекту (POM), представленої файлом pom.xml. POM описує проект, управляє залежностями і налаштовує плагіни для збирання програмного забезпечення. POM також визначає відносини між модулями багатомодульних проектів.

Розглянемо переваги Maven:

- просте налаштування проекту відповідно до кращих практик: Maven намагається уникнути якомога більшої конфігурації, надаючи шаблони проектів;
- управління залежностями: воно включає автоматичне оновлення, завантаження і перевірку сумісності, а також повідомлення про закриття залежностей;
- ізоляція між залежностями проекту та плагінами: з Maven залежності проекту витягуються з репозиторіїв залежностей, в той час як будь-які залежності плагіна витягуються з репозиторіїв плагінів, що призводить до меншої кількості конфліктів, коли плагіни починають завантажувати додаткові залежності;
- система центрального сховища: залежності проекту можуть бути завантажені з локальної файлової системи або з загальнодоступних репозиторіїв, таких як Maven Central.

2.3 Мікросервіси Java

Мікросервіси Java — це набір програмних додатків, написаних мовою програмування Java, розроблених для обмеженого обсягу, які працюють один з

одним для формування більшого рішення. Кожен мікросервіс має мінімальні можливості для створення дуже модульної загальної архітектури. Архітектура мікросервісів аналогічна виробничій складальній лінії, де кожен мікросервіс схожий на станцію на конвеєрі. Так само, як кожна станція відповідає за одне конкретне завдання, те саме стосується і мікросервісів. Кожна станція та мікросервіс є «експертами» у своїх обов’язках, що сприяє ефективності, послідовності та якості робочого процесу та результатів. Це аналог монолітної програмної програми, яка виконує всі завдання в рамках одного процесу [19].

Мікросервіси являють собою шаблон проектування, в якому кожен мікросервіс є лише одним невеликим елементом більшої загальної системи. Кожна мікросистема виконує конкретне завдання з обмеженим обсягом, яке сприяє досягненню кінцевого результату. Кожне завдання може бути таким простим, як «розрахувати стандартне відхилення набору вхідних даних» або «підрахувати кількість слів у тексті». Ключ до створення мікросервісів — це планування системи для визначення окремих підзадач, а потім написання програм, які вирішують кожну підзадачу. Оскільки кожний мікросервіс має доставити вихідні дані наступному мікросервісу, архітектура мікросервісів часто використовує легку систему обміну повідомленнями для передачі даних.

Ключові особливості:

- сервіси, що «спілкуються» за допомогою REST;
- невеликі, добре розподілені сервіси, що розгортаються окремо;
- сервіси мають бути хмарними.

Мікросервіс визначає підхід до архітектури, яка поділяє програму на пул слабозв’язаних сервісів, які реалізують бізнес-вимоги. Найважливішою особливістю архітектури на основі мікросервісу є те, що вона може виконувати безперервну доставку великої та складної програми. Мікросервіс допомагає поділити програму та створювати логічно незалежні менші програми (рис. 2.1).

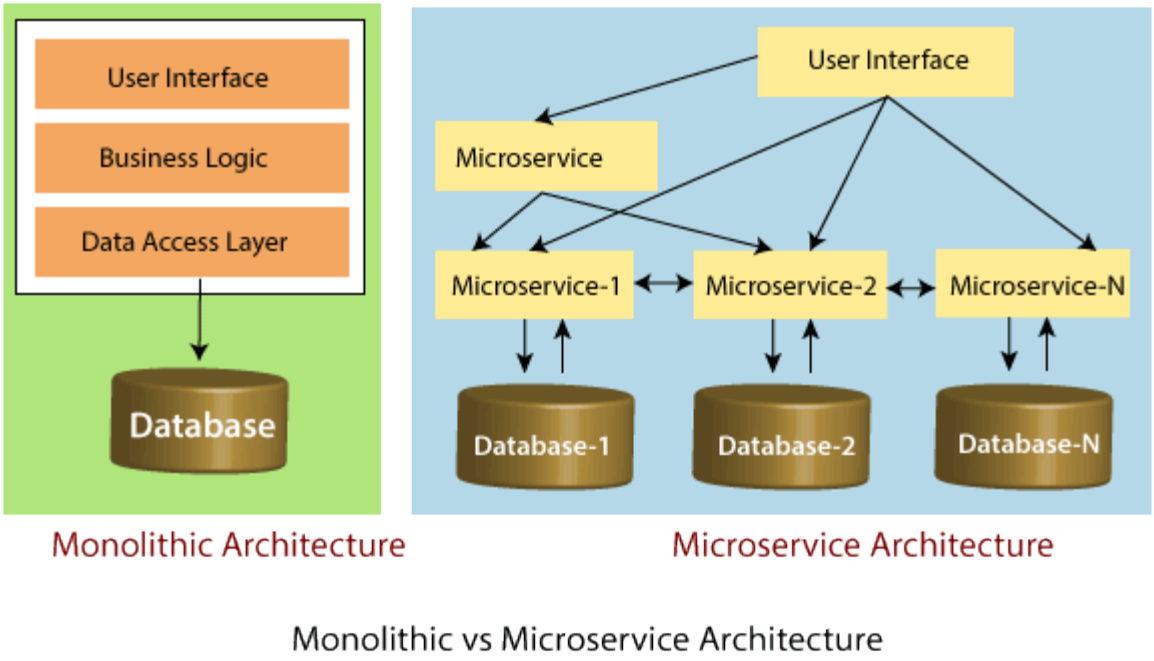


Рисунок 2.1 – Співставлення монолітної та мікросервісної архітектури

Існують наступні принципи мікросервісів:

- принцип єдиної відповідальності;
- моделювання навколо бізнес-домену;
- ізолювання помилок;
- автоматизація інфраструктури;
- незалежне розгортання.

2.4 IntelliJ IDEA

IntelliJ IDEA [20] – зручна середа розробки. Розглянемо основні корисні функції IntelliJ IDEA:

1. Розумне завершення коду: це набір методів, використовуваних розробником для аналізу фрагментів коду. Він підтримує завершення на основі контексту. Він може виявляти і розрізняти велику кількість мов.

2. Аналіз коду «на льоту»: це допомагає поліпшити наш код, визначаючи, чи дійсний вираз чи ні, і видає помилку компіляції, якщо така є. Ця функція завжди буде аналізувати наш вихідний код в фоновому режимі, виявляючи помилки і надаючи можливі поліпшення.
3. Розширений рефакторинг: IntelliJ IDEA має великий вибір можливостей рефакторінга. Це допомагає розробникам швидше та безпечніше реорганізувати його код. IntelliJ IDEA автоматично пропонує рефакторинг. Якщо не пропонується будь-який варіант рефакторінга, ми можемо вибрати його в меню рефакторінга.
4. Виявлення дублікатів: допомагає знайти повторювані фрагменти коду та дає користувачу підказку.
5. Огляд та швидкі виправлення: коли IntelliJ IDEA виявляє помилку, в редакторі з'являється маленька лампочка. Коли ви натискаєте на неї, вона відкриває список дій, які можна зробити, щоб все виправити.
6. Ярлики для всього: IntelliJ IDEA надає поєднання клавіш для більшості завдань, включаючи швидкий вибір і перемикання між вікнами інструментів і редактором.
7. Термінал: IntelliJ IDEA IDE поставляється з вбудованим терміналом. Залежно від платформи ми можемо працювати з командним рядком, PowerShell або bash.
8. Навігація та пошук: це одна з чудових характеристик IntelliJ IDEA, яка допомагає знаходити ресурси і переходити до них. Вона може шукати всі елементи управління, існуючі в середовищі IDE.
9. Підтримка інструментів та фреймворків: IntelliJ забезпечує підтримку безлічі різних інструментів, які допомагають в розробці нашого застосування. Це допомагає розробникам зосередитись і зменшити кількість доробок в середовищі IDE.
10. Сервер-додатки: IntelliJ IDEA підтримує безліч серверів: Tomcat, JBoss, WebSphere, WebLogic, Glassfish тощо.

2.5 Amazon Web Services

Сервіс AWS [21] надається Amazon, який використовує розподілену IT-інфраструктуру для надання різних IT-ресурсів, доступних за запитом. Він надає різні послуги, такі як інфраструктура як послуга (IaaS), платформа як послуга (PaaS) та пакетне програмне забезпечення як послуга (SaaS). AWS надає послуги клієнтам на основі концепції Pay-As-You-Go («Плати тільки за користування»), тобто надає клієнтам послуги у міру необхідності без будь-яких попередніх зобов'язань або авансових вкладень (рис. 2.2).

Розглянемо основні переваги AWS:

1. Гнучкість. Можливо отримати більше часу для основних бізнес-завдань завдяки миттєвій доступності нових функцій і сервісів в AWS. Він забезпечує простий хостинг успадкованих додатків. AWS не вимагає вивчення нових технологій, а міграція додатків на AWS забезпечує передові обчислення і ефективне сховище. AWS також пропонує вибір, чи хочемо ми запускати додатки і сервіси разом чи ні. Ми також можемо запустити частину IT-інфраструктури в AWS, а решту - в центрах обробки даних.

Рисунок 2.2 – «Pay-As-You-Go» сервіси

2. Економічна ефективність. AWS не вимагає попередніх вкладень, довгострокових зобов'язань та мінімальних витрат в порівнянні з традиційною IT-інфраструктурою, яка вимагає величезних вкладень.
3. Масштабованість/еластичність. За допомогою AWS методи автомасштабування та еластичного балансування навантаження автоматично масштабуються в більшу або меншу сторону при збільшенні або зменшенні попиту відповідно. Методи AWS підходять для роботи з непередбачуваними або дуже високими навантаженнями. З цієї причини організації користуються перевагами зниження витрат і підвищення задоволеності користувачів.
4. Безпека. AWS забезпечує клієнтам повну безпеку і конфіденційність. AWS має віртуальну інфраструктуру, яка забезпечує оптимальну доступність, забезпечуючи при цьому повну конфіденційність і ізоляцію своїх операцій. Клієнти можуть розраховувати на високий рівень фізичної безпеки завдяки багаторічному досвіду Amazon в проектуванні, розробці і обслуговуванні великомасштабних операційних центрів IT.

AWS забезпечує три аспекти безпеки: конфіденційність, цілісність і доступність даних користувача.

У процесі розробки доречне використання додаткової бази даних Redshift для бекапів та аналітики даних, а також декілька сервісів для розгортання та виконання рекомендаційного мікросервіса. AWS Redshift — це рішення для керованого сховища даних від Amazon Web Services.

Розглянемо основні переваги AWS Redshift:

- пропонує значне підвищення швидкості запитів;
- зосереджений на простоті використання та доступності;
- забезпечує швидке нарощування з невеликою кількістю ускладнень;
- забезпечує відносно низькі витрати;
- надає надійні інструменти безпеки.

Для того, щоб зняти високе навантаження на мікросервіси під час великої кількості запитів доречне використання SQS (Simple Queue Service).

Amazon SQS — це веб-служба, яка надає доступ до черги повідомлень, яку можна використовувати для зберігання повідомлень під час очікування, поки ваш сервіс їх обробить. Це система розподіленої черги, яка дозволяє програмам веб-служб швидко й надійно ставити в чергу повідомлення, які один компонент програми генерує для використання іншим компонентом, де черга є тимчасовим сховищем для повідомлень, які очікують на обробку.

За допомогою SQS можливо відправляти, зберігати та отримувати повідомлення між програмними компонентами без втрати повідомлень. Можливо розділити компоненти програми, щоб вони могли працювати незалежно, спрощуючи керування повідомленнями між компонентами. Будь-який компонент розподіленої програми може зберігати повідомлення в черзі. Повідомлення можуть містити до 256 КБ тексту в будь-якому форматі, наприклад json, xml тощо. Будь-який компонент програми може пізніше отримати повідомлення програмним шляхом за допомогою Amazon SQS API.

Черга діє як буфер між компонентом, який створює та зберігає дані, і компонент отримує дані для обробки. Це означає, що черга вирішує проблеми, які виникають, якщо продюсер створює роботу швидше, ніж споживач може її обробити, або якщо продюсер або споживач лише періодично підключаються до мережі.

2.6 MongoDB

MongoDB [22], база даних NoSQL, заснована на документах, — це база даних без схем з переконливими характеристиками та помітними функціями, яка дозволяє користувачам запитувати дані найбільш простим і технологічно підкованим способом. База даних, яка підтримується сховищем у стилі JSON, дозволяє користувачам без проблем маніпулювати даними та отримувати доступ до них.

Розглянемо універсальність. З мовою неструктурованих запитів не потрібно створювати таблиці під час роботи з MongoDB. Існує значний ступінь універсальності у зберіганні, управлінні та доступі до даних. Універсальність додає велику перевагу при зберіганні великих даних без категорій.

Розглянемо швидкість. Оскільки немає необхідності створювати таблицю чи схему, швидкість бази даних вражає. Використовуючи MongoDB, швидкість CRUD (створення, читання, оновлення, видалення) вища, ніж у інших баз даних. Запит MongoDB працює в 100 разів швидше, що дозволяє користувачам найшвидше індексувати свій пошук.

Розглянемо легкодоступність. MongoDB підтримує майже всі основні мови програмування C, C++, C#, Java, Node.js, Perl, PHP, Python, Ruby, Scala тощо. MongoDB має драйвери для малопопулярних мов програмування. Можливо розмістити MongoDB на його хмарному сервісі MongoDB Atlas.

У системах, що динамічно зростають, обсяги даних, як правило, швидко збільшуються і рано чи пізно можна зіткнутися з проблемою, коли поточних ресурсів машини не вистачатиме для підтримки нормальної роботи.

Для вирішення цієї проблеми застосовують масштабування. Масштабування буває 2-х видів - горизонтальне та вертикальне. Вертикальне масштабування – це нарощування потужностей однієї машини – додавання CPU, RAM, HDD. Горизонтальне масштабування – додавання нових машин до існуючих та розподіл даних між ними. Перший випадок найпростіший, т.к. не вимагає додаткових налаштувань програми та будь-якої додаткової конфігурації БД, але його недолік у тому, що потужності однієї машини теоретично рано чи пізно впреться в глухий кут. Другий випадок більш складний у конфігурації, але має такі переваги:

- теоретично нескінченне масштабування (машин можна поставити скільки завгодно, питання впирається лише у грошові витрати);
- більша безпека даних (тільки при використанні реплікації) - машини можуть розташовуватися в різних дата центрах (при падінні однієї з них залишаться інші).

Шардування є окремим випадком горизонтального масштабування (рис. 2.3). Суть їх у поділі бази даних окремі частини за певним правилом те щоб кожну їх можна було винести окремий сервер. Однак реплікування даних є обов'язковою вимогою для шардування Mongo. Сервер конфігурації і кожен шард являють собою «репліка-сет».

Це група екземплярів mongod (демон mongoDB), які зберігають однакові набори даних. У наборі копії один вузол - ключовий, який отримує всі операції запису. Всі інші вузли – вторинні, приймають операції з першого, таким чином зберігаючи такі ж записи, як і первинний вузол. Набір копій може мати лише один первинний вузол. У разі відмови інший вузол може зайняти його місце або випадковим чином, або за рішенням сервера-арбітра.

Властивості набору копій (Replica Set):

- будь-який вузол може бути первинним;
- усі операції запису йдуть у первинний вузол;
- кластер складається з N вузлів;
- автоматична відмовостійкість;
- автоматичне відновлення даних;
- автоматичний вибір первинного ключа.

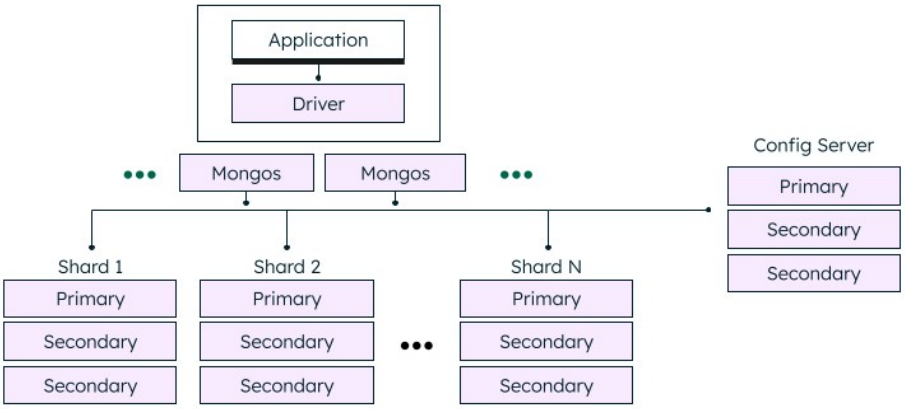


Рисунок 2.3 – Архітектура шардування

2.7 Swagger

Front end та Back end часто розділяють web-ресурс. У такому сценарії дуже важливо мати відповідні специфікації для внутрішніх API. У той же час документація API має бути інформативною, читабельною та легкою для дотримання. Довідкова документація повинна одночасно описувати кожну зміну в API. Доречно використовувати реалізацію Springfox специфікації Swagger [23]. Останню версію можна знайти на Maven Central. Для проектів на основі Spring Boot достатньо додати одну залежність springfox-boot-starter:

```
<dependency>
   <groupId>io.springfox</groupId>
   <artifactId>springfox-boot-starter</artifactId>
   <version>3.0.0</version>
</dependency>
```

Конфігурація Swagger зосереджена навколо компонента Docket:

```
@Configuration
public class SpringFoxConfig {
   @Bean
   public Docket api() {
      return new Docket(DocumentationType.SWAGGER_2)
        .select()
        .apis(RequestHandlerSelectors.any())
        .paths(PathSelectors.any())
        .build();}}
```

Після визначення bean-компонента Docket його метод select() повертає екземпляр ApiSelectorBuilder, який забезпечує спосіб керування кінцевими точками, наданими Swagger.

Swagger UI — це вбудоване рішення, яке значно полегшує взаємодію користувача з документацією API, створеною Swagger. У відповіді Swagger є список усіх контролерів, визначених у програмі. Розгортання кожного методу контролерів надає додаткові корисні дані, такі як статус відповіді, тип вмісту та список параметрів.

2.8 GraphQL

Традиційні REST API працюють із концепцією ресурсів, якими керує сервер. Можливо маніпулювати цими ресурсами деякими стандартними способами, дотримуючись різних дієслів HTTP. Це працює добре, доки API відповідає концепції ресурсу, але швидко розпадається, коли потрібно відхилитися від неї.

Також виникають складності, коли клієнту потрібні дані з кількох ресурсів одночасно, наприклад запит на публікацію в блозі та коментарі. Коли клієнт робить кілька запитів або сервер надає додаткові дані, які не завжди потрібні, це призводить до більшого розміру відповіді.

GraphQL [24] пропонує вирішення обох цих проблем. Це дозволяє клієнту точно вказати, які дані йому потрібні, включаючи навігацію дочірніми ресурсами в одному запиті, та дозволяє виконувати кілька запитів в одному запиті.

2.9 Аналіз методів підтримки прийняття рішень

2.10 Дослідження методів оцінювання рекомендаційного мікросервісу

Для підтримки прийняття рішень лишається ризиком власна програмна розробка рекомендацій, що мають важливе значення та характеризуються високою складністю. Процедура проведення фактичного оцінювання рекомендаційного мікросервісу має бути однозначною та систематизованою, а критерії, покладені в її основу, мають бути чіткими та детальними.

Доцільно проводити оцінювання рекомендаційного мікросервісу стосовно трьох ключових аспектів системи підтримки прийняття рішень (СППР): задач, користувачів та середовища на етапах життєвого циклу СППР. В основу оцінювання покладено три методи: метод витрат/вигід, метод

визначення цінності інформації та моделювання багатоатрибутної корисності [26].

Результати дослідження методу витрат/вигід. Часто витрати та вигоди неправильно або не досить точно прораховані. Метод не придатний для оцінювання рекомендаційного мікросервісу, який має відповідати кориснішим та складнішим критеріям, пов’язаним з вартістю, що він привносить до загального ефекту від прийняття рішень.

Результати дослідження методу визначення цінності інформації. Часто рішення, що ґрунтуються на достовірній або хибній інформації, приймаються за обставин, коли необхідна інформація недоступна, зусилля на одержання необхідної інформації потребують величезних витрат, бракує відомостей про існування корисної інформації, інформація є, але вона подана в неприйнятній формі. Це негативно впливає на визначення релевантності, своєчасності та точності інформації. Визначення цінності ускладнюється неможливістю зводити якісні оцінки до кількісних та зменшувати розбіжності у експертних оцінках. Метод придатний для оцінювання рекомендаційного мікросервісу, але його застосування потребує вирішення проблеми узагальнення великої кількості статистичних даних.

Результати дослідження моделі багатоатрибутної корисності. Для рекомендаційного мікросервісу зростає кількість атрибутів корисності, які отримують як чіткі, так й нечіткі оцінки та утворюють ієрархічну структуру із складними зв’язками. Недоліком є оцінювання атрибутів усіх вищих рівнів, що здійснюється шляхом визначення середніх значень. Як правило, кожному атрибуту надається однакова вага у ієрархії. Модель придатна для оцінювання рекомендаційного мікросервісу, але її застосування потребує вирішення проблеми невизначеності атрибутів, що впливають на загальну корисність.

У даній роботі виявлено основні недоліки існуючих методів оцінювання рекомендаційного мікросервісу. Більшість сучасних ситуацій у сфері надання ріелторських послуг приводять до слабоструктурованих та неструктурованих

задач. Першочергово це приводить до необхідності застосування методів підтримки прийняття рішень з галузі штучного інтелекту з метою вирішення проблеми невизначеності. Таким чином, у даній роботі для оцінювання рекомендаційного мікросервісу запропоновано віддавати перевагу моделі багатоатрибутної корисності.

2.11 Висновки до другого розділу

Проведено аналіз методів підтримки прийняття рішень та методів оцінювання рекомендаційного мікросервісу. Наведено огляд та переваги використання у розробці рекомендаційного мікросервісу наступного стеку технологій: Java, Spring, Maven, MongoDB, AWS Redshift, AWS SQS, Swagger, GraphQL.

РОЗДІЛ 3
РОЗРОБКА РЕКОМЕНДАЦІЙНОГО МІКРОСЕРВІСУ

3.1 Огляд існуючих програмних аналогів

Розглянемо перший альтернативний web-ресурс RocketHomes [27].

RocketHomes надає продавцям можливість продажу житла. Ця компанія є надійним вибором для продажу через агентів, перегляду списків та отримання фінансування для будинків. Реалізовано можливості продавати з агентом за допомогою RocketHomes або продавати самостійно. Якщо прийнято рішення продавати самостійно, то реалізовано підтримку від початку створення оголошення про будинок, розгляду пропозицій від покупців, обговорення їх тощо до завершення успішного продажу.

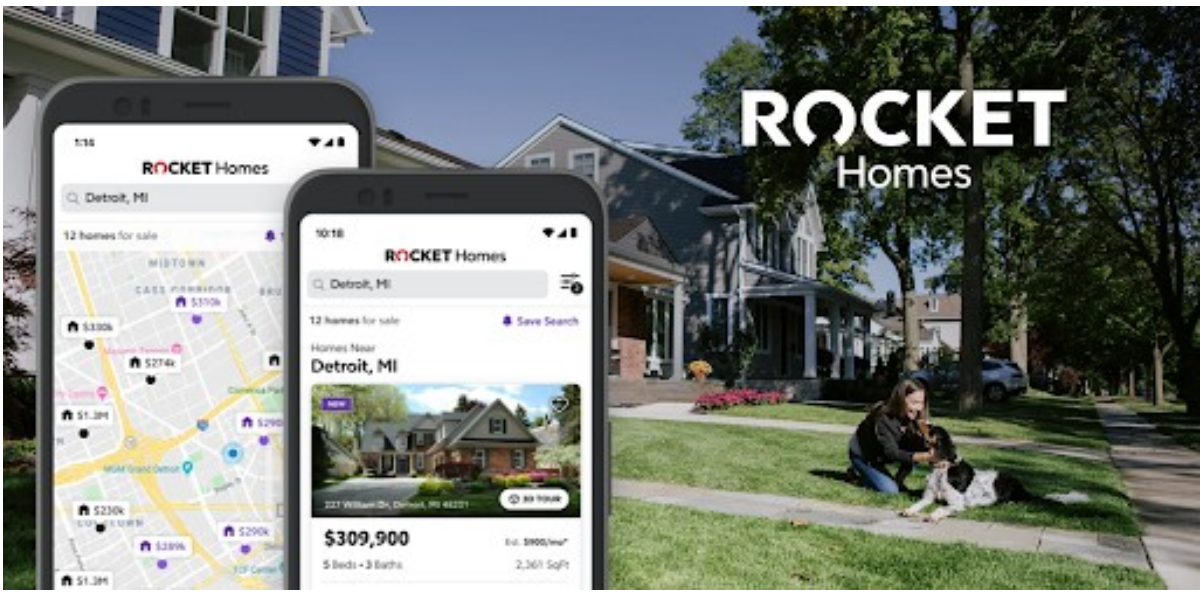


Рисунок 3.1 – Альтернативний web-ресурс RocketHomes

Розглянемо другий альтернативний web-ресурс Xome [28].

Xome − один з найпопулярніших програмних аналогів для продавців житла. Люди можуть не тільки купувати та продавати нерухомість на Xome, але

й отримувати повний досвід роботи з нерухомістю від початку до кінця. До основних належать послуги з іпотеки та прав власності, нотаріальні підписи, оцінка ефективності тощо. Xome використовує аналітику інвесторського рівня, загальнонаціональну мережу провідних агентів та власний Xome Offer Marketplace, щоб революціонізувати процес продажу житла.

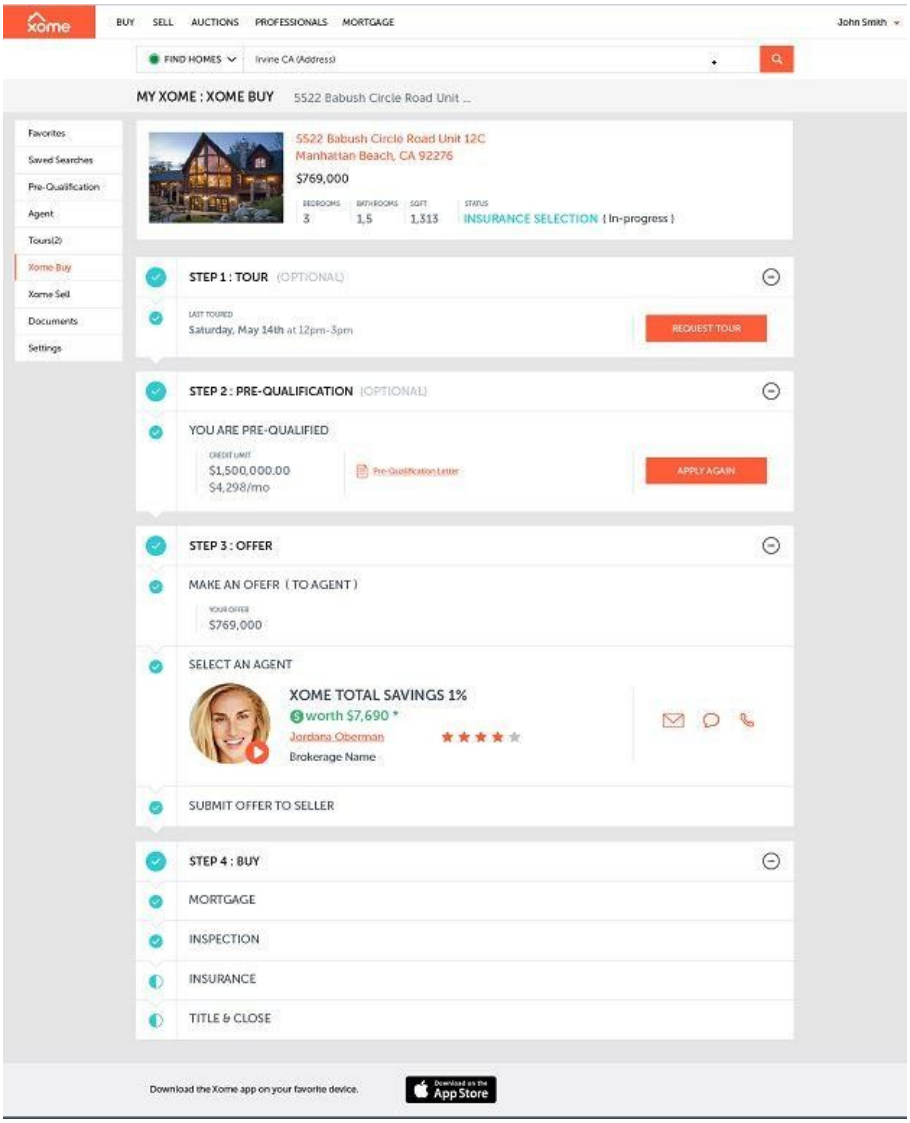


Рисунок 3.2 – Альтернативний web-ресурс Xome

Розглянемо третій альтернативний web-ресурс UpNest [29].

UpNest – це безкоштовна послуга для продавців та покупців житла, що дозволяє знайти найкращих місцевих агентів з нерухомості. Платформа UpNest

дозволяє порівнювати кількох агентів у необхідному регіоні, чим забезпечує можливість порівнювати відгуки, ставки комісії, попередні продажі тощо.

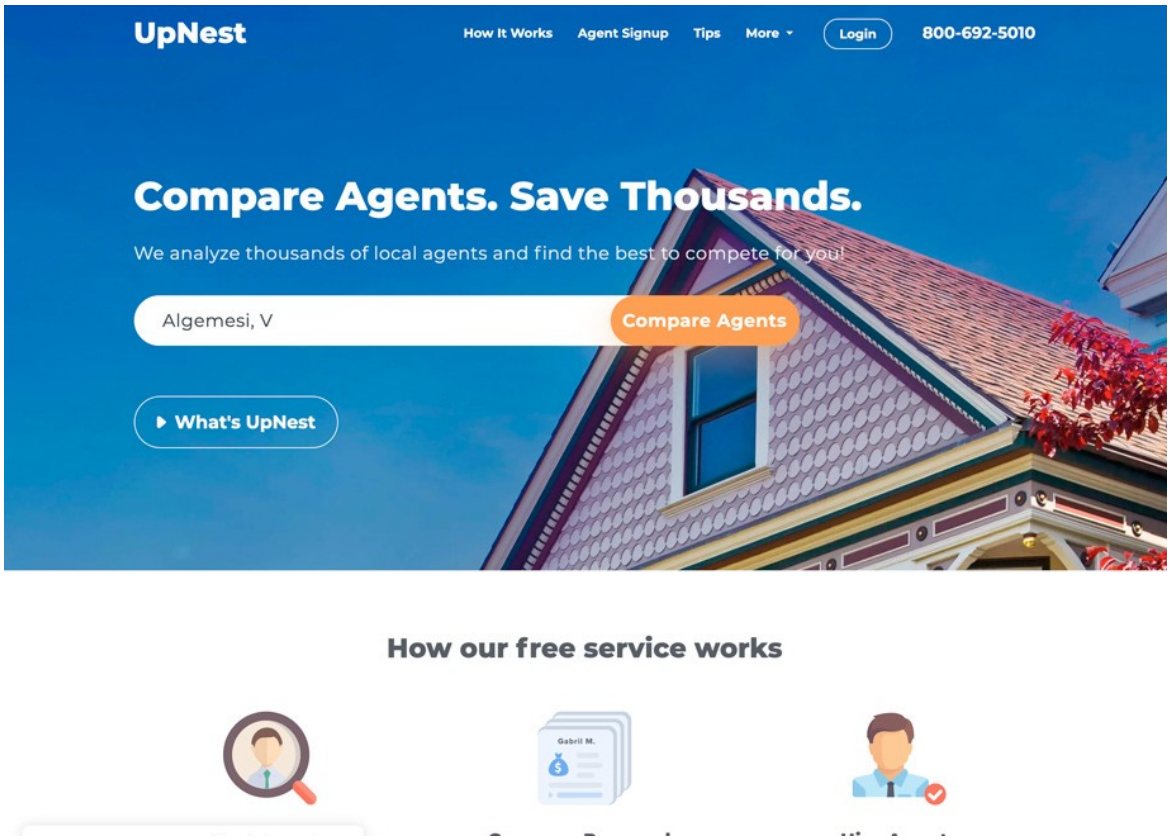


Рисунок 3.3 – Альтернативний web-ресурс UpNest

3.2 Особливості мікросервісної архітектури

Мікросервісна архітектура передбачає, що кожен сервіс є окремим та незалежним, тобто може розгортатися та працювати цілком самостійно. Для створення рекомендаційного алгоритму створюється окремий проект (рис. 3.4). Цей проект розподіляється на два модулі: перший (api) використовується для взаємодії із зовнішнім середовищем (API), другий (application) – уся бізнес логіка, що стосується рекомендацій (тобто прийняття, обробка, обчислення даних та їх зберігання). Модуль API потрібен для того, щоб можна було викликати кінцеві точки (endpoints) з інших сторонніх програм (мікросервісів) просто викликаючи відповідні методи.

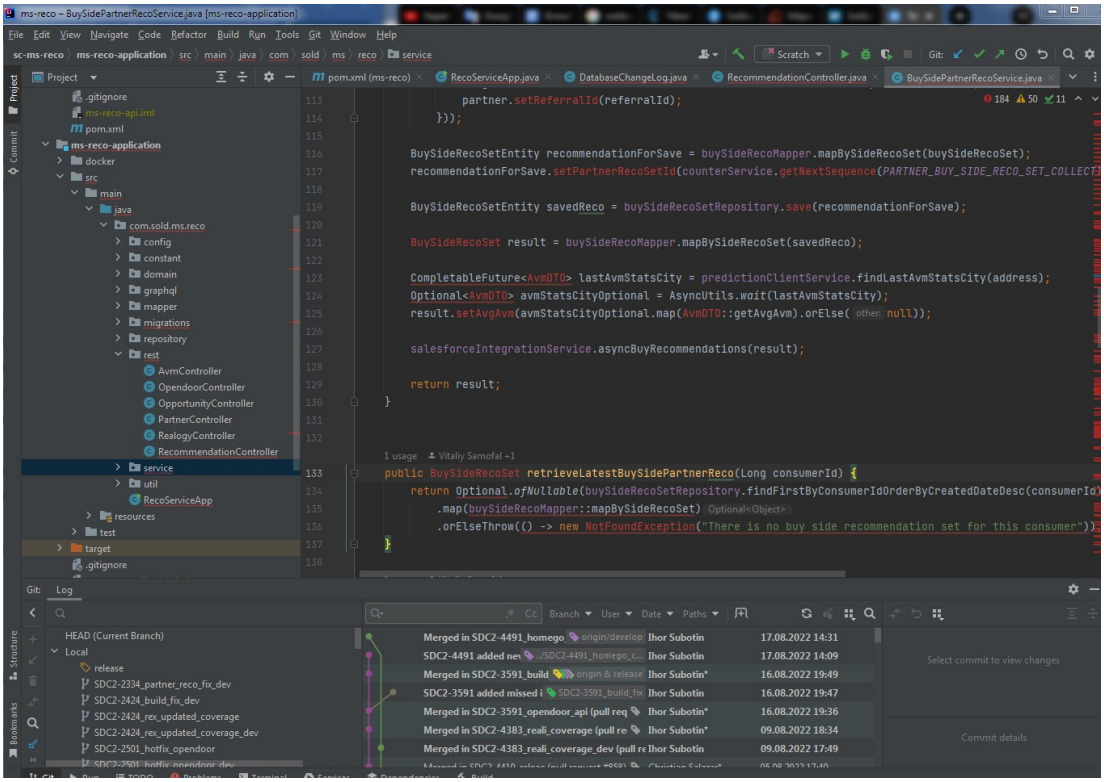


Рисунок 3.4 – Структура проекту (з редактору)

Розглянемо особливості структури Model-View-Controller.

На верхньому рівні знаходиться контролер – клас, в якому прописані всі REST-endpoints, DTO (Data Transfer Object) – об’єкт, що передається у endpoint, GraphQL – клас, в якому також прописані методи взаємодії з API. Далі знаходяться сервіси, де прописана бізнес-логіка. Та на нижчому рівні реалізовано взаємодію з БД (репозиторії, сутності). Також є директорія utils та mapper – використовується для «маппінгу» (перетворення однакових чи дуже схожих об’єктів, зазвичай DTO – сутність – DTO).

Також є окремий мікросервіс, що використовується для виконання певних процесів в зазначений час. Це реалізовано за допомогою бібліотеки Quartz, але не є прямим об’єктом розробки даного проекту. Цей мікросервіс має назву ms-jobs та кожен процес job має своє ім’я, час та логіку.

Більшу частину логіки покрито тестами за допомогою бібліотеки JUnit.

3.3. Діаграми проектування

До існуючої системи надання послуг ріелтора вже закладено декілька ключових сутностей, що задіяні у алгоритмі для формування рекомендацій. Перша сутність consumer (споживач), яку з її властивостями використано як вхідний параметр. Друга сутність відноситься до ріелтора та має два типи – agent або partner. Тому побудовано два типи рекомендацій відповідно. Споживача та агента/партнера поєднує зв'язок, що називається referral (реферал). Список рефералів є вихідними даними сформованої рекомендації. Це означає, що система здібна визначати, яких агентів або партнерів рекомендувати споживачам в залежності від їх властивостей, та повертати список рефералів між ними. Вхідні та віхідні параметри рекомендаційного алгоритму зображено на рис. 3.5.

Однією з найбільш інформативних є побудована діаграма класів (рис. А.1), які розроблено у новому рекомендаційному мікросервісі.

Оскільки процес тісно пов’язаний з існуючими сутностями consumer, agent/parthner, referral, то необхідно дослідити процес створення та оновлення зазначених сутностей (рис. А.2).

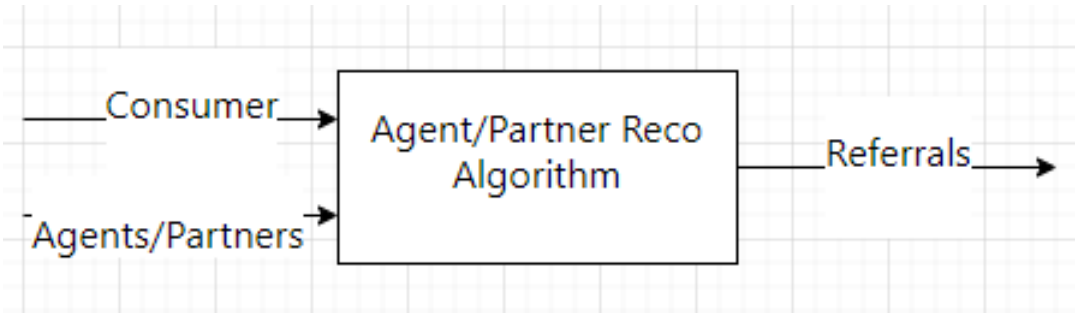


Рисунок 3.5 − Схема рекомендаційного алгоритму

3.4. Опис програмної реалізації

Система має дві категорії сутностей, що допомагають consumer (користувачу) – агент, що є приватною особою, та партнер, що є компанією, за якою співпрацює клієнт. Зважаючи на це, бізнес логіка теж розподілена на дві частини: перша – для агентів, друга – для партнерів.

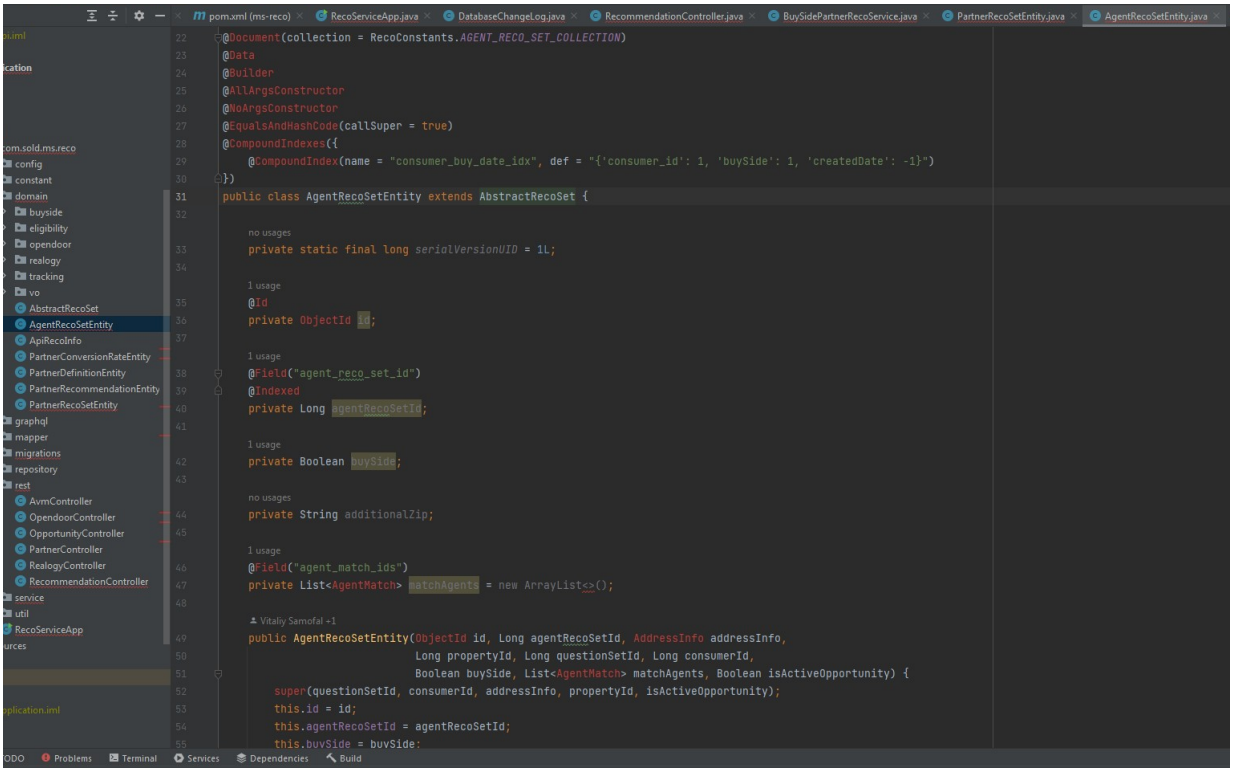


Рисунок 3.6 – Приклад сутності класа AgentReco

Правила у алгоритмі Partner Reco Algorithm потрібні для пошуку відповідних партнерів для споживачів.

У мікросервісі ms-reco є файл (partnersInfo.json) з усіма партнерами та правилами для рекомендацій.

Для того, щоб зміни, внесені до правил партнера, було застосовано, потрібно оновити версію міграції до наступної. Усі зміни зберігаються до бази даних MongoDB, це дозволяє рекомендаційному мікросервісу ними користуватися.

Щоб відключити партнерів, необхідно використати /v1/partner/activate ендпоінт для дезактивації партнера з рекомендацій.

У файлі (partnersInfo.json) можливо оновити поле activePartner до false та запустити міграцію.

Реалізовано можливість створити нове правило, кожне правило має поле «_class» із шляхом до цього правила. Наприклад, як зображено на рис. 3.7.

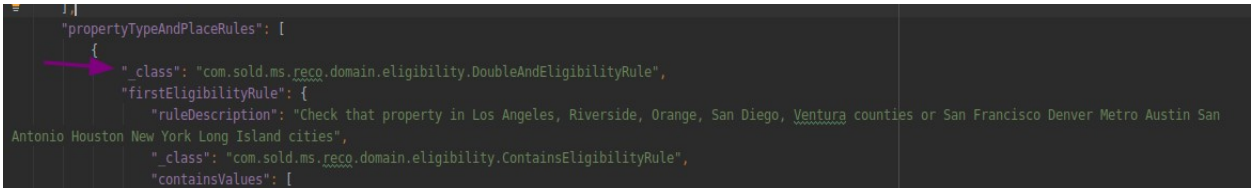


Рисунок 3.7 – Приклад оголошення правила

Кожне правило має спеціальну логіку та включає різні типи даних. Реалізовано можливість створити основне правило, а всередині нього створити правила з реалізацією.

Наприклад, є логіка рекомендувати партнера FIZBER для всіх партнерських рекомендацій. Немає значення, які там правила.

Реалізовано 4 типи партнерів: Agent, NonTraditionalAgent, CashOffer, FSBO. Для кожного споживача після проходження опитування рекомендується партнер відповідного типу партнера.

Реалізовано отримання оцінки для партнерів з файлу selling_types_reasos.json. Існує вага (бал) кожного запитання для кожного типу партнера. Якщо обрано кілька партнерів з однаковим типом партнера, виконується сортування за балом.

Розглянемо основні правила. Якщо партнер відповідає всім правилам, то відбувається рекомендація цього партнера.

1. homeValueRules — правило для перевірки вартості будинку. Приклад avm між 100k та 200k.
2. propertyTypeAndPlaceRules — правило, яке рекомендує партнерам лише відповідні поштові індекси.

3. lotSizeRules — правило, яке дозволяє знаходити партнерів за критеріями до квадратних метрів будинку.
4. yearBuiltRule — правило, яке дозволяє знаходити партнерів за критерієм року побудови будинку.
5. bedroomsRule — правило для перевірки кількості спалень, якщо партнери мають максимальну чи мінімальну кількість спалень.
6. bathroomsRule — правило для перевірки кількості ванних кімнат, якщо партнери мають максимальну чи мінімальну кількість ванних кімнат.
7. partnerFee/buySidePartnerFee − правило дозволяє вибрати відповідного партнера за гонораром.
8. buySidePartner − правило означає, чи буде цей партнер рекомендований клієнтам, які хочуть бути на стороні покупки.

Розглянемо Agent Reco algorithm.

Коли генеруються рекомендації агентів, максимальна рекомендована кількість агентів становить 5. Завжди показано підходящих агентів, якщо SGD Agent, якщо поточний статус не дорівнює «Internal - Test» та «DUPLICATE». Якщо агент не SGD, показано агентів зі статусом PREFERRED_AGENT.

Також реалізовано отримання агентів за рекомендацією з колекції «territory» MongoDB. Там збережено агентів територіальних асоціацій.

Якщо агент, для якого ми створюємо рекомендацію, має транзакцію на поштовий індекс, то виконується спроба його порекомендувати.

Jobs AGENT_SIGN_UP_QUALIFICATION_JOB оновлюють дані, пов’язані з кваліфікацією агентів, а також оновлюють поле zipCodes, яке використовується для виявлення транзакцій (колекція агента, profileInfo -> zipCodes). Job працює кожні 10 хвилин, та Terradatum оновлює QUALIFICATIONS для агентів.

Агенти відфільтровуються із чорного списку та тестувальників. Якщо ім’я або прізвище починається з «test», а електронна адреса має домен «@sold.com», ці агенти вважаються тестовими агентами.

Щоб знайти найкращих агентів, потрібно підрахувати бали.

Спочатку нам потрібно розрахувати точки перетворення. Підраховуються направлення агента, з урахованням AppointmentSetEver, що дорівнює true за agentID.

Якщо кількість зустрічей дорівнює або перевищує 6, а конверсія дорівнює або перевищує 50%, повертається 35 балів. Якщо кількість зустрічей менше 6 і конверсія дорівнює або перевищує 50%, повертається 30 балів. Якщо кількість зустрічей дорівнює або перевищує 6, а конверсія менше 50%, повертається 25 балів. Якщо кількість зустрічей менше 6 і конверсія менше 50%, повертається 20 балів. Інакше 0 балів.

Дані з параграфа (b) плюсуємо (конверсія * 100).

1. Розрахувати точки концентрації. Для розрахунку точок концентрації, якщо кількість транзакцій за поштовим індексом для агента та розділена на загальну суму транзакцій для агента дорівнює або перевищує 20%, а кількість транзакцій дорівнює або перевищує 4, ми повертаємо 15 балів. Інакше повертаємо 0 балів.
2. Обчислити точки об’єму. Якщо агент має понад 10 транзакцій, ми повертаємо 10 балів. Інакше повертаємо 0 балів.
3. Підрахувати бали досвіду. Якщо агент має досвід роботи більше 5 років, ми повертаємо 5 балів. Інакше повертаємо 0 балів.
4. Підсумувати всі бали з кожного абзацу для кожного агента.
5. Відсортувати агентів за балами, максимальна рекомендована кількість агентів становить 5. PLATINUM_AGENTs будуть першими в списку рекомендацій.

Рисунок 3.8 – Типи агентів за їх статусом

Розглянемо статистику агента. У ms-referral (/agent/referrals/get-full-agent-stats) у нас є «ендпоінт», яка повертає повну статистику агента на основі «рефералок» агента.

Поля для відповіді:

- agentId - ідентифікатор агента;
- Zip - поле для отримання статистики за поштовим індексом;
- Passive Referrals Lifetime - знаходить пасивні реферали для агентів за полем 'portalSection' = PASSIVE_REFERRALS, agentId і zipCode за весь час;
- Passive Referrals Date From — знаходить пасивні реферали для агентів за полем 'portalSection' = PASSIVE_REFERRALS, agentId і zipCode за останні дні (із запиту ми отримуємо кількість останніх днів);
- Active Referrals Lifetime - знаходить усі реферали для агентів за полем listingAppointmentSetEver = true, agentId та zipCode за весь час;
- Active Referrals Date From - знаходить усіх рефералів для агентів за полем listingAppointmentSetEver = true, agentId та zipCode за останні дні (із запиту ми отримуємо кількість останніх днів);
- Listings Count - знаходить усіх рефералів для агентів за конкуруючими статусами, ідентифікатором агента та поштовим індексом за весь час;

- Sales count – знаходить рефералів для агентів за полем «portalSection» = CLOSED, listingAppointmentSetEver = true, friendlyStatus = «SOLD», agentId та zipCode;
- Avg Monthly Referral Volume - знаходить усіх рефералів за ідентифікатором агента та поштовим індексом за весь час.

Ці дані використовуються для сторінки інструмента ціноутворення та SGD Zip Sniper Tool, що показує статистику агентів.

Розглянемо SGD Інформацію/Статистику. Максимальна кількість місць для території (поштовий індекс) становить 2. Якщо на території вже є 2 агенти SGD, вони можуть додати їх до списку очікування, і їм сповіщення буде надіслано сповіщення, коли на території з’являться місця. Ціна залежить від кількості «лідів» на території. Колекція zip_tier_price.

Tier	Leads/mo	Base Price
6	13+ leads/mo	$120
5	9-12 leads/mo	$100
4	6-8 leads/mo	$75
3	4-5 leads/mo	$65
2	2-3 leads/mo	$45
1	0-1 leads/mo	$30

Рисунок 3.9 – Ціна рівня за кількістю «лідів»

Колекція zip_prioritization_collection — колекція, де міститься інформація про територію (поштовий індекс (zip), місто, штат, кількість потенційних клієнтів, доступний zip, рівень zip). Ця колекція оновлюється щодня о 09:30:00. Назва job: GUARANTEED_DISPLAY_ZIP_PRIORITIZATION_JOB. Ця job оновлює інформацію про кількість потенційних клієнтів, доступний zip, рівень zip у колекції zip_prioritization_collection.

Розглянемо платежі. Усі збори зберігаються до колекції charge_data (agent). Для оплати використовується сервіс Salesforce blackthorn. Потрібно перевірити, чи є вільні слоти на території, яку хоче купити агент (максимум 2

на територію). Після успішного придбання агентом територій, агент оновлюється у Salesforce (рис. 3.10), zip_prioritization_collection та колекції агентів.

Розглянемо Discount and Promo. У колекції promo ми зберігаємо promoCode. Кожен промокод може мати поле discountOption, де знижка залежить від кількості місяців, які агент хоче купити, або знижки за замовчуванням. Агенти також можуть отримати знижку, якщо агенти купують багато поштових індексів.

10+ zips = 15%, 8+ zips = 10%. Агенти отримують знижку, якщо купують більше 1 місяця. 3 місяці = 10%, 6 місяців = 20%, 12 місяців = 35%.

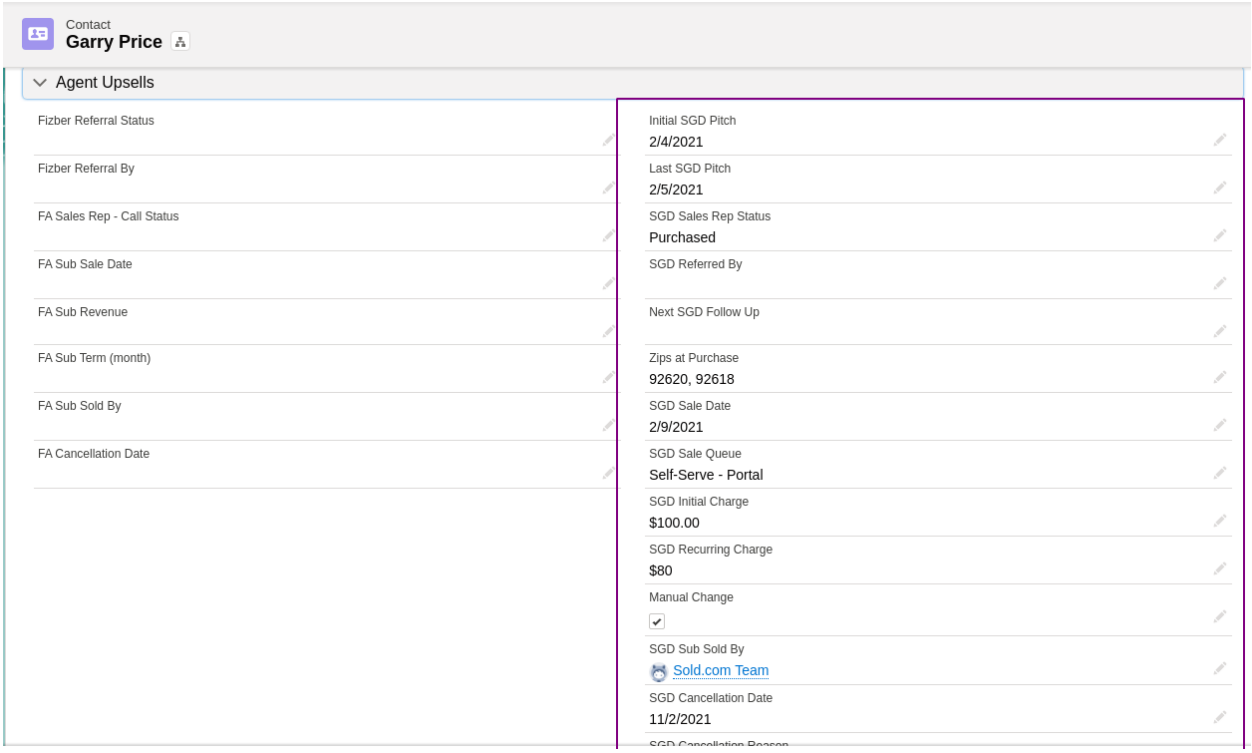
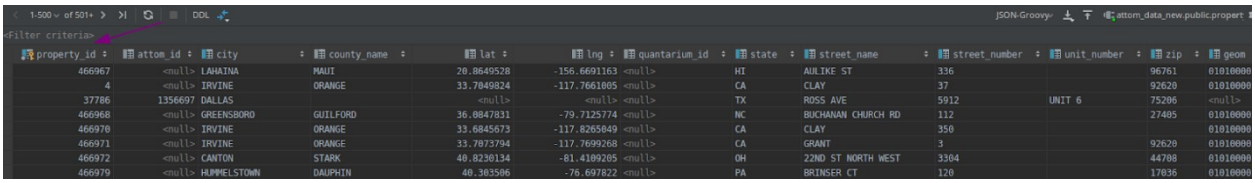


Рисунок 3.10 – Приклад агента, якого синхронізовано з Salesforce

Розглянемо дані про поштові індекси. Реалізовано мікросервіс predictions. Він підключається до бази даних PostgreSQL і має таблицю all_us_zipcodes_2. Там зберігається уся інформація про поштові індекси. Для пошуку найближчої адреси використовується розширення PostGIS, яке дозволяє створювати запити про місцезнаходження. Таблиця marketreport_new зберігає інформацію про

ринок за поштовим індексом, штатом, роком. Ці дані використовуються для статистики повернення ринку за поточний та минулий рік, а також для статистичних даних по регіонах.

Розглянемо таблицю property, де зберігаються усі адреси. Створюється нова властивість після нормалізації адреси, якщо не знайдено її в цій таблиці, інакше повертається властивість із таблиці.



property_id	attom_id	city	county_name	lat	lng	quantarium_id	state	street_name	street_number	unit_number	zip	geom
466967	<null>	LAHAINA	MAUI	20.8649528	-156.6691163	<null>	HI	AULIKE ST	336		96761	01010000
4	<null>	IRVINE	ORANGE	33.7049824	-117.7661005	<null>	CA	CLAY	37		92620	01010000
37786	1356697	DALLAS		<null>	<null>	<null>	TX	ROSS AVE	5912	UNIT 6	75206	<null>
466968	<null>	GREENSBORO	GUILFORD	36.0847831	-79.7125774	<null>	NC	BUCHANAN CHURCH RD	112		27405	01010000
466970	<null>	IRVINE	ORANGE	33.6845673	-117.8265049	<null>	CA	CLAY	350			01010000
466971	<null>	IRVINE	ORANGE	33.7073794	-117.7699268	<null>	CA	GRANT	3		92620	01010000
466972	<null>	CANTON	STARK	40.8230134	-81.4109205	<null>	OH	22ND ST NORTH WEST	3304		44708	01010000
466979	<null>	HUMMELSTOWN	DAUPHIN	40.303506	-76.697822	<null>	PA	BRINSER CT	120		17036	01010000

Рисунок 3.11 – Приклад таблиці property

Розглянемо конверсію. Для оновлення conversionRate та listingAppointmentsAmount у агентах використовується планувальник (AGENT_CONVERSION_RATE_JOB). Коли обчислюється тип агента, використовується conversionRate та listingAppointmentsAmount. Ця job виконується о 03:00:00 ранку кожні 3 дні, починаючи з 1 числа, кожного місяця. listingAppointmentsAmount, яке отримується з колекції agent_referral (враховує agent referral за ідентифікатором агента, а поле Pc Set Ever має значення true). conversionRate отримує готові списки для агента та ділить їх на listingAppointmentsAmount (коефіцієнт конверсії формули sum(completedListings) / sum(кількість рефералів із appointmentSetEver - true)). Ці дані обновляються у колекції агентів. Використовується conversionRate, щоб отримати рейтинг агента (тип агента).

Розглянемо ранг агента (табл. 3.1):

- якщо агент sgd і конверсія дорівнює або перевищує 80% - PLATINUM_AGENT;
- якщо агент sgd - FEATURED_AGENT;
- якщо конверсія агента дорівнює або перевищує 80% - GOLD_AGENT;

– інакше NORMAL_AGENT.

Конверсію використано у алгоритмі Agent Reco для розрахунку балів, а також для рейтингу агента.

Таблиця 3.1 – Умови для визначення рангу агента

	Rate (conversion equal or more 80%)	Featured (featureTerritory is greater than 0)	Featured & Rate
NORMAL_AGENT	false	false	false
PLATINUM_AGENT	true	true	true
FEATURED_AGENT	false	true	false
GOLD_AGENT	true	false	false

Незважаючи на те, що розробка стосувалася переважно Back end, Front end отримує та відображає розраховані дані. Далі наведено показові скріншоти того, які рекомендації сформовано для користувача. Це приклади.

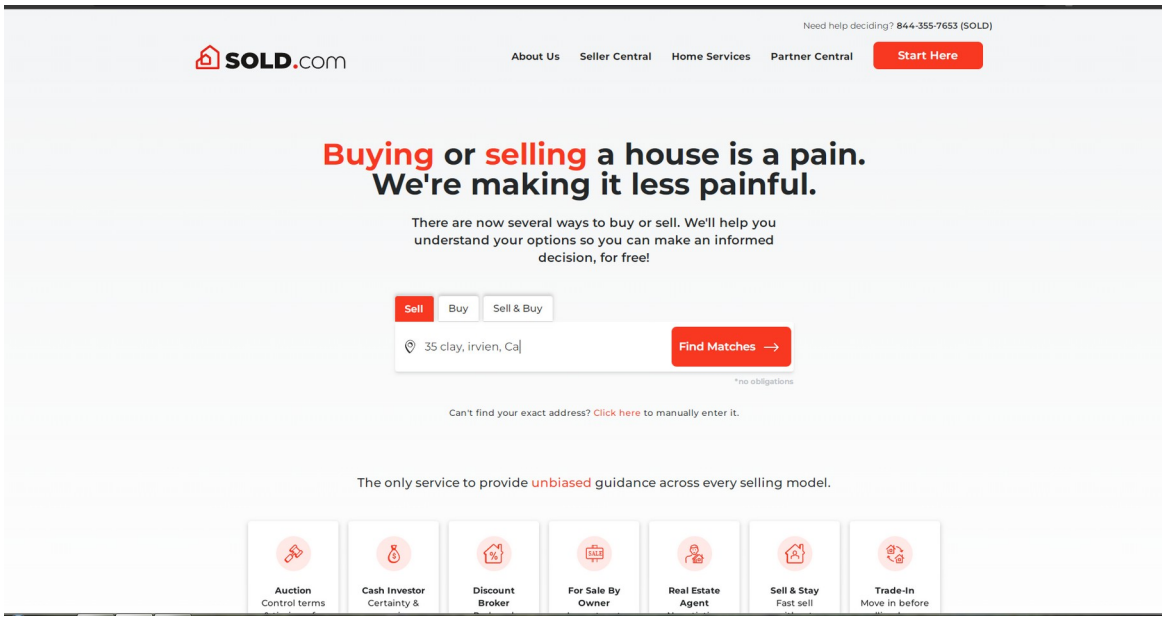


Рисунок 3.12 – Головна сторінка для переходу до опитування

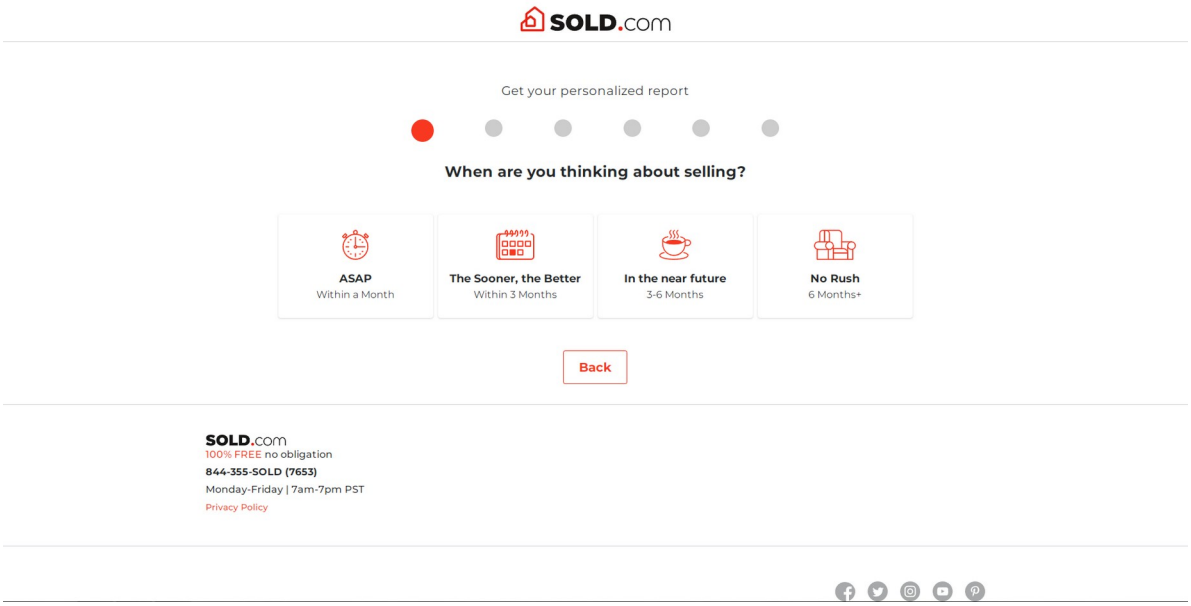


Рисунок 3.13 – Приклад сторінки опитування 1

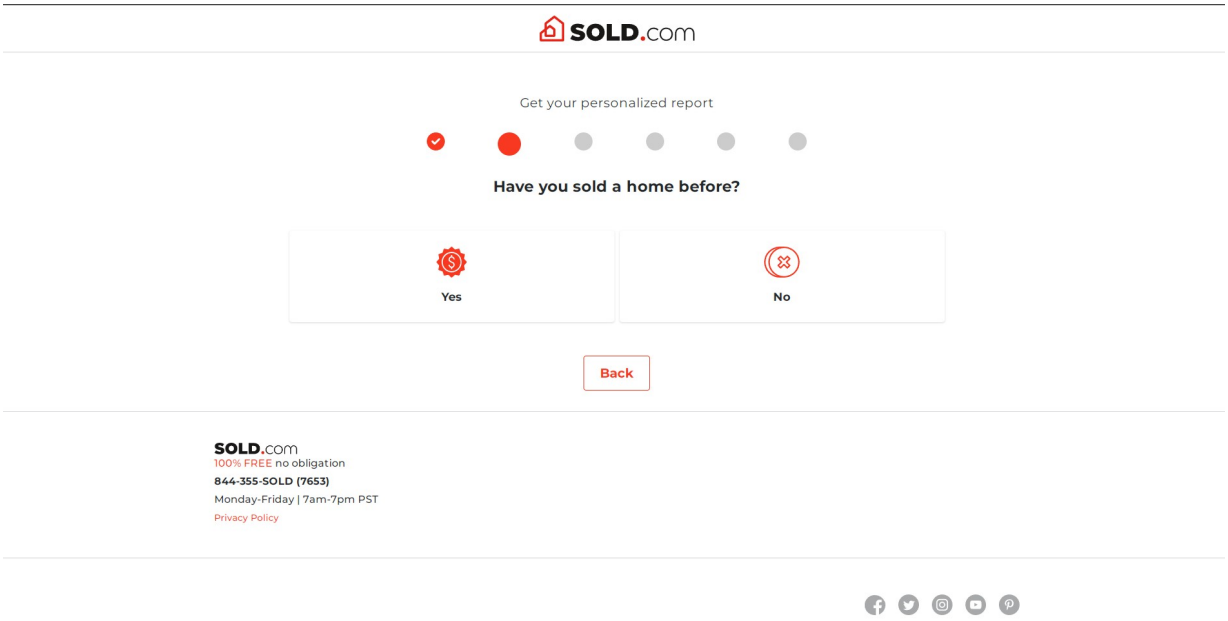


Рисунок 3.14 – Приклад сторінки опитування 2

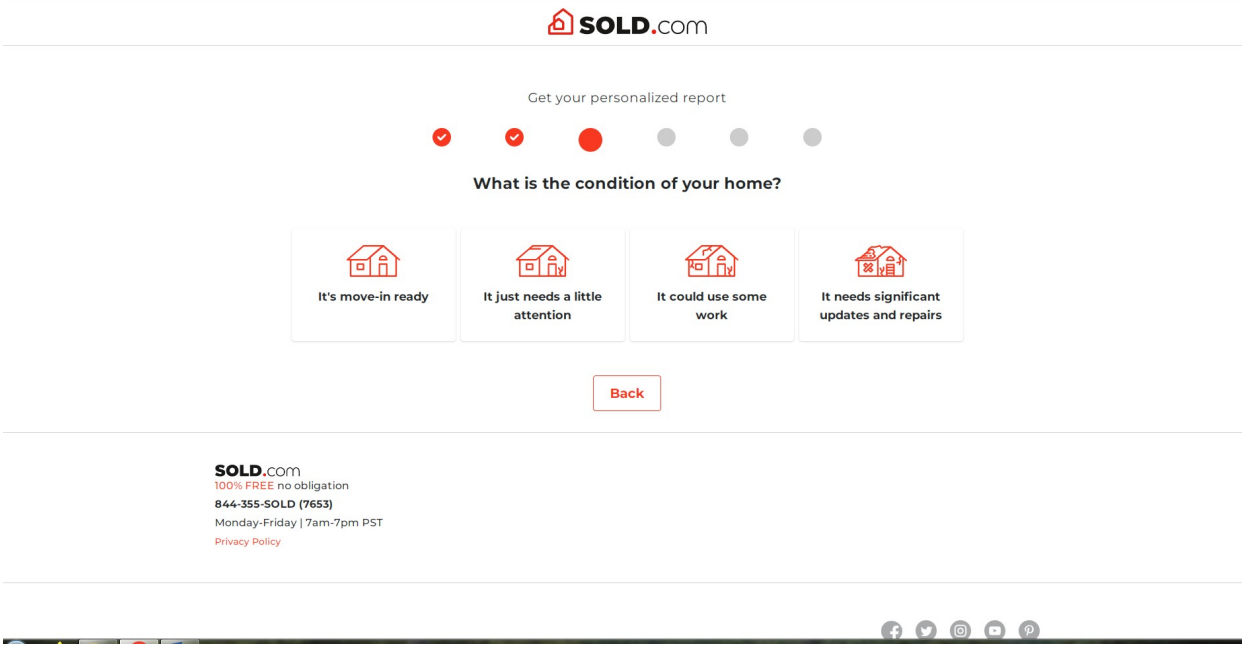


Рисунок 3.15 – Приклад сторінки опитування 3

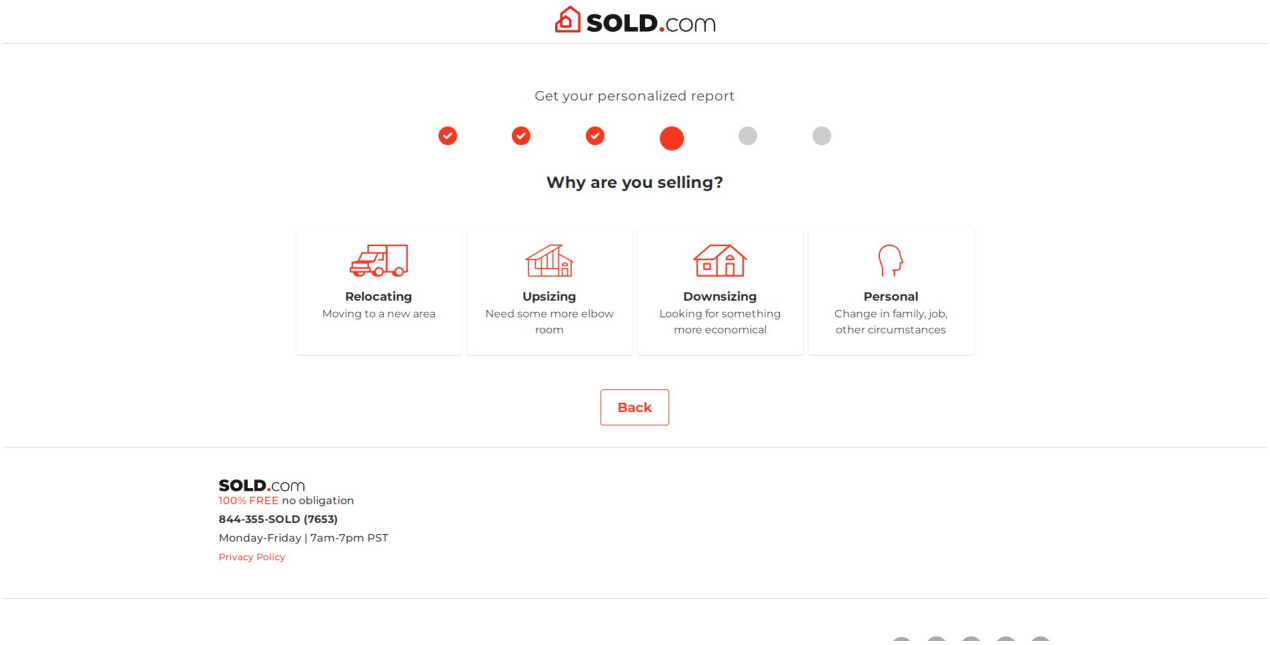


Рисунок 3.16 – Приклад сторінки опитування 4

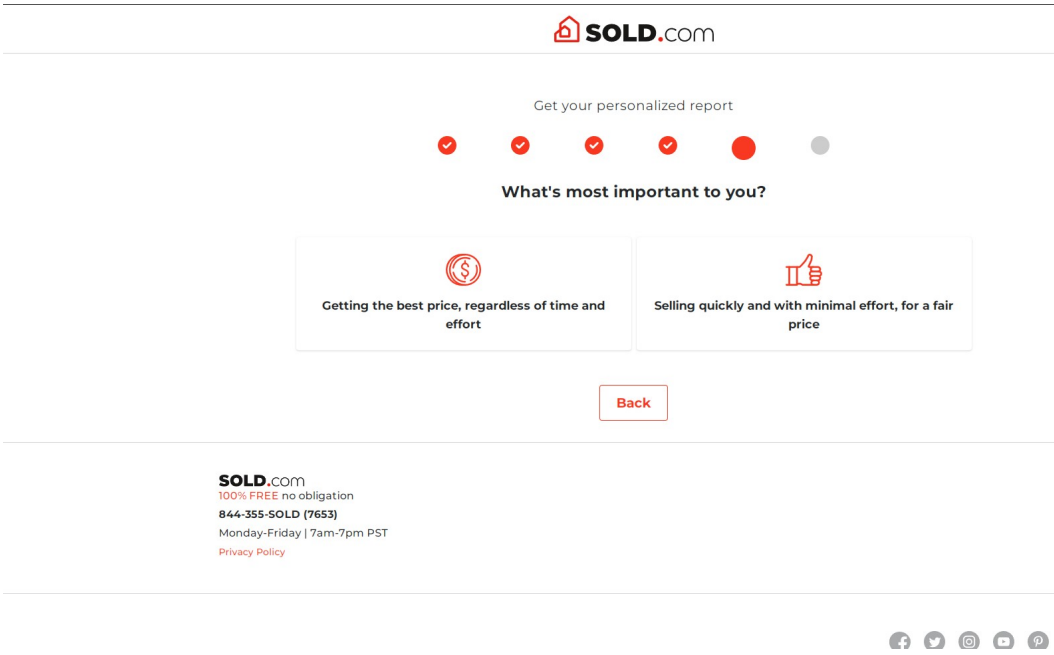


Рисунок 3.17 – Приклад сторінки опитування 5

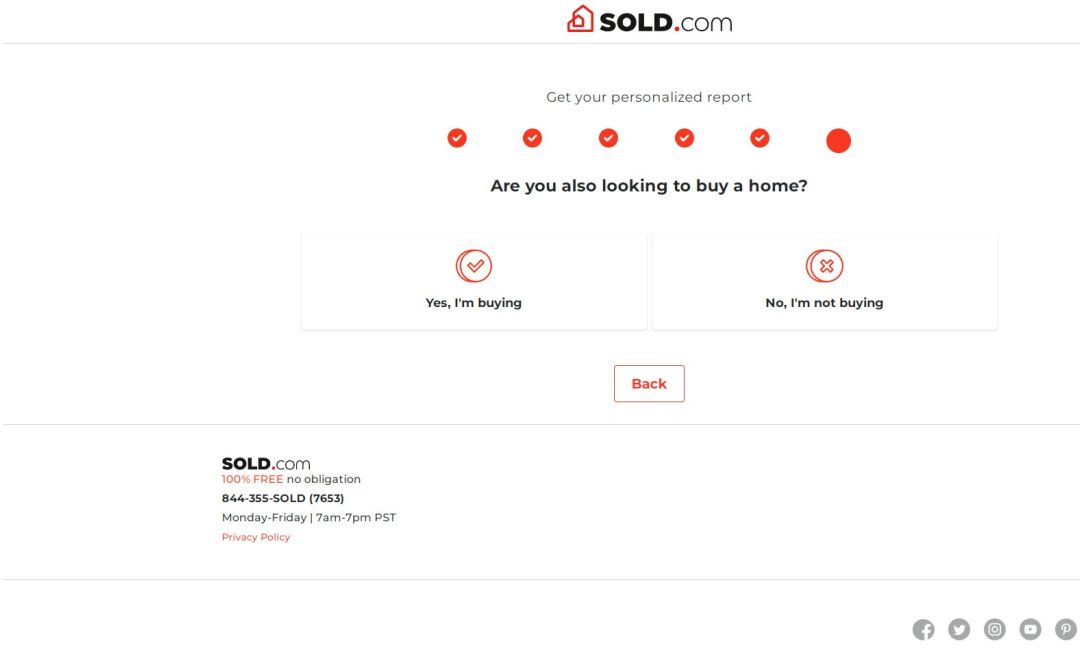


Рисунок 3.18 – Приклад сторінки опитування 6

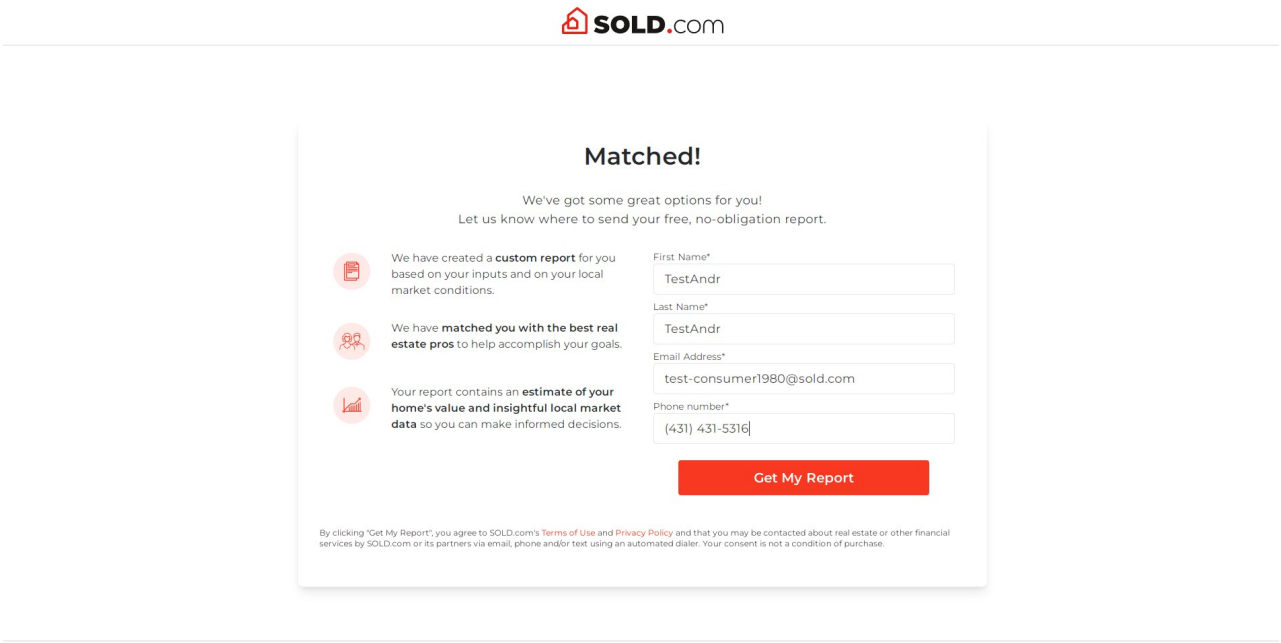


Рисунок 3.19 – Приклад запиту персональних даних

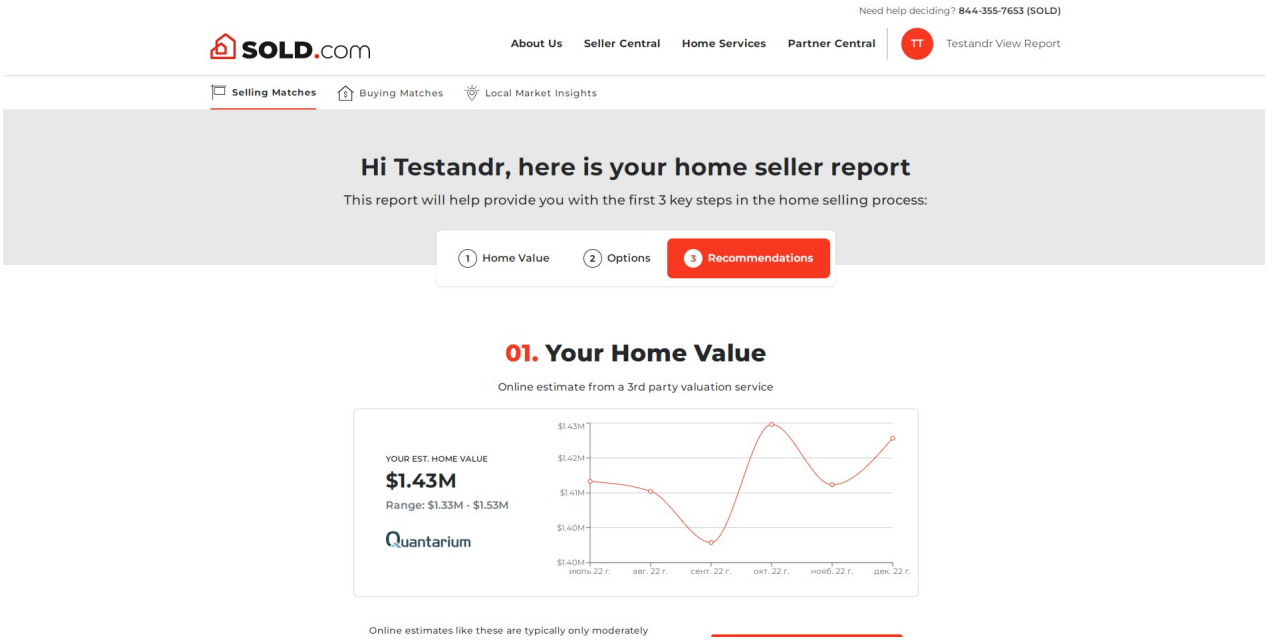


Рисунок 3.20 – Приклад сторінки з рекомендацією. Оціночна вартість

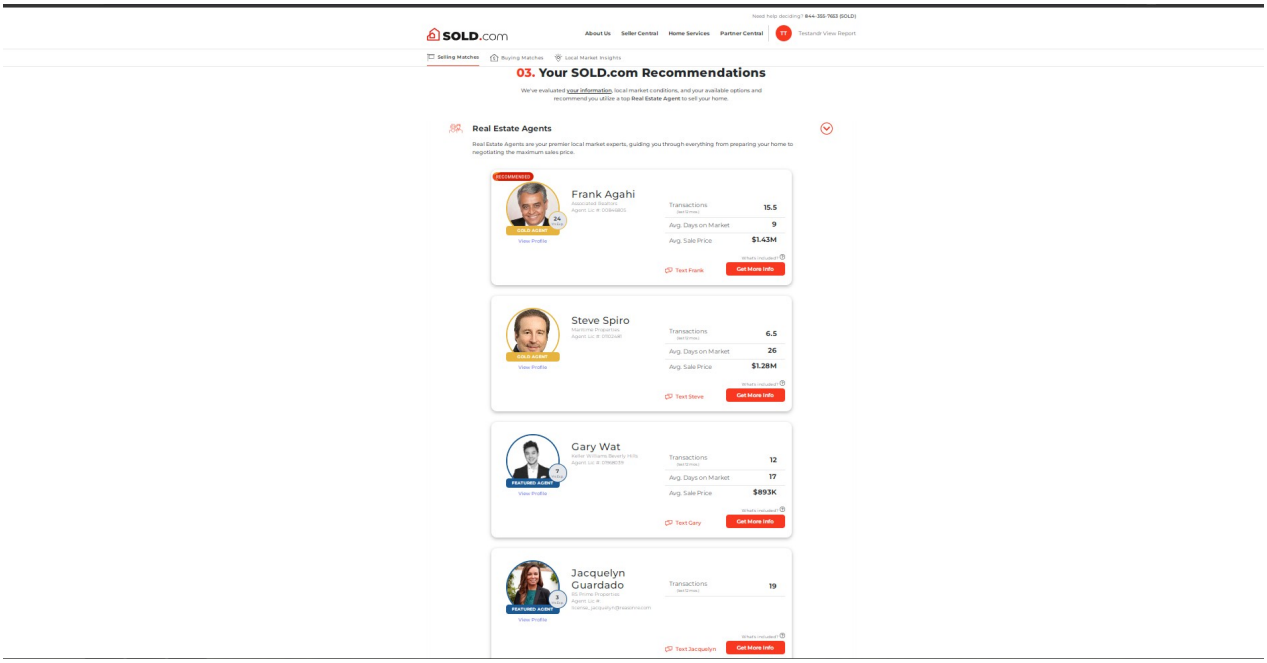


Рисунок 3.21 – Приклад сторінки з рекомендацією. Перелік агентів

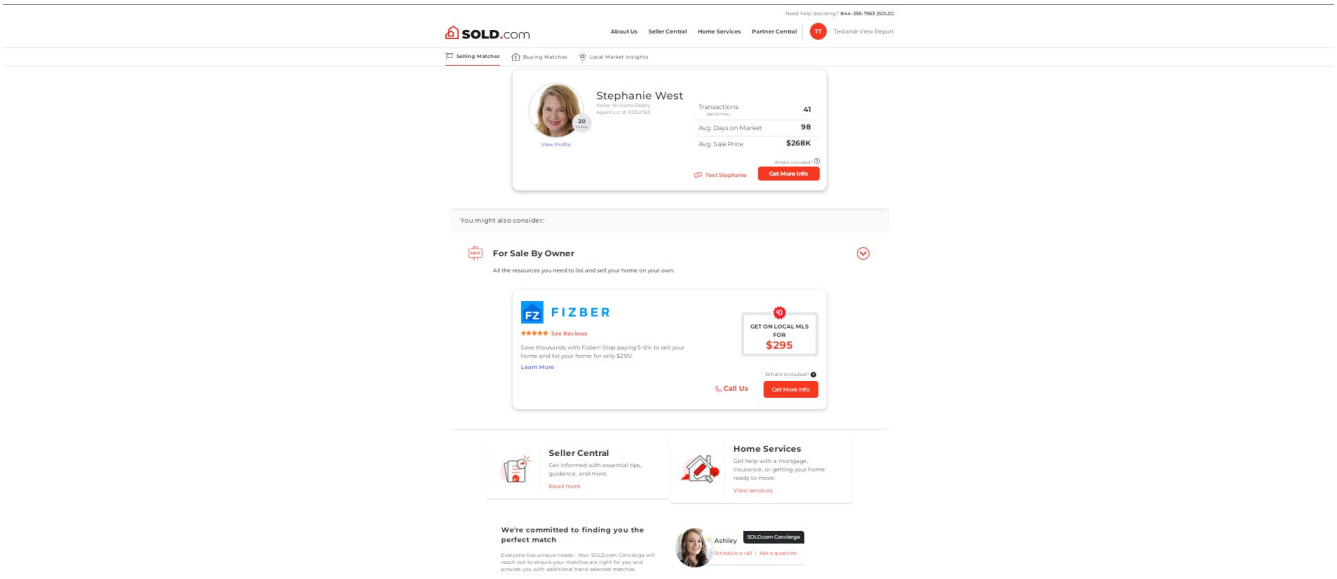


Рисунок 3.22 – Приклад сторінки з рекомендацією. Перелік партнерів

3.5. Висновок до третього розділу

Виконано огляд популярних програмних аналогів. Наведено опис специфіки архітктури та спроектовано діаграми рекомендаційного

мікросервісу. Описано програмну реалізацію рекомендаційного мікросервісу як для розробника, так і для користувача. Наведено приклади сторінок системи надання послуг ріелтора після інтеграції рекомендаційного мікросервісу.

ВИСНОВКИ

Внаслідок виконання даної магістерської дипломної роботи отримано наступні результати:

1. У результаті аналізу сучасного стану сфери купівлі/продажу нерухомості обгрунтовано доцільність розробки та програмної реалізації рекомендаційного алгоритму для надання послуг ріелтора.
2. У результаті дослідження методів підтримки прийняття рішень розроблено рекомендаційний алгорим для надання послуг ріелтора.
3. Розроблено базу даних рекомендацій з урахуванням виявлених критеріїв, що впливають на прийняття рішень користувачами системи надання послуг ріелтора.
4. На основі запропонованого рекомендаційного алгоритму розроблено рекомендаційний мікросервіс.
5. Розроблений рекомендаційний мікросервіс інтегровано до системи надання послуг ріелтора, що дозволяє підвищити швидкість та якість обслуговування.

ПЕРЕЛІК ПОСИЛАНЬ

1. Машинне навчання і бізнес | DOU [Електронний ресурс] – Режим доступу: https://dou.ua/forums/topic/40114/ − Дата доступу: 18.12.22
2. Data Mining and Analysis − Fundamental Concepts and Algorithms | DOU [Електронний ресурс] – Режим доступу: https://dou.ua/calendar/16614/?from=fb − Дата доступу: 10.12.22
3. U.S. Real Estate Market [Електронний ресурс] – Режим доступу: https://www.grandviewresearch.com/services/market-research-reports − Дата доступу: 10.11.22
4. COVID-19. Вплив на глобальний ринок нерухомості [Електронний ресурс] – Режим доступу: https://propertytimes.com.ua/retail_property/covid19_vpliv_na_globalniy_rinok_neruhomosti − Дата доступу: 08.11.22
5. Національна Асоціація Ріелторів (NAR) [Електронний ресурс] – Режим доступу: https://ua.nesrakonk.ru/national-association-of-realtors/ − Дата доступу: 18.10.22
6. Alibaba Group Annual Report 2021 [Електронний ресурс] – Режим доступу: https://doc.irasia.com/listco/hk/alibabagroup/annual/2021/ar2021.pdf − Дата доступу: 19.10.22
7. Apartment List's 2022 Millennial Homeownership Report [Електронний ресурс] – Режим доступу: https://www.apartmentlist.com/research/millennial-homeownership-2022 − Дата доступу: 19.10.22
8. Compound Annual Growth Rate (CAGR). Formula and Calculation [Електронний ресурс] – Режим доступу: https://www.investopedia.com/terms/c/cagr.asp − Дата доступу: 10.10.22

9. Personalized Product Recommendations [Електронний ресурс] – Режим доступу: https://www.barilliance.com/product-recommendations-engine/ − Дата доступу: 10.10.22
10. Research & Advisory [Електронний ресурс] – Режим доступу: https://www.gartner.com/en/product-management/product-decisions − Дата доступу: 10.10.22
11. SOLD [Електронний ресурс] – Режим доступу: https://www.sold.com − Дата доступу: 10.10.22
12. Herbert Schildt. Java: A Beginner's Guide, Eighth Edition, McGraw-Hill, 2018, 720 pp.
13. Bruce Eckel. Thinking in Java, Prentice Hall, 2006, 1482 pp.
14. Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship, Pearson Education, 2008, 464 pp.
15. Craig Walls. Spring in Action, Sixth Edition, Simon and Schuster, 2022, 520 pp.
16. Spring [Електронний ресурс] − Режим доступу: https://spring.io/ − Дата доступу: 12.09.22
17. Stackoverflow [Електронний ресурс] − Режим доступу: https://stackoverflow.com/ − Дата доступу: 01.09.22
18. Maven [Електронний ресурс] − Режим доступу: https://maven.apache.org/ − Дата доступу: 01.09.22
19. REST API [Електронний ресурс] − Режим доступу: https://restfulapi.net/ − Дата доступу: 01.09.22
20. Elastic [Електронний ресурс] − Режим доступу: https://www.elastic.co/guide/index.html − Дата доступу: 15.09.22
21. AWS [Електронний ресурс] − Режим доступу: https://aws.amazon.com/ − Дата доступу: 15.09.22
22. MongoDB [Електронний ресурс] − Режим доступу: https://www.mongodb.com/use-cases − Дата доступу: 15.09.22

23. Swagger [Електронний ресурс] − Режим доступу: https://swagger.io/resources/open-api/ − Дата доступу: 15.09.22
24. GraphQL [Електронний ресурс] − Режим доступу: https://graphql.org/learn/ − Дата доступу: 25.09.22
25. Ситник В.Ф. Системи підтримки прийняття рішень: Навч. посіб. — К.: КНЕУ, 2004. — 614 с.
26. Суботін І.О. «Дослідження методів оцінювання програмного забезпечення СППР», Збірник тез XXIV науково-практичної студентської конференції ЗІЕІТ, Запоріжжя, ЗІЕІТ, 2022 - 89 с.
27. Rockethomes [Електронний ресурс] − Режим доступу: https://www.rockethomes.com/ − Дата доступу: 27.10.22
28. Xome [Електронний ресурс] − Режим доступу: https://www.xome.com/ − Дата доступу: 27.10.22
29. UpNest [Електронний ресурс] − Режим доступу: https://www.upnest.com/ − Дата доступу: 27.10.22
30. Стандарт підприємства. «Методичні вказівки до виконання кваліфікаційних робіт (випускних, дипломних, магістерських). Основні вимоги». СТП 29-2016.
31. ДСТУ 3008-2015. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення / А. Стогній (керівн. розроб.). – Вид. офіц. – [Чинний від 2015-06-22]. – К. : ДП «УкрНДНЦ», 2016. – 26 с. – (Державний стандарт України)

ДОДАТКИ

ДОДАТОК А
ДІАГРАМИ ПРОЕКТУВАННЯ

Рисунок А.1 – Схема класів

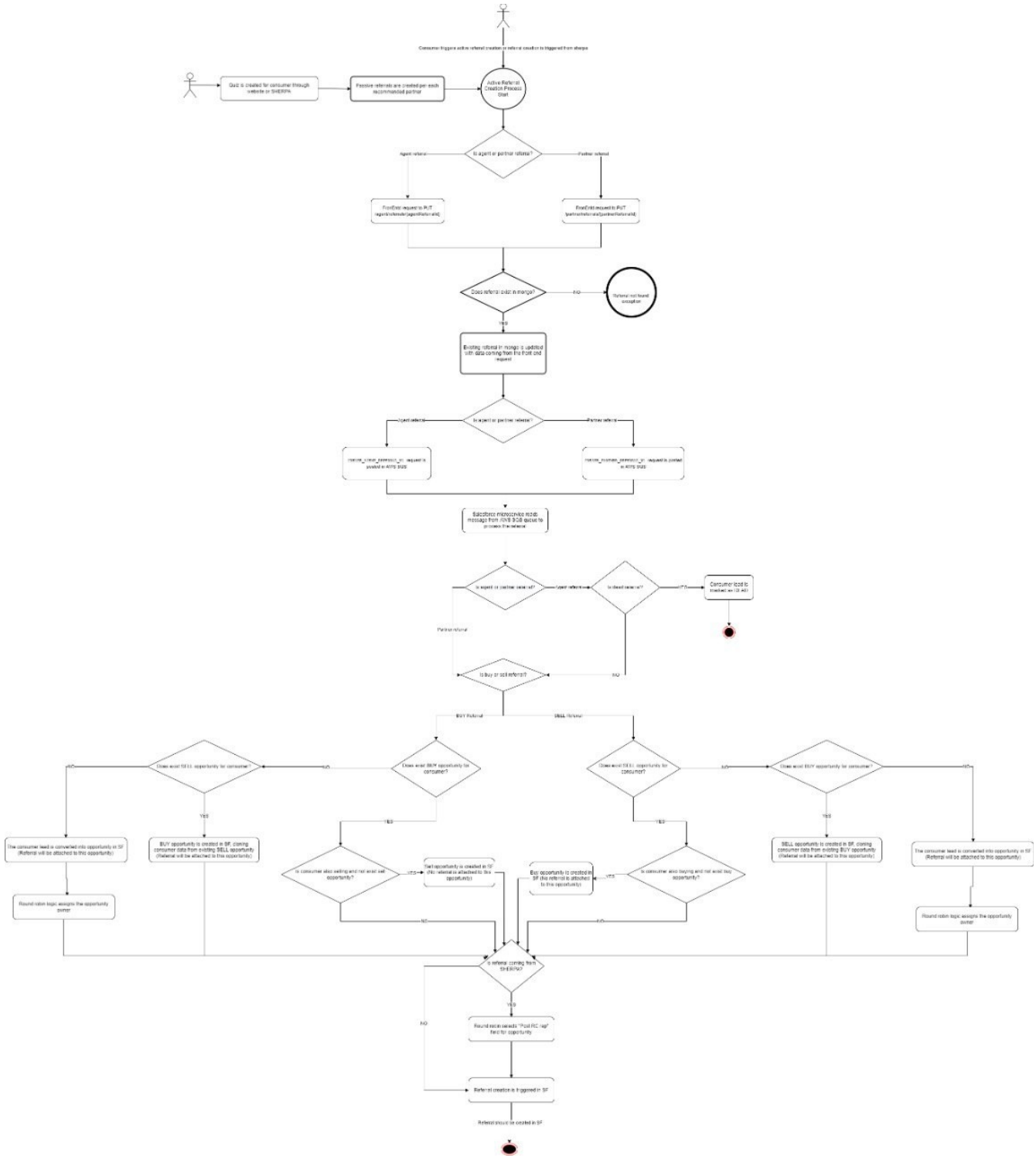


Рисунок А.2 – Діаграма станів

ДОДАТОК Б

ФРАГМЕНТ ЛІСТІНГУ РЕКОМЕНДАЦІЙНОГО АЛГОРИТМУ

```
package com.sold.ms.reco.rest;

import com.sold.ms.commons.domain.ThreadLocalHolder;
import com.sold.ms.commons.rest.APIError;
import com.sold.ms.commons.rest.RestConstants;
import com.sold.ms.commons.rest.dto.AddressInfoPartial;
import com.sold.ms.reco.constant.internal.ServiceSourceConstant;
import com.sold.ms.reco.rest.dto.*;
import com.sold.ms.reco.rest.dto.buyside.BuySidePurchaseFeedback;
import com.sold.ms.reco.rest.dto.buyside.BuySideRecoSet;
import com.sold.ms.reco.rest.dto.buyside.BuySideRecoSummary;
import com.sold.ms.reco.service.*;
import com.sold.ms.referral.rest.dto.AgentReferralPagingResponse;
import com.sold.ms.referral.rest.dto.RecommendationSource;
import com.sold.ms.report.rest.dto.ConsumerIdsBuySide;
import com.sold.ms.report.rest.dto.ConversionRate;
import io.swagger.annotations.*;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.web.bind.annotation.*;

import java.time.Duration;
import java.time.Instant;
import java.util.List;
import java.util.Map;
import java.util.Set;
import static com.sold.ms.reco.constant.internal.ServiceSourceConstant.AGENT_RECO_REFRESH;
import static com.sold.ms.reco.constant.internal.ServiceSourceConstant.AUTO_REFRESH_SYSTEM;
import static com.sold.ms.reco.constant.internal.ServiceSourceConstant.FRESH_AGENT_PARTNER_RECO;
import static com.sold.ms.reco.constant.internal.ServiceSourceConstant.FRESH_AGENT_RECO;
import static com.sold.ms.reco.constant.internal.ServiceSourceConstant.FRESH_PARTNER_RECO;
import static com.sold.ms.reco.constant.internal.ServiceSourceConstant.PARTNER_RECO_REFRESH;
import static com.sold.ms.reco.constant.internal.ServiceSourceConstant.SIMPLE_PARTNER_RECO;
import static com.sold.ms.referral.rest.dto.RecommendationSource.CONSUMER;

@Api(tags = {"Recommendation API"})
@RestController
@RequestMapping(value = RestConstants.VERSION_ONE)
@Slf4j
@RequiredArgsConstructor
public class RecommendationController {

    private final PartnerRecoService partnerRecoService;
    private final BuySidePartnerRecoService buySidePartnerRecoService;
    private final AgentRecoService agentRecoService;
    private final RefreshAgentRecoService refreshAgentRecoService;
    private final PartnerUtilService partnerUtilService;
    private final QuestionSetService questionSetService;
    private final AgentPartnerRecoService agentPartnerRecoService;
    private final EmailIntegrationService emailIntegrationService;
```

```
        @ApiOperation("Request new partner recommendations for consumer")
        @ApiResponses(value = {
          @ApiResponse(code = 404, response = APIError.class, message = "Invalid quiz, question, or option code"),
          @ApiResponse(code = 200, response = PartnerRecoSet.class, message = "Success"),
          @ApiResponse(code = 500, response = APIError.class, message = "Internal server error.")
        })
        @RequestMapping(method = RequestMethod.POST, value = "/partner-reco/new/{consumerId}")
        public PartnerRecoSet generatePartnerRecommendation(
          @ApiParam(name = "consumerId", value = "ID of the consumer", required = true) @PathVariable Long consumerId,
          @RequestParam(name = "sendEmail", defaultValue = "true") Boolean sendEmail
        ) {
          log.debug("Requesting new partner recommendation for consumer {}", consumerId);
          this.setServiceSource(FRESH_PARTNER_RECO);
          return partnerRecoService.generateNewRecommendation(consumerId, sendEmail);
        }

        @RequestMapping(method = RequestMethod.POST, value = "/agent-partner-reco/new/{consumerId}")
        public PartnerRecoSet generateAgentPartnerRecommendation(
          @ApiParam(name = "consumerId", value = "ID of the consumer", required = true) @PathVariable Long consumerId,
          @RequestParam(name = "sendEmail", defaultValue = "true") Boolean sendEmail,
          @RequestParam(name = "buySide", defaultValue = "true") Boolean buySide
        ) {
          log.debug("Requesting new agent - partner recommendation for consumer {}", consumerId);
          this.setServiceSource(FRESH_AGENT_PARTNER_RECO);
          return agentPartnerRecoService.generateNewAgentPartnerRecommendation(consumerId, sendEmail, buySide, CONSUMER);
        }
```

```
package com.sold.ms.reco.service;

import com.sold.ms.commons.rest.dto.AddressInfo;
import com.sold.ms.commons.service.CounterService;
import com.sold.ms.commons.util.TextUtils;
import com.sold.ms.commons.util.converter.EntityConverter;
import com.sold.ms.consumer.api.client.ConsumerClient;
import com.sold.ms.consumer.constants.ConsumerSource;
import com.sold.ms.consumer.rest.dto.Consumer;
import com.sold.ms.data.api.exceptions.NotFoundException;
import com.sold.ms.data.rest.dto.AvmDataResponse;
import com.sold.ms.data.rest.dto.AvmProviderType;
import com.sold.ms.reco.constant.PartnerType;
import com.sold.ms.reco.constant.PartnersEnum;
import com.sold.ms.reco.domain.ApiRecoInfo;
import com.sold.ms.reco.domain.PartnerConversionRateEntity;
import com.sold.ms.reco.domain.PartnerDefinitionEntity;
import com.sold.ms.reco.domain.PartnerRecoSetEntity;
import com.sold.ms.reco.domain.PartnerRecommendationEntity;
import com.sold.ms.reco.domain.opendoor.GenerateOfferResponse;
import com.sold.ms.reco.mapper.BuySideRecoMapper;
import com.sold.ms.reco.mapper.ConversionRateMapper;
import com.sold.ms.reco.mapper.PartnerRecoSetMapper;
```

```
import com.sold.ms.reco.mapper.PartnerRecommendationMapper;
import com.sold.ms.reco.repository.PartnerConversionRateRepository;
import com.sold.ms.reco.repository.proxy.ProxyPartnerRecoSetRepository;
import com.sold.ms.reco.rest.dto.EligiblePartner;
import com.sold.ms.reco.rest.dto.PartnerRecoReport;
import com.sold.ms.reco.rest.dto.PartnerRecoReport.PartnerMatch;
import com.sold.ms.reco.rest.dto.PartnerRecoSet;
import com.sold.ms.reco.rest.dto.PartnerRecommendation;
import com.sold.ms.reco.rest.dto.PriceRange;
import com.sold.ms.reco.rest.dto.SimplePartnerRecoExchange;
import com.sold.ms.reco.rest.dto.buyside.BuySideRecoSet;
import com.sold.ms.reco.service.holder.PartnersHolder;
import com.sold.ms.reco.service.partners.OpendoorService;
import com.sold.ms.reco.service.utils.ExceptionService;
import com.sold.ms.reco.util.AsyncUtils;
import com.sold.ms.reco.util.CustomTimeUtils;
import com.sold.ms.reco.util.MapEntitiesUtils;
import com.sold.ms.reco.util.QuestionSetUtils;
import com.sold.ms.referral.api.client.PartnerReferralClient;
import com.sold.ms.referral.constants.ReferralType;
import com.sold.ms.referral.rest.dto.AllReferralsByIdsRequest;
import com.sold.ms.referral.rest.dto.PartnerReferral;
import com.sold.ms.referral.rest.dto.PartnerTrackingRequest;
import com.sold.ms.referral.rest.dto.ReferralConsumer;
import com.sold.ms.referral.rest.dto.ReferralParty;
import com.sold.ms.referral.rest.dto.UpdateRecoSetIdRequest;
import com.sold.ms.report.api.client.ReportClient;
import com.sold.ms.report.rest.dto.ConsumerIdsBuySide;
import com.sold.ms.report.rest.dto.ConversionRate;
import com.sold.ms.riddler.rest.dto.QuestionResponse;
import com.sold.ms.riddler.rest.dto.QuestionSet;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.bson.types.ObjectId;
import org.jetbrains.annotations.NotNull;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.dao.DataAccessResourceFailureException;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;
import org.springframework.util.CollectionUtils;

import java.time.Duration;
import java.time.Instant;
import java.util.*;
import java.util.concurrent.CompletableFuture;
import java.util.stream.Collectors;
import java.util.stream.Stream;

import static com.google.api.client.repackaged.com.google.common.base.Strings.isNullOrEmpty;
import static com.sold.ms.reco.constant.ErrorMessages.RECO_SET_NOT_FOUND;
```

```
import static com.sold.ms.reco.constant.RecoConstants.PARTNER_RECO_SET_COLLECTION;
import static com.sold.ms.reco.service.AgentRecoService.ACTIVE_STATUSES;
import static com.sold.ms.reco.service.AgentRecoService.ACTIVE_SUB_STATUSES;
import static com.sold.ms.referral.constants.ReferralStatus.PASSIVE;
import static java.lang.Boolean.TRUE;
import static java.util.concurrent.CompletableFuture.supplyAsync;
import static java.util.stream.Collectors.toList;
import static java.util.stream.Collectors.toSet;

@Slf4j
@Service
@RequiredArgsConstructor
public class PartnerRecoService {

    private final static EntityConverter<PartnerRecoSet, PartnerRecoSetEntity> PARTNER_RECO_SET_CONVERTER = new EntityConverter<>(PartnerRecoSet.class, PartnerRecoSetEntity.class);
    private final static int DEFAULT_PAGE_SIZE = 10;
    private final PartnersHolder partnersHolder;

    private final ApiRecoService apiRecoService;
    private final ProxyPartnerRecoSetRepository partnerRecoSetRepository;
    private final PartnerConversionRateRepository partnerConversionRateRepository;
    private final EmailIntegrationService emailIntegrationService;
    private final SalesforceIntegrationService salesforceIntegrationService;
    private final ConsumerClient consumerClient;
    private final ConsumerClientService consumerClientService;
    private final RiddlerClientService riddlerClientService;
    private final RecommendationAlgorithm recommendationAlgorithm;
    @Qualifier(value = "CounterService")
    private final CounterService counterService;
    private final IPredictionClientService predictionClientService;
    private final ReferralService referralService;
    private final PartnerRecoSetMapper partnerRecoSetMapper;
    private final BuySideRecoMapper buySideRecoMapper;
    private final ConversionRateMapper conversionRateMapper;
    private final ReportClient reportClient;
    private final PartnerReferralClient partnerReferralClient;
    private final PartnerRecommendationMapper partnerRecommendationMapper;
    private final OpendoorService opendoorService;
    private final ExceptionService exceptionService;

    public List<SimplePartnerRecoExchange.SimplePartnerRecoResponse> generateSimplePartnerReco(SimplePartnerRecoExchange.SimplePartnerRecoRequest request) {
        return request.getZipCodes().stream().map(zipCode -> {
            List<String> partners = recommendationAlgorithm.defineEligiblePartnerByZip(zipCode);
            return new SimplePartnerRecoExchange.SimplePartnerRecoResponse(zipCode, partners);
        }).collect(toList());
    }

    @Async
    void generateAsyncPartnerRecoForList(List<ConsumerIdsBuySide> consumersForUpdate) {
        consumersForUpdate.forEach(consumerIdBuySide -> {
            generateNewRecommendation(consumerIdBuySide.getConsumerId(), false);
        });
    }
```

```
                public PartnerRecoSet generateNewRecommendation(Long consumerId, Boolean sendEmail) {
                    return generateNewRecommendation(consumerId, sendEmail, false);
                }

                public PartnerRecoSet generateNewRecommendation(Long consumerId, Boolean sendEmail, Boolean buySide) {
                    try {
                        return generatePartnerRecommendation(consumerId, sendEmail, buySide);
                    } catch (DataAccessResourceFailureException e) {
                        log.error("Error with data retrieval in PartnerRecoService.generateNewRecommendation for consumer {}. Error {}", consumerId, e.getMessage());
                        e.printStackTrace();
                        exceptionService.restartWhenDataAccessResourceFailureException();
                        throw e;
                    }
                }

                @NotNull
                private PartnerRecoSet generatePartnerRecommendation(Long consumerId, Boolean sendEmail, Boolean buySide) {
                    log.debug("PartnerRecoService.generatePartnerRecommendation consumerId: {}, sendEmail: {}", consumerId, sendEmail);
                    CompletableFuture<QuestionSet> futureQuestionSet = supplyAsync(() -> riddlerClientService.getCurrentQuestionForConsumer(consumerId, buySide));
                    Consumer consumer = consumerClient.getConsumerById(consumerId);
                    CompletableFuture<Map<AvmProviderType, AvmDataResponse>> allAvmFuture = supplyAsync(() -> predictionClientService.findAvmForAllProviders(consumer.getAddress(), consumer.getNormalizedAddress()));
                    List<CompletableFuture<String>> futureApiRecommendation = apiRecoService.doFutureApiRecommendation(consumer.getAddress());
                    Map<AvmProviderType, AvmDataResponse> allAvm = AsyncUtils.wait(allAvmFuture).orElse(null);
                    Map<AvmProviderType, AvmDataResponse> avmForUse = filterAvmForInternalUsage(allAvm);
                    Double averageAvm = calculateAverageAvm(avmForUse.values());
                    QuestionSet currentQuestionSet = AsyncUtils.wait(futureQuestionSet).orElse(null);
                    Optional<AvmDataResponse> bestAvm = findBestAvm(allAvm);
                    PartnerRecoSet reco = recommendationAlgorithm.recommendPartners(consumer.getAddress(), currentQuestionSet, averageAvm, bestAvm, futureApiRecommendation);
                    reco.setAvm(avmForUse.values());
                    reco.setAvgAvm(averageAvm);

                    AddressInfo currentAddress = Optional.ofNullable(consumer.getNormalizedAddress()).orElse(consumer.getAddress());

                    PartnerRecoSetEntity partnerRecoSetEntity = PARTNER_RECO_SET_CONVERTER.toT2(reco);
                    Long recoSetId = counterService.getNextSequence(PARTNER_RECO_SET_COLLECTION);
                    partnerRecoSetEntity.setPartnerRecoSetId(recoSetId);
                    partnerRecoSetEntity.setPropertyId(QuestionSetUtils.findAddressQuestionResponse(currentQuestionSet).map(QuestionResponse::getPropertyId).orElse(null));

                    partnerRecoSetEntity.setAddressInfo(currentAddress);
                    partnerRecoSetEntity.setQuestionSetId(currentQuestionSet.getQuestionSetId());

                    if (!ConsumerSource.OPS_TEAM.equals(consumer.getSource())) {

                        QuestionSet finalCurrentQuestionSet = currentQuestionSet;
                        Map<String, PartnerReferral> collect = partnerRecoSetEntity.getPartnerRecommendations().stream()
                                .flatMap(r -> r.getEligiblePartners().stream().map(partner -> {
                                    ReferralParty referralParty = new ReferralParty(Long.parseLong(partner.getPartnerId()), ReferralType.PARTNER);
```

```
                                PartnerReferral
partnerReferral = new PartnerReferral(null,
referralParty,
                                MapEntitiesUtils.mapPartn
erType(r.getPartnerType()), averageAvm, null,
null);
                        partnerReferral.setAddressInf
o(currentAddress);
                        partnerReferral.setReferralSta
tus(PASSIVE);
                        partnerReferral.setType(Refe
rralType.PARTNER);
                        partnerReferral.setBuySideRe
ferral(false);
                        partnerReferral.setRecoSetId(
recoSetId);
                        partnerReferral.setPropertyId
(consumer.getPropertyId());
                        partnerReferral.setReferralCo
nsumer(new ReferralConsumer(consumerId,
                                finalCurrentQuestionSet.ge
tQuestionSetId(), null));
                        partnerReferral.setPropertyId
(partnerRecoSetEntity.getPropertyId());

                                                        if
(PartnersEnum.SWIFT_HOMES.getId().equals(p
artner.getPartnerId())) {
                                partnerReferral.setOfferA
mount(Optional.ofNullable(partner.getPriceRange
()).map(PriceRange::getUpper).orElse(null));
                        }
                        return partnerReferral;
                    })).collect(Collectors.toMap(p -
> String.valueOf(p.getPartner().getPartyId()), r ->
r));
                                Map<String, Long>
referralIdsMap                                  =
referralService.createPassivePartnerReferrals(coll
ect);
                partnerRecoSetEntity.getPartnerR
ecommendations()
                    .forEach(
                                                    r ->
r.getEligiblePartners().forEach(
                                            partner ->
partner.setReferralId(referralIdsMap.get(partner.g
etPartnerId()))
                    )
                );
            }

            partnerRecoSetEntity.getPartnerReco
mmendations()
                .stream()
                .flatMap(partnerRecommendation
->
partnerRecommendation.getEligiblePartners().stre
am())
                .forEach(eligiblePartner -> {
                                                        if
(PartnersEnum.OPENDOOR.getId().equals(eligib
lePartner.getPartnerId())) {
                        processOpendoorOffer(consu
mer, currentAddress, eligiblePartner);
                    }
                });

                PartnerRecoSetEntity saved =
partnerRecoSetRepository.save(partnerRecoSetEn
tity);
                        PartnerRecoSet result =
PARTNER_RECO_SET_CONVERTER.toT1(sa
ved);
                log.debug("PartnerRecoService
consumerId: {} result -> {}", consumerId, result);

                                result.setAvm(new
ArrayList<>(allAvm.values()));
            if (sendEmail) {
                log.info("Sending reco email to
consumer {}", consumerId);
                emailIntegrationService.sendReco
mmendationEmail(consumerId);
            }

            //sync to Salesforce
            salesforceIntegrationService.asyncRe
commendations(result);

            return result;
        }

    package com.sold.ms.reco.service;

    import
com.google.common.collect.ImmutableMap;
    import com.google.common.collect.Lists;
    import
com.sold.ms.commons.rest.dto.AddressInfo;
    import
com.sold.ms.data.rest.dto.AvmDataResponse;
    import
com.sold.ms.data.rest.dto.AvmStatsDTO;
    import
com.sold.ms.data.rest.dto.HouseEligibilityData;
    import
com.sold.ms.reco.constant.PartnerType;
    import
com.sold.ms.reco.constant.PartnersEnum;
```

```
import com.sold.ms.reco.domain.ApiRecoInfo;
import com.sold.ms.reco.domain.PartnerConversionRateEntity;
import com.sold.ms.reco.mapper.AddressMapper;
import com.sold.ms.reco.repository.PartnerConversionRateRepository;
import com.sold.ms.reco.rest.dto.EligiblePartner;
import com.sold.ms.reco.rest.dto.PartnerRecoSet;
import com.sold.ms.reco.rest.dto.PartnerRecommendation;
import com.sold.ms.reco.rest.dto.PriceRange;
import com.sold.ms.reco.rest.dto.buyside.BuySideRecoSet;
import com.sold.ms.reco.rest.dto.buyside.BuySideRecommendation;
import com.sold.ms.reco.rest.dto.buyside.EligibleBuySidePartnerReco;
import com.sold.ms.reco.rest.dto.partners.msho.HomeValueResponse;
import com.sold.ms.reco.rest.dto.partners.swifthomes.EligibleAddressRequest;
import com.sold.ms.reco.rest.dto.partners.swifthomes.EligibleAddressResponse;
import com.sold.ms.reco.service.holder.QuizBuySideHolder;
import com.sold.ms.reco.service.holder.SellingTypesHolder;
import com.sold.ms.reco.service.partners.MshoService;
import com.sold.ms.reco.service.vo.SellingTypeInfo;
import com.sold.ms.reco.util.AsyncUtils;
import com.sold.ms.reco.util.ListUtils;
import com.sold.ms.riddler.rest.dto.QuestionResponse;
import com.sold.ms.riddler.rest.dto.QuestionSet;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang.StringUtils;
import org.apache.commons.lang3.tuple.Pair;
import org.springframework.stereotype.Service;

import java.util.*;
import java.util.Map.Entry;
import java.util.concurrent.CompletableFuture;
import java.util.function.Function;
import java.util.function.Predicate;
import java.util.stream.Collectors;
import java.util.stream.Stream;

import static com.sold.ms.reco.constant.PartnerType.AGENT_PARTNER;
import static com.sold.ms.reco.constant.PartnerType.CASH_OFFER_PARTNER;
import static com.sold.ms.reco.constant.PartnerType.FSB0_PARTNER;
import static com.sold.ms.reco.constant.PartnerType.MORTGAGE;
import static com.sold.ms.reco.constant.PartnerType.NON_TRADITIONAL_AGENT_PARTNER;
import static com.sold.ms.reco.constant.PartnerType.SELL_AND_STAY;
import static com.sold.ms.reco.constant.PartnerType.TRADE_IN;
import static com.sold.ms.reco.constant.PartnersEnum.*;
import static com.sold.ms.reco.service.partners.MshoService.mshoCashOfferPartnerPosition;
import static com.sold.ms.reco.service.partners.MshoService.recoContainsMshoPartner;
import static com.sold.ms.reco.util.BeanUtils.eachNotNull;
import static java.lang.Boolean.TRUE;
import static java.util.stream.Collectors.toList;
import static org.hibernate.validator.internal.util.CollectionHelper.asSet;

@Slf4j
@Service
```

```
@RequiredArgsConstructor
public class RecommendationAlgorithm {

    private final SellingTypesHolder sellingTypesHolder;
    private final PartnerEligibilityService partnerEligibilityService;
    private final IPredictionClientService predictionClientService;
    private final QuizBuySideHolder quizBuySideHolder;
    private final PartnerConversionRateRepository partnerConversionRateRepository;
    private final SwiftHomesService swiftHomesService;
    private final ApiRecoService apiRecoService;
    private final AddressMapper addressMapper;
    private static final List<PartnerType> SWAP_PARTNER_TYPES = Lists.newArrayList(AGENT_PARTNER, CASH_OFFER_PARTNER);
    private final MshoService mshoService;
    private final SalesforceIntegrationService salesforceIntegrationService;


    public List<String> defineEligiblePartnerByZip(String zip) {
        List<String> partners = partnerEligibilityService.defineEligiblePartnersByZip(zip);

        Map<String, Double> partnersConversions = partnerConversionRateRepository.findAll()
                .stream()
                .collect(Collectors.toMap(cr -> String.valueOf(cr.getPartnerId()), PartnerConversionRateEntity::getConversionRate));

        partners.stream().sorted(Comparator.comparingDouble(p -> partnersConversions.getOrDefault(p, Double.MIN_VALUE)).reversed());

        return partners;
    }

    public PartnerRecoSet recommendPartners(AddressInfo addressInfo, QuestionSet questionSet, Double averageAvm, Optional<AvmDataResponse> avmDataResponse, List<CompletableFuture<String>> futureApiRecommendation) {
        log.debug("RecommendationAlgorithm.recommendPartners questionSet: {}", questionSet.getQuestionSetId());
        CompletableFuture<HouseEligibilityData> houseEligibilityDataFuture = predictionClientService.findHouseEligibilityData(addressInfo);
        CompletableFuture<AvmStatsDTO> lastAvmStatsFuture = predictionClientService.findLastAvmStats(addressInfo);

        Map<String, Double> partnersConversions = partnerConversionRateRepository.findAll()
                .stream()
                .collect(Collectors.toMap(cr -> String.valueOf(cr.getPartnerId()), PartnerConversionRateEntity::getConversionRate));
        Double bestAvm = avmDataResponse.map(AvmDataResponse::getValue).orElse(null);
        Optional<HouseEligibilityData> eligibilityOpt = AsyncUtils.wait(houseEligibilityDataFuture);
        eligibilityOpt = Optional.of(eligibilityOpt.map(v -> {
            v.setEstimatedValue(bestAvm);
            return v;
        }).orElseGet(() -> {
            HouseEligibilityData r = new HouseEligibilityData();
            r.setEstimatedValue(bestAvm);
            return r;
        }));
        Optional<AvmStatsDTO> avmStatsOptional = AsyncUtils.wait(lastAvmStatsFuture);
        Map<PartnerType, PartnerRecommendation> recommendations = doPartnerRecommendation(addressInfo, questionSet, avmStatsOptional, eligibilityOpt, averageAvm, futureApiRecommendation);
        eliminatePartnerIfNeeded(recommendations, HOME_BAY);

        PartnerRecommendation nonTraditionalAgentRecommendation = recommendations.getOrDefault(NON_TRADITIONAL_AGENT_PARTNER, null);
```

```
            // If the house avm is > 175% of the median value for zip or value is > 2 mil
                                boolean shouldNonTraditionalNotBeFirst = avmStatsOptional.isPresent()
                                        && eachNotNull(nonTraditionalAgentRecommendation, eligibilityOpt.get().getEstimatedValue(), avmStatsOptional.get().getMedianAvmZipCode())
                    && ((eligibilityOpt.get().getEstimatedValue() > avmStatsOptional.get().getMedianAvmZipCode() * 1.75) || (
                            eligibilityOpt.get().getEstimatedValue() > 2000000));

                            List<PartnerRecommendation> notEligiblePartners = recommendations.values().stream().filter(filterNotEligiblePartner(true))
                            .collect(toList());
                                List<PartnerRecommendation> partnerRecommendations = recommendations.values().stream().filter(filterNotEligiblePartner(false))
                            .sorted((o1, o2) -> o2.getScore() - o1.getScore()).collect(toList());

                    if (shouldNonTraditionalNotBeFirst && NON_TRADITIONAL_AGENT_PARTNER.equals(partnerRecommendations.get(0).getPartnerType())) {
                            ListUtils.swapIfCan(partnerRecommendations, 0, 1);
                    }

                    partnerRecommendations.addAll(notEligiblePartners);

                    partnerRecommendations.forEach(reco -> {

                                        List<EligiblePartner> nullConversionRatePartners = reco.getEligiblePartners().stream()
                                                .filter(v -> !partnersConversions.containsKey(v.getPartnerId()))
                            .collect(toList());

                                        List<EligiblePartner> zeroConversionRatePartners = reco.getEligiblePartners().stream()
                                                .filter(v -> partnersConversions.containsKey(v.getPartnerId()) && partnersConversions.get(v.getPartnerId()).equals(0.))
                            .collect(toList());

                                        List<EligiblePartner> conversionRatePartner = reco.getEligiblePartners().stream()
                                                .filter(v -> partnersConversions.containsKey(v.getPartnerId()) && !partnersConversions.get(v.getPartnerId()).equals(0.))
                            .sorted(Comparator.comparingDouble(p -> partnersConversions.get(((EligiblePartner) p).getPartnerId())).reversed())
                            .collect(toList());

                                        Map<Double, List<EligiblePartner>> partnersByConversionRate = conversionRatePartner.stream()
                                    .collect(Collectors.groupingBy(c -> partnersConversions.get(c.getPartnerId()), LinkedHashMap::new, toList()));

                                partnersByConversionRate.values().forEach(Collections::shuffle);

                                        conversionRatePartner = partnersByConversionRate.values().stream().flatMap(Collection::stream).collect(toList());

                                Collections.shuffle(zeroConversionRatePartners);
                                Collections.shuffle(nullConversionRatePartners);

                                reco.setEligiblePartners(Stream.concat(Stream.concat(conversionRatePartner.stream(), nullConversionRatePartners.stream()), zeroConversionRatePartners.stream())
                                    .collect(toList()));

                    });

                                                if (isOpendoorPresent(partnerRecommendations)) {
                                swapPartners(partnerRecommendations);
                    }
```

```
                List<PartnerRecommendation>
sortedPartnerRecommendations                =
partnerRecommendations.stream()
                                .filter(pr  ->  !
pr.getEligiblePartners().isEmpty()                ||
AGENT_PARTNER == pr.getPartnerType())
                .map(partnerRecommendationTyp
e -> {
                    List<EligiblePartner> sorted =
getSortedPartnerRecommendations(partnerRecom
mendationType,
                        partnersConversions);

                    partnerRecommendationType.se
tEligiblePartners(sorted);
                                return
partnerRecommendationType;
                })
                .collect(toList());

            projectRequirementsSetRecommend
ation(sortedPartnerRecommendations);
            moveMortgagePartnersToLastPositio
n(sortedPartnerRecommendations);

                    PartnerRecoSet reco = new
PartnerRecoSet();
             reco.setConsumerId(questionSet.get
ConsumerId());
            reco.setPartnerRecommendations(sor
tedPartnerRecommendations);

            return reco;
        }

                            public void
projectRequirementsSetRecommendation(List<Pa
rtnerRecommendation>
partnerRecommendations) {
            HashSet<String> reasonFSBO = new
HashSet<>();
                    reasonFSBO.add("Project
requirements");

                    PartnerRecommendation
partnerRecommendationFSBO = new
PartnerRecommendation();
             partnerRecommendationFSBO.setPa
rtnerType(FSB0_PARTNER);
            partnerRecommendationFSBO.setRe
asons(reasonFSBO);
              List<EligiblePartner> eligibility =
new ArrayList<>();
            eligibility.add(partnerEligibilityServi
ce.returnEligiblePartner(FIZBER.getStringValue(
)));
            partnerRecommendationFSBO.setEli
giblePartners(eligibility);
             partnerRecommendations.add(partne
rRecommendationFSBO);
        }

                private List<EligiblePartner>
getSortedPartnerRecommendations(PartnerRecom
mendation partnerRecommendationType,

Map<String, Double> partnersConversions) {
             if (partnerRecommendationType ==
null || partnersConversions == null) {
                return Collections.emptyList();
            }

                            List<EligiblePartner>
sortedEligiblePartners                =
partnerRecommendationType.getEligiblePartners(
)
                .stream()
                .sorted((o1, o2) -> {
                    int convRateCompareResult =
partnersConversions.getOrDefault(o2.getPartnerId
(), 0d)
                        .compareTo(partnersConversi
ons.getOrDefault(o1.getPartnerId(), 0d));

                                return
convRateCompareResult != 0
                        ? convRateCompareResult
                                :
o1.getPartnerName().compareTo(o2.getPartnerNa
me());
                })
                .collect(toList());

                                if
(CASH_OFFER_PARTNER.equals(partnerReco
mmendationType.getPartnerType()))
             moveMshoCashOfferToFirstPositi
on(sortedEligiblePartners);

            return sortedEligiblePartners;
        }

                            private boolean
isOpendoorPresent(List<PartnerRecommendation
> partnerRecommendations) {
                                return
partnerRecommendations.stream()
                                .filter(p  ->
CASH_OFFER_PARTNER.equals(p.getPartnerT
ype()))
                                .anyMatch(pr  ->
pr.getEligiblePartners()
```

```
                .stream()
                                .anyMatch(ep ->
OPENDOOR.getId().equals(ep.getPartnerId())));
        }

        /**
         * WARN impure
         */
                                private void
swapPartners(List<PartnerRecommendation>
partnerRecommendations) {

            final boolean allSwapPartnersPresent
= partnerRecommendations.stream()
                .map(PartnerRecommendation::get
PartnerType)
                .collect(toList())
                 .containsAll(SWAP_PARTNER_
TYPES);

            if (allSwapPartnersPresent) {
                final PartnerType firstPartnerType
=
partnerRecommendations.get(0).getPartnerType();

                    if (AGENT_PARTNER ==
firstPartnerType) {
                ListUtils.swapIfCan(partnerRec
ommendations,
findPartnerIndex(CASH_OFFER_PARTNER,
partnerRecommendations), 1);
                                } else if
(CASH_OFFER_PARTNER.equals(firstPartnerT
ype)) {
                ListUtils.swapIfCan(partnerRec
ommendations,
findPartnerIndex(AGENT_PARTNER,
partnerRecommendations), 1);
                }
            }
        }

                                private Integer
findPartnerIndex(PartnerType pt,
List<PartnerRecommendation> recos) {
            for (int i = 0; i < recos.size(); i++) {
                final PartnerRecommendation reco
= recos.get(i);
                if (pt == reco.getPartnerType()) {
                    return i;
                }
            }
            //same index for not swapping
            return 1;
        }
                                private
Predicate<PartnerRecommendation>
filterNotEligiblePartner(Boolean revers) {
            return reco ->
            {
                boolean eligibleAtLeastOne =
AGENT_PARTNER.equals(reco.getPartnerType(
)) ||
                    reco.getEligiblePartners()
                        .stream()
                        .anyMatch(EligiblePartner::is
Eligible);
                                return revers !=
eligibleAtLeastOne;
            };
        }

                        public Map<PartnerType,
PartnerRecommendation>
doPartnerRecommendation(AddressInfo
addressInfo, QuestionSet questionSet,

Optional<AvmStatsDTO> avmStatsOptional,

Optional<HouseEligibilityData> eligibilityOpt,
Double averageAvm,

List<CompletableFuture<String>>
futureApiRecommendation) {
            log.debug("RecommendationAlgorit
hm.doPartnerRecommendation questionSet: {}",
questionSet.getQuestionSetId());
                                Map<PartnerType,
PartnerRecommendation> partnersReco = new
HashMap<>();

                                Map<PartnerType,
List<SellingTypeInfo>> sellingTypesByPartner =
questionSet.getQuestionResponses()
                .stream()
                                .filter(resp ->
StringUtils.isNotEmpty(resp.getResponseOptionI
d()))
                .map((qr ->
                                new
ArrayList<>(sellingTypesHolder.getSellingTypes
InfoByQuestionAndAnswer(qr.getQuestionId(),
qr.getResponseOptionId()).values())))
                .reduce(new ArrayList<>(), (acc,
list) -> {
                    acc.addAll(list);
                    return acc;
                }).stream()
                .collect(Collectors.groupingBy(Sel
lingTypeInfo::getPartnerType));
```

```
            sellingTypesByPartner.remove(FSB0
_PARTNER);
            sellingTypesByPartner.putIfAbsent(
MORTGAGE, Collections.emptyList());
            sellingTypesByPartner.putIfAbsent(
SELL_AND_STAY, Collections.emptyList());
            sellingTypesByPartner.putIfAbsent(
TRADE_IN, Collections.emptyList());

            sellingTypesByPartner.forEach((part
nerType, sellingTypes) -> {
                PartnerRecommendation
partnerReco                                =
partnersReco.getOrDefault(partnerType,      new
PartnerRecommendation());
                partnerReco.setPartnerType(partne
rType);
                sellingTypes.sort((o1, o2) ->
o2.getAnswerWeight() - o1.getAnswerWeight());

                                int     score     =
sellingTypes.stream().map(SellingTypeInfo::getW
eight).filter(Objects::nonNull).mapToInt(Integer::i
ntValue).sum();
                partnerReco.setScore(score);
                    Set<String> reasons =
sellingTypes.stream().map(SellingTypeInfo::getR
eason).filter(Objects::nonNull).collect(Collectors.t
oSet());
                partnerReco.setReasons(reasons);
                    partnersReco.put(partnerType,
partnerReco);
            });

                        Map<PartnerType,
List<EligiblePartner>> eligiblePartnersByType =
partnerEligibilityService
            .defineEligibleSellSidePartners(eli
gibilityOpt,      addressInfo,      questionSet,
averageAvm, futureApiRecommendation)
            .stream()
            .collect(Collectors.groupingBy(Eli
giblePartner::getPartnerType));
                Map<String,  EligiblePartner>
partnersByType                             =
ListUtils.groupByWithOverwrite(
            eligiblePartnersByType.get(NON_
TRADITIONAL_AGENT_PARTNER),
            EligiblePartner::getPartnerId);

            eligiblePartnersByType.put(NON_T
RADITIONAL_AGENT_PARTNER,
eliminatePartners(partnersByType));

        //      preset static partner pros and cons
and eligible partners
            partnersReco.forEach((pt, reco) -> {
                reco.setStaticSellingTypeInfo(selli
ngTypesHolder.getStaticSellingTypeInfo(pt));
                reco.setEligiblePartners(eligiblePa
rtnersByType.getOrDefault(pt,
Collections.emptyList()));
            });

                                                if
(recoContainsSwiftHomesPartner(partnersReco))
{
                    EligibleAddressRequest
eligibleAddressRequest                     =
addressMapper.mapToEligibleAddressRequest(ad
dressInfo);
                EligibleAddressResponse
addressEligibleResponse                    =
swiftHomesService.isAddressEligibleForSwiftHo
mes(eligibleAddressRequest);
                if (addressEligibleResponse ==
null
|| !"Offer".equalsIgnoreCase(addressEligibleResp
onse.getResult()))
                removeSwiftHomesPartner(part
nersReco);
            }

                                                if
(recoContainsMshoPartner(partnersReco)) {
                    HomeValueResponse
homeValueResponse                          =
mshoService.validateHome(addressInfo);
            if (homeValueResponse == null || !
TRUE.equals(homeValueResponse.getIsBuyBoxF
it()))
                removeMshoPartner(partnersRe
co);

            if (homeValueResponse == null) {
                homeValueResponse = new
HomeValueResponse();
                homeValueResponse.setIsBuyB
oxFit(false);
            } else {
                PriceRange priceRange = new
PriceRange(homeValueResponse.getLow(),
homeValueResponse.getHigh());
                partnersReco.values().stream()
                                .flatMap(r ->
r.getEligiblePartners().stream())
                    .filter(eligiblePartner ->
asSet(MSHO_CASH_OFFER.getId(),
MSHO_SELL_AND_STAY.getId()).contains(elig
iblePartner.getPartnerId()))
```

```
                .forEach(mshoPartner -> mshoPartner.setPriceRange(priceRange));
            }

            salesforceIntegrationService.asyncSendMshoHomeData(questionSet.getConsumerId(), homeValueResponse);
        }

        long numberOfPartners = partnersReco.values().stream().map(PartnerRecommendation::getEligiblePartners).mapToLong(Collection::size).sum();

        if (avmStatsOptional.isPresent() && eligibilityOpt.isPresent()) {
            Double medianAvmZipCode = avmStatsOptional.get().getMedianAvmZipCode();
            Double estimatedValue = eligibilityOpt.get().getEstimatedValue();
            if (eachNotNull(medianAvmZipCode, estimatedValue) && numberOfPartners > 1) {
                Set<PartnerType> partnersForRemove = filterPartnerRecosByAvm(medianAvmZipCode, estimatedValue);
                partnersReco.keySet().removeAll(partnersForRemove);
            }
        }

        List<EligiblePartner> cashOfferPartners = Optional.ofNullable(partnersReco.get(CASH_OFFER_PARTNER)).map(PartnerRecommendation::getEligiblePartners).orElse(Collections.emptyList());

//      cashOfferPartners.stream().filter(p -> OPENDOOR.getId().equals(p.getPartnerId())).findFirst().ifPresent(openDoor -> {
//          cashOfferPartners.clear();
//          cashOfferPartners.add(openDoor);
//      });

        cashOfferPartners.stream().filter(p -> SWIFT_HOMES.getId().equals(p.getPartnerId()))
                .findFirst().ifPresent(eligiblePartner -> {
            CompletableFuture<String> swifthomesPossibleReco = apiRecoService.doSwiftHomesRecommendation(addressInfo);
            ApiRecoInfo apiRecoInfo = swiftHomesService.retrieveFutureResponse(swifthomesPossibleReco);
            if (apiRecoInfo.getLowerPrice() != null || apiRecoInfo.getUpperPrice() != null) {
                eligiblePartner.setPriceRange(new PriceRange(apiRecoInfo.getLowerPrice(), apiRecoInfo.getUpperPrice()));
            }
        });

        return partnersReco;
    }

    private void removeSwiftHomesPartner(Map<PartnerType, PartnerRecommendation> partnersReco) {
        partnersReco.values()
                .forEach(eligiblePartnersByType -> {
                    eligiblePartnersByType.getEligiblePartners()
                            .removeIf(eligiblePartner -> SWIFT_HOMES.getId().equals(eligiblePartner.getPartnerId()));
                });
    }

    private boolean recoContainsSwiftHomesPartner(Map<PartnerType, PartnerRecommendation> partnersReco) {
        return partnersReco.values()
                .stream()
                .flatMap(eligiblePartnersByType -> eligiblePartnersByType.getEligiblePartners().stream())
                .anyMatch(eligiblePartner -> SWIFT_HOMES.getId().equals(eligiblePartner.getPartnerId()));
    }

    private static Set<PartnerType> filterPartnerRecosByAvm(Double medianAvmZipCode, Double estimatedValue) {
        return PARTNERS_AVM_CONDITIONS.entrySet()
                .stream()
                .filter((entry) -> entry.getValue().apply(Pair.of(medianAvmZipCode, estimatedValue)))
                .map(Entry::getKey)
                .collect(Collectors.toSet());

    }
```

```
        /**
         * Pair -> key is the median avm value is the estimated value
         */
        private static final Map<PartnerType, Function<Pair<Double, Double>, Boolean>> PARTNERS_AVM_CONDITIONS = ImmutableMap.of(
            FSB0_PARTNER, (avm) -> {
                final Double MIN_AVM_VALUE = 700_000.;
                Double medianAvm = avm.getLeft();
                Double estimatedAvm = avm.getRight();
                return estimatedAvm > medianAvm * 1.25 || estimatedAvm > MIN_AVM_VALUE;
            }
        );

        private void eliminatePartnerIfNeeded(Map<PartnerType, PartnerRecommendation> recommendations, PartnersEnum partnerToEliminate) {
            recommendations.values().forEach(rv -> rv.getEligiblePartners().stream()
                .filter(p -> !partnerToEliminate.getId().equals(p.getPartnerId()))
                .findFirst()
                .ifPresent(fp -> rv.getEligiblePartners().removeIf(p -> partnerToEliminate.getId().equals(p.getPartnerId()))));
        }

        public BuySideRecoSet recommendBuySidePartners(Long consumerId, QuestionSet currentQuestionSet) {
            log.debug("RecommendationAlgorithm.recommendBuySidePartners consumerId: {} currentQuestionSet -> {}", consumerId, currentQuestionSet);
            BuySideRecoSet.BuySideRecoSetBuilder recoSetBuilder = BuySideRecoSet.builder()
                .consumerId(consumerId)
                .questionSetId(currentQuestionSet.getQuestionSetId());

            List<QuestionResponse> questionResponses = currentQuestionSet.getQuestionResponses();
            Map<String, EligibleBuySidePartnerReco> partnersByType = ListUtils.groupByWithOverwrite(
                partnerEligibilityService.defineEligibleBuySidePartner(currentQuestionSet),
                EligibleBuySidePartnerReco::getPartnerId);

            List<EligibleBuySidePartnerReco> eligiblePartners = eliminatePartners(partnersByType);

            Map<PartnerType, Integer> partnersWeights = calculatePartnersWeights(questionResponses);
            partnersWeights.putIfAbsent(MORTGAGE, 0);
            partnersWeights.putIfAbsent(TRADE_IN, 0);

            List<BuySideRecommendation> buySideRecommendations = partnersWeights.entrySet().stream()
                .map(entry -> Pair.of(entry.getKey(), entry.getValue()))
                .sorted((o1, o2) -> o2.getValue() - o1.getValue()).map(e -> {
                    switch (e.getKey()) {
                        case TRADE_IN:
                        case NON_TRADITIONAL_AGENT_PARTNER:
                        case MORTGAGE:
                            return new BuySideRecommendation(
                                e.getKey(), e.getValue(), eligiblePartners.stream().filter(p -> e.getKey().equals(p.getPartnerType())).collect(toList())
                            );
                        case AGENT_PARTNER:
                            return new BuySideRecommendation(e.getKey(), e.getValue(), null);
                        default:
                            log.debug("There is no handler for this type of partners {}, please check the logic. Consumer: {}", e.getKey(), consumerId);
                            return null;
                    }
                }).collect(toList());

            Map<String, Double> partnersConversions = partnerConversionRateRepository.findAll()
                .stream()
```

```
                .collect(Collectors.toMap(cr -> String.valueOf(cr.getPartnerId()), PartnerConversionRateEntity::getConversionRate));


            buySideRecommendations.stream().filter(reco -> Objects.nonNull(reco.getEligiblePartners())).forEach(reco -> {

                List<EligibleBuySidePartnerReco> nullConversionRatePartners = reco.getEligiblePartners().stream()
                        .filter(v -> !partnersConversions.containsKey(v.getPartnerId()))
                        .collect(toList());

                List<EligibleBuySidePartnerReco> zeroConversionRatePartners = reco.getEligiblePartners().stream()
                        .filter(v -> partnersConversions.containsKey(v.getPartnerId()) && partnersConversions.get(v.getPartnerId()).equals(0.))
                        .collect(toList());

                List<EligibleBuySidePartnerReco> conversionRatePartner = reco.getEligiblePartners().stream()
                        .filter(v -> partnersConversions.containsKey(v.getPartnerId()) && !partnersConversions.get(v.getPartnerId()).equals(0.))
                        .sorted(Comparator.comparingDouble(p -> partnersConversions.get(((EligibleBuySidePartnerReco) p).getPartnerId())).reversed())
                        .collect(toList());

                Map<Double, List<EligibleBuySidePartnerReco>> partnersByConversionRate = conversionRatePartner.stream()
                        .collect(Collectors.groupingBy(c -> partnersConversions.get(c.getPartnerId()), LinkedHashMap::new, toList()));

                partnersByConversionRate.values().forEach(Collections::shuffle);

                conversionRatePartner = partnersByConversionRate.values().stream().flatMap(Collection::stream).collect(toList());

                Collections.shuffle(zeroConversionRatePartners);
                Collections.shuffle(nullConversionRatePartners);

                reco.setEligiblePartners(Stream.concat(Stream.concat(conversionRatePartner.stream(), nullConversionRatePartners.stream()), zeroConversionRatePartners.stream())
                        .collect(toList()));
            });

            log.debug("RecommendationAlgorithm consumerId: {} buySideRecommendations -> {}", consumerId, buySideRecommendations);
            return recoSetBuilder
                    .partnerRecommendations(buySideRecommendations)
                    .build();
        }


        // remove partners from reco
        private <T> List<T> eliminatePartners(Map<String, T> partners) {
            log.debug("RecommendationAlgorithm.eliminatePartners partners -> {}", partners);
            Map<String, T> partnersForProcess = new HashMap<>(partners);

            PARTNERS_INTERACTIONS_RULES.forEach((mainPartner, partner) -> Optional.ofNullable(partnersForProcess.get(mainPartner))
                    .ifPresent((v) -> partnersForProcess.remove(partner)));

            return new ArrayList<>(partnersForProcess.values());
        }

        // elimination rules definitions, helps to remove one partner if another presented
        private static final Map<String, String> PARTNERS_INTERACTIONS_RULES = ImmutableMap.of(
                REALI_TRADE.getId(), REALI.getId(),
                FLY_HOMES_TRADE.getId(), FLY_HOMES.getId()
        );
```

```
    private Map<PartnerType, Integer> calculatePartnersWeights(List<QuestionResponse> questionResponses) {
        log.debug("RecommendationAlgorithm.calculatePartnersWeights questionResponses -> {}", questionResponses);
        return ListUtils.reduce(questionResponses, (r, acc) -> {
            Map<PartnerType, SellingTypeInfo> partnerData = Optional.ofNullable(
                quizBuySideHolder.getSellingTypesInfoByQuestionAndAnswer(r.getQuestionId(), r.getResponseOptionId()))
                .orElse(Collections.emptyMap());

            partnerData.forEach((key, value) -> acc.put(key, value.getWeight() + acc.getOrDefault(key, 0)));
            return acc;
        }, new HashMap<>());
    }

    private void moveMortgagePartnersToLastPosition(List<PartnerRecommendation> sortedPartnerRecommendations) {
        int mortgageRecommendationIndex = mortgagePartnerTypeIndex(sortedPartnerRecommendations);
        if (mortgageRecommendationIndex != -1 && mortgageRecommendationIndex != sortedPartnerRecommendations.size() - 1) {
            PartnerRecommendation mortgagePartnerRecommendation = sortedPartnerRecommendations.get(mortgageRecommendationIndex);
            sortedPartnerRecommendations.remove(mortgageRecommendationIndex);
            sortedPartnerRecommendations.add(mortgagePartnerRecommendation);
        }
    }

    private int mortgagePartnerTypeIndex(List<PartnerRecommendation> partnerRecommendations) {
        int mortagePartnerTypeIndex = -1;
        int loopIndex = 0;
        for (PartnerRecommendation partnerRecommendation : partnerRecommendations) {
            if (MORTGAGE.equals(partnerRecommendation.getPartnerType()))
                mortagePartnerTypeIndex = loopIndex;

            loopIndex++;
        }
        return mortagePartnerTypeIndex;
    }

    private void removeMshoPartner(Map<PartnerType, PartnerRecommendation> partnersReco) {
        partnersReco.values()
            .forEach(eligiblePartnersByType -> {
                eligiblePartnersByType.getEligiblePartners()
                    .removeIf(eligiblePartner -> asSet(MSHO_CASH_OFFER.getId(), MSHO_SELL_AND_STAY.getId()).contains(eligiblePartner.getPartnerId()));
            });
    }

    private void moveMshoCashOfferToFirstPosition(List<EligiblePartner> sortedEligiblePartners) {
        int mshoCashOfferPosition = mshoCashOfferPartnerPosition(sortedEligiblePartners);
        if (mshoCashOfferPosition != -1 && mshoCashOfferPosition != 0) {
            EligiblePartner mshoPartner = sortedEligiblePartners.get(mshoCashOfferPosition);
            sortedEligiblePartners.remove(mshoCashOfferPosition);
            sortedEligiblePartners.add(0, mshoPartner);
        }
    }
}

package com.sold.ms.reco.service;

import com.google.common.collect.Lists;
import com.sold.ms.agent.api.client.AgentClient;
import com.sold.ms.agent.rest.dto.Agent;
import com.sold.ms.agent.rest.dto.AgentRecoPoints;
import com.sold.ms.agent.rest.dto.AgentReloDirectorResponse;
```

```
import com.sold.ms.agent.rest.dto.AgentTerritoryRecommendationType;
import com.sold.ms.agent.rest.dto.AgentType;
import com.sold.ms.agent.rest.dto.RankedRecommendationTypeAgent;
import com.sold.ms.commons.rest.dto.AddressInfo;
import com.sold.ms.commons.service.CounterService;
import com.sold.ms.consumer.api.client.ConsumerClient;
import com.sold.ms.consumer.constants.ConsumerSource;
import com.sold.ms.consumer.rest.dto.AllConsumersByIdsRequest;
import com.sold.ms.consumer.rest.dto.Consumer;
import com.sold.ms.data.api.client.DataClient;
import com.sold.ms.data.api.exceptions.NotFoundException;
import com.sold.ms.data.rest.dto.AvmDataResponse;
import com.sold.ms.data.rest.dto.AvmProviderType;
import com.sold.ms.data.rest.dto.AvmRequest;
import com.sold.ms.data.rest.dto.AvmResponse;
import com.sold.ms.email.api.client.EmailServiceClient;
import com.sold.ms.email.rest.dto.reco.EmptyAgentRecoNotificationRequest;
import com.sold.ms.reco.constant.RecoConstants;
import com.sold.ms.reco.domain.AgentRecoSetEntity;
import com.sold.ms.reco.mapper.AddressMapper;
import com.sold.ms.reco.mapper.AgentRecoSetMapper;
import com.sold.ms.reco.repository.CustomAgentRecoSetRepository;
import com.sold.ms.reco.repository.proxy.ProxyAgentRecoSetRepository;
import com.sold.ms.reco.rest.dto.AgentMatch;
import com.sold.ms.reco.rest.dto.AgentRecoSet;
import com.sold.ms.reco.service.utils.ExceptionService;
import com.sold.ms.reco.util.ListUtils;
import com.sold.ms.referral.api.client.AgentReferralClient;
import com.sold.ms.referral.constants.ReferralStatusEnum;
import com.sold.ms.referral.constants.ReferralType;
import com.sold.ms.referral.rest.dto.AgentReferral;
import com.sold.ms.referral.rest.dto.AllReferralsByIdsRequest;
import com.sold.ms.referral.rest.dto.PortalStatusInfo;
import com.sold.ms.referral.rest.dto.RecommendationSource;
import com.sold.ms.referral.rest.dto.ReferralConsumer;
import com.sold.ms.referral.rest.dto.ReferralParty;
import com.sold.ms.referral.rest.dto.UpdateRecoSetIdRequest;
import com.sold.ms.report.api.client.ReportClient;
import com.sold.ms.report.rest.dto.ConsumerIdsBuySide;
import com.sold.ms.riddler.rest.dto.QuestionResponse;
import com.sold.ms.riddler.rest.dto.QuestionSet;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.dao.DataAccessResourceFailureException;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
```

```
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;

import java.time.Duration;
import java.time.Instant;
import java.util.*;
import java.util.concurrent.CompletableFuture;
import java.util.stream.Collectors;

import static com.sold.ms.agent.rest.dto.AgentType.CONCIERGE;
import static com.sold.ms.agent.rest.dto.AgentType.FEATURED_AGENT;
import static com.sold.ms.agent.rest.dto.AgentType.UNKNOWN;
import static com.sold.ms.agent.rest.dto.RecommendAgentExchange.RecommendAgentResponse;
import static com.sold.ms.referral.constants.ReferralStatus.PASSIVE;
import static com.sold.ms.referral.rest.dto.DirectorReferralsStatus.HIDE;
import static com.sold.ms.referral.rest.dto.RecommendationSource.AUTO_REFRESH_SYSTEM;
import static java.lang.Boolean.TRUE;
import static java.util.concurrent.CompletableFuture.supplyAsync;
import static java.util.stream.Collectors.toList;
import static org.springframework.util.CollectionUtils.isEmpty;

@Slf4j
@Service
@RequiredArgsConstructor
public class AgentRecoService {

    private static final RankedRecommendationTypeAgent CONCIERGE_AGENT = new RankedRecommendationTypeAgent(0, Collections.singletonList(new AgentRecoPoints(0L, 0L, false, false, CONCIERGE)));
    private static final Integer MAX_AGENT_MATCH_SIZE = 5;
    private static final Long CONCIERGE_ID = 0L;
    public static final Set<String> ACTIVE_STATUSES = getActiveStatusOrSubStatus(ReferralStatusEnum.getAllStatuses());
    public static final Set<String> ACTIVE_SUB_STATUSES = getActiveStatusOrSubStatus(ReferralStatusEnum.getAllSubStatuses());

    private final QuestionSetService questionSetService;
    @Qualifier(value = "CounterService")
    private final CounterService counterService;
    private final AgentClientService agentClientService;
    private final AgentClient agentClient;
    private final AgentReferralClient agentReferralClient;
    private final ProxyAgentRecoSetRepository agentRecoSetRepository;
    private final AgentRecoSetMapper agentRecoSetMapper;
    private final CustomAgentRecoSetRepository customAgentRecoSetRepository;
    private final AddressMapper addressMapper;
    private final ReportClient reportClient;
    private final ConsumerClient consumerClient;
    private final RiddlerClientService riddlerClientService;
    private final DataClient dataClient;
    private final EmailServiceClient emailServiceClient;
    private final RealogyService realogyService;
    private final ExceptionService exceptionService;

    private final static Set<String> ADDRESS_QUESTION = new HashSet<>(Arrays.asList("SP1_BUYING_LOCATION", "SP1_ADDR"));

    @Async
    void generateAsyncAgentRecoForList(Set<ConsumerIdsBuySide> consumersForUpdate) {
```

```
                consumersForUpdate.forEach(consu
merIdsBuySide -> {
                    generateFreshAgentReco(consume
rIdsBuySide.getConsumerId(),
consumerIdsBuySide.getBuySide(),
AUTO_REFRESH_SYSTEM);
                });
            }

                            public AgentRecoSet
generateFreshAgentReco(AgentRecoSet
lastAgentRecoSet, RecommendationSource
recoSource) {
                log.debug("AgentRecoService.gener
ateFreshAgentReco lastAgentRecoSet: {}",
lastAgentRecoSet.getAgentRecoSetId(),
recoSource);
                            AgentRecoSetEntity
freshRecoSetEntity =
generateAgentRecoSetEntity(lastAgentRecoSet.g
etConsumerId(), lastAgentRecoSet.getBuySide(),
recoSource);

                                        if
(freshRecoSetEntity.getMatchAgents() != null
&& lastAgentRecoSet.getMatchAgents() != null)
{
                    List<Long> freshListOfAgents =
freshRecoSetEntity.getMatchAgents().stream().m
ap(AgentMatch::getAgentId).collect(toList());
                List<Long> previousListOfAgents
=
lastAgentRecoSet.getMatchAgents().stream().map
(AgentMatch::getAgentId).collect(toList());
                                        if
(freshListOfAgents.containsAll(previousListOfAg
ents)                                   &&
previousListOfAgents.containsAll(freshListOfAg
ents)) {
                    log.debug("AgentRecoService.g
eneratingFreshAgentReco lastAgentRecoSet {}.
Reco was not updated.",
lastAgentRecoSet.getAgentRecoSetId());
                    return lastAgentRecoSet;
                }
            }

                                    return
agentRecoSetMapper.mapEntity(agentRecoSetRe
pository.save(freshRecoSetEntity));
            }

                            public AgentRecoSet
generateFreshAgentReco(Long consumerId,
Boolean buySide, RecommendationSource
recoSource) {
                log.debug("AgentRecoService.gener
ateFreshAgentReco consumerId: {} - buySide: {}
- recoSource: {}", consumerId, buySide,
recoSource);
                try {
                                        return
agentRecoSetMapper.mapEntity(generatingFresh
AgentReco(consumerId, buySide, recoSource));
                                    } catch
(DataAccessResourceFailureException e) {
                    log.error("Error with data retrieval
in
AgentRecoService.generateNewRecommendation
for consumer {}. Error {}", consumerId,
e.getMessage());
                    e.printStackTrace();
                    exceptionService.restartWhenData
AccessResourceFailureException();
                    throw e;
                }
            }

                            private AgentRecoSetEntity
generatingFreshAgentReco(Long consumerId,
Boolean buySide, RecommendationSource
recoSource) {
                log.debug("AgentRecoService.gener
atingFreshAgentReco consumerId: {}, buySide:
{}, recoSource: {}", consumerId, buySide,
recoSource);
                AgentRecoSetEntity recoSetEntity =
generateAgentRecoSetEntity(consumerId,
buySide, recoSource);
                AgentRecoSetEntity savedReco =
agentRecoSetRepository.save(recoSetEntity);

                try {
                                        if (!
isEmpty(recoSetEntity.getMatchAgents())   &&
recoSetEntity.getMatchAgents().size() == 1 &&
recoSetEntity.getMatchAgents().get(0).getAgentI
d().equals(0L)) {
                    log.debug("Only concierge is
recommended for consumer {}", consumerId);

                    EmptyAgentRecoNotificationR
equest request = new
EmptyAgentRecoNotificationRequest();
                    request.setConsumerId(consum
erId);
                    request.setRecoAddress(savedR
eco.getAddressInfo());
                    emailServiceClient.sendEmpty
AgentRecommendationNotification(request);
                }
            } catch (Exception e) {
```

```
                log.error("Error sending alert of empty agent recommendation for consumer {}", consumerId);
            }

            return savedReco;
        }

        private AgentRecoSetEntity generateAgentRecoSetEntity(Long consumerId, Boolean buySide, RecommendationSource recoSource) {
            QuestionSet qs = riddlerClientService.getCurrentQuestionForConsumer(consumerId, buySide);
            Consumer consumer = consumerClient.getConsumerById(consumerId);

            Long propertyId = buySide ? consumer.getBuyPropertyId() : consumer.getPropertyId();

            AddressInfo address = buySide
                    ? addressMapper.mapAddress(findAddressQuestionResponse(qs).map(QuestionResponse::getResponseAddress).orElseGet(consumer::getBuySideAddress))
                    : Optional.ofNullable(consumer.getNormalizedAddress()).orElse(consumer.getAddress());

            List<String> zipCodes = buySide
                ? getZipsForBuySide(address, qs)
                    : Lists.newArrayList(address.getZip());

            log.debug("AgentRecoService consumerId: {}, address -> {}, zipCodes: {}", consumerId, address, zipCodes);
            if (zipCodes.size() == 1) {
                address.setZip(zipCodes.get(0));
            }

            if (zipCodes.isEmpty() && !buySide) {
                log.error("There is no zip code for consumer - {}", consumer);
                throw new RuntimeException("There is no zip code.");
            } else if (ListUtils.isListMoreThanOne(zipCodes)) {
        //      recommend concierge
                log.debug("AgentRecoService consumerId: {} isListMoreThanOne", consumerId);
                List<RankedRecommendationTypeAgent> rankedAgents = new ArrayList<>(agentClientService.findAgentTerritoryByZips(zipCodes, buySide)
                        .map(RecommendAgentResponse::getRankedRecommendationTypeAgents)
                        .orElse(Collections.emptyList()));

                rankedAgents.add(CONCIERGE_AGENT);
                Long recoSetId = counterService.getNextSequence(RecoConstants.AGENT_RECO_SET_COLLECTION);
                List<AgentReferral> agentReferrals = getAgentReferrals(consumerId, buySide, consumer, qs, address, null, rankedAgents, recoSetId, recoSource, propertyId);
                List<AgentMatch> agentMatches = getAgentMatches(agentReferrals, rankedAgents);
                AgentRecoSetEntity recoSetEntity = new AgentRecoSetEntity(null,
                        recoSetId,
                        address,
                        propertyId,
                        qs.getQuestionSetId(), consumerId, buySide, agentMatches, true);

                return agentRecoSetRepository.save(recoSetEntity);
            }

            CompletableFuture<List<AvmResponse>> futureAvm = supplyAsync(() -> {
                AvmRequest request = new AvmRequest(Lists.newArrayList(AvmProviderType.QUANTARIUM,
                        AvmProviderType.COLLATERAL),
                        consumer.getAddress(),
                        consumer.getNormalizedAddress());
                return dataClient.retrieveAvmByProvider(request);
            });

            List<RankedRecommendationTypeAgent> rankedAgents = zipCodes.isEmpty()
                    ? new ArrayList<>()
                    : new ArrayList<>(agentClientService.findAgentTerritoryByZip(zipCodes.get(0), buySide, address.getState())
                    .map(AgentTerritoryRecommendationType::getRankedRecommendationTypeAgents)
                    .orElse(Collections.emptyList()));
```

```
                sendRealogyLead(rankedAgents,
consumer);

            //add concierge to agents on BE
              rankedAgents.add(CONCIERGE_A
GENT);

                  log.debug("AgentRecoService
rankedAgents -> {}", rankedAgents);

                        Long recoSetId =
counterService.getNextSequence(RecoConstants.
AGENT_RECO_SET_COLLECTION);

              Double avmAvm = buySide ? null :
Optional.ofNullable(findAvmByProviders(future
Avm))
                              .map(avm ->
avm.values().stream().filter(Objects::nonNull).ma
pToDouble(AvmDataResponse::getValue).filter(
Objects::nonNull).average())
                              .map(avgAvm ->
avgAvm.isPresent() ? avgAvm.getAsDouble() :
null)
                .orElse(null);

              List<AgentReferral> agentsReferral
= getAgentReferrals(consumerId, buySide,
consumer, qs, address, avmAvm, rankedAgents,
recoSetId, recoSource, propertyId);

              List<AgentMatch> agentsMatch =
getAgentMatches(agentsReferral, rankedAgents);

              AgentRecoSetEntity recoSetEntity =
new AgentRecoSetEntity(null, recoSetId,
                address, propertyId,
                        qs.getQuestionSetId(),
consumerId, buySide, agentsMatch, true);

              return recoSetEntity;
            }

                        private List<String>
getZipsForBuySide(AddressInfo address,
QuestionSet qs) {
              log.debug("AgentRecoService.getZip
sForBuySide questionSet -> {}", qs);
               return address.getZip() == null &&
qs.getAdditionalZip() != null
                                          ?
Collections.singletonList(qs.getAdditionalZip())
                                          :
ListUtils.getFirstIfOnlyOne(questionSetService.g
etBuySideZipCodes(qs));
            }

                  private List<AgentMatch>
getAgentMatches(List<AgentReferral>
agentsReferral,
List<RankedRecommendationTypeAgent>
agents) {
              log.debug("AgentRecoService.getAg
entMatches agentsReferral -> {}",
agentsReferral);
                  Map<Long, AgentPointsType>
agentPointsByAgentId =
agents.stream().flatMap(r ->
r.getAgentsRecoPoints().stream())
                .collect(Collectors.toMap(AgentR
ecoPoints::getAgentId, a -> new
AgentPointsType(a.getPoints(), a.getAgentType(),
a.getPolePositionByAdmin(), a.getRoiBoosted()),
(o, o2) -> o));

              return agentsReferral.stream().map(r
-> {
                  AgentType agentType =
Optional.ofNullable(agentPointsByAgentId.get(r.
getAgent().getPartyId())).map(AgentPointsType::
getAgentType).orElse(UNKNOWN);
                  Long agentPoints =
Optional.ofNullable(agentPointsByAgentId.get(r.
getAgent().getPartyId())).map(AgentPointsType::
getPoints).orElse(null);
                  Boolean polePositionByAdmin =
Optional.ofNullable(agentPointsByAgentId.get(r.
getAgent().getPartyId())).map(AgentPointsType::
getPolePositionByAdmin).orElse(null);
                  Boolean roiBoosted =
Optional.ofNullable(agentPointsByAgentId.get(r.
getAgent().getPartyId())).map(AgentPointsType::
getRoiBoosted).orElse(null);

                  return new
AgentMatch(r.getAgentReferralId(),
r.getAgent().getPartyId(), agentType, false, null,
agentPoints, polePositionByAdmin, roiBoosted);
              }).collect(toList());
            }

                  private List<AgentReferral>
getAgentReferrals(Long consumerId, Boolean
buySide, Consumer consumer, QuestionSet qs,
AddressInfo address, Double avmAvm,
List<RankedRecommendationTypeAgent>
rankedAgents, Long recoSetId,
RecommendationSource recoSource, Long
propertyId) {
```

```
            log.debug("AgentRecoService.getAgentReferrals consumerId: {} - buySide: {}", consumerId, buySide);
                    List<Long> agentsIds = rankedAgents.stream()
                     .flatMap(rankedAgent -> rankedAgent.getAgentsRecoPoints().stream().map(AgentRecoPoints::getAgentId))
                .collect(toList());

                    Map<Long, List<String>> featuredTerritoriesByAgentId = agentClient.findAgentsByIds(agentsIds).stream()
                .collect(Collectors.toMap(Agent::getAgentId, Agent::getFeaturedTterritory));

            Map<Long, Long> mapResponses = ListUtils.groupByWithOverwrite(agentClient.getAgentsByIds(agentsIds),
                AgentReloDirectorResponse::getAgentId,
                AgentReloDirectorResponse::getReloDirectorId);

            List<AgentReferral> listForUpsert = new ArrayList<>();
             List<AgentReferral> agentReferrals = rankedAgents.stream()
                                .flatMap(ra -> ra.getAgentsRecoPoints().stream().map(arp -> {
                 AgentReferral agentReferral = new AgentReferral();
                  agentReferral.setAgent(new ReferralParty(arp.getAgentId(), ReferralType.AGENT));
                agentReferral.setAddressInfo(address);
                 agentReferral.setAvgAvm(avmAvm);
                agentReferral.setReloDirectorId(mapResponses.get(arp.getAgentId()));
                                                    if (agentReferral.getReloDirectorId() != null) {
                    agentReferral.setReloAssignmentDate(Instant.now());
                }
                 findAddressQuestionResponse(qs).map(QuestionResponse::getPropertyId).ifPresent(agentReferral::setPropertyId);
                 agentReferral.setBuySideReferral(buySide);
                 agentReferral.setReferralConsumer(new ReferralConsumer(consumerId, qs.getQuestionSetId(), recoSetId));
                agentReferral.setReferralStatus(PASSIVE);
                    agentReferral.setType(ReferralType.AGENT);

                     Boolean forFeaturedAgent = featuredTerritoriesByAgentId.getOrDefault(arp.getAgentId(), Collections.emptyList())
                    .contains(address.getZip());

                agentReferral.setForFeaturedAgent(forFeaturedAgent);
                agentReferral.setRecoSetId(recoSetId);
                 agentReferral.setRecommendationSource(recoSource);
                 agentReferral.setPropertyId(propertyId);

                                                    if (consumer.getLeadMatchInfo() != null && TRUE.equals(consumer.getLeadMatchInfo().getNeedToAutoDql())) {
                                    PortalStatusInfo dqlPortalStatusInfo = new PortalStatusInfo();
                                Instant currentDate = Instant.now();
                    dqlPortalStatusInfo.setStatus(agentReferral.getReferralStatus());
                    dqlPortalStatusInfo.setFriendlyStatus("NONE");
                     dqlPortalStatusInfo.setLastModifiedDate(currentDate);
                     dqlPortalStatusInfo.setStatusUpdateDate(currentDate);
                     dqlPortalStatusInfo.setOppStageStatus("DQL");
                    dqlPortalStatusInfo.setOppSubStatus("Bad Contact Info");
                     dqlPortalStatusInfo.setPortalSection(HIDE);
                    agentReferral.addPortalStatusInfo(dqlPortalStatusInfo);
                }

                                                    if (!ConsumerSource.OPS_TEAM.equals(consumer.getSource())) {
                    listForUpsert.add(agentReferral);
                }
                return agentReferral;
            }))
            .collect(toList());

        List<AgentReferral> upsertedList = agentReferralClient.upsertAgentReferrals(listForUpsert);
```

```
            upsertedList.addAll(agentReferrals);
                                                return
upsertedList.stream().filter(agentReferral ->
agentReferral.getAgentReferralId() !=
null).collect(toList());
        }

                public Page<AgentRecoSet>
findAllAgentRecoSetsByConsumerId(Long
consumerId, Boolean buySide, Pageable
pageable) {
            Page<AgentRecoSetEntity> result =
Optional.ofNullable(buySide)
                                    .map(bs ->
agentRecoSetRepository.findAllByConsumerIdA
ndBuySide(consumerId, bs, pageable))
                                .orElseGet(() ->
agentRecoSetRepository.findAllByConsumerId(c
onsumerId, pageable));

            if (result.getTotalElements() == 0) {
                                    throw new
NotFoundException("Agent reco sets not found");
            }

                                                return
result.map(agentRecoSetMapper::mapEntity);
        }

                            public AgentRecoSet
findAgentRecoSet(Long recoSetId) {
                                                return
agentRecoSetRepository.findByRecoSetId(recoSe
tId)
                .map(agentRecoSetMapper::mapE
ntity)
                            .orElseThrow(() -> new
NotFoundException("Cannot find agent reco set
by id " + recoSetId));
        }

                            public AgentRecoSet
getLatestAgentReco(Long consumerId,
RecommendationSource recoSource) {
            AgentRecoSet lastAgentRecoSet =
Optional.ofNullable(customAgentRecoSetReposit
ory.findLatestSellSideRecoSet(consumerId))
                .map(agentRecoSetMapper::mapE
ntity)
                            .orElseThrow(() -> new
NotFoundException("There is no latest agent reco
for consumer - " + consumerId));

                                                return
getMergedAgentRecoSet(lastAgentRecoSet,
recoSource);
    }

                        private AgentRecoSet
getMergedAgentRecoSet(AgentRecoSet
lastAgentRecoSet, RecommendationSource
recoSource) {
            AgentRecoSet freshAgentRecoSet =
generateFreshAgentReco(lastAgentRecoSet,
recoSource);

                                                if
(freshAgentRecoSet.getAgentRecoSetId().equals(l
astAgentRecoSet.getAgentRecoSetId()))
            return lastAgentRecoSet;

        // if recoSet contains only concierge
further actions don't make sense
                                                if
(freshAgentRecoSet.getMatchAgents().size() ==
1)
            return freshAgentRecoSet;

            List<AgentMatch> agentMatches =
mergeMatchAgents(lastAgentRecoSet.getMatchA
gents(), freshAgentRecoSet.getMatchAgents(),
freshAgentRecoSet.getAgentRecoSetId());
            freshAgentRecoSet.setMatchAgents(
agentMatches);
            agentRecoSetRepository.findByReco
SetId(freshAgentRecoSet.getAgentRecoSetId())
            .ifPresent(reco -> {
                reco.setMatchAgents(agentMatc
hes);
                agentRecoSetRepository.save(re
co);
            });
        return freshAgentRecoSet;
    }

                        public List<AgentMatch>
mergeMatchAgents(List<AgentMatch>
lastAgentMatch, List<AgentMatch>
newAgentMatch, Long recoSetId) {

            Map<Long, AgentMatch> result =
new LinkedHashMap<>();

            Optional<AgentMatch> concierge =
newAgentMatch.stream()
            .filter(a -> a.getAgentId() == 0)
            .findFirst();

                        Map<Long, AgentMatch>
agentMatchWithActiveReferrals =
getAgentMatchWithActiveReferrals(lastAgentMa
tch);
```

```
            updateRecoSetIdInActiveReferralsIf
Needed(agentMatchWithActiveReferrals,
newAgentMatch, recoSetId);

            result.putAll(agentMatchWithActive
Referrals);

                    Map<Long, AgentMatch>
featuredNew = new LinkedHashMap<>();
                        if (result.size() <
MAX_AGENT_MATCH_SIZE) {
                            featuredNew =
newAgentMatch.stream()
                .filter(a -> a.getAgentType() ==
FEATURED_AGENT)
                .limit(MAX_AGENT_MATCH
_SIZE - result.size())
                .collect(Collectors.toMap(Agent
Match::getAgentId, a -> a));

                        if (result.size() +
featuredNew.size()                          <
MAX_AGENT_MATCH_SIZE) {
                    List<Long> featuredIds =
featuredNew.values().stream().map(AgentMatch::
getAgentId).collect(toList());
                    Map<Long, AgentMatch>
otherSorted = newAgentMatch.stream()
                            .filter(am -> !
result.containsKey(am.getAgentId())
                                    && !
featuredIds.contains(am.getAgentId())
                                    && !
am.getAgentId().equals(CONCIERGE_ID))
                            .sorted((o1, o2) ->
o2.getPoints().compareTo(o1.getPoints()))
                    .limit(MAX_AGENT_MAT
CH_SIZE - result.size() - featuredNew.size())
                    .collect(Collectors.toMap(Ag
entMatch::getAgentId, a -> a));
                result.putAll(otherSorted);

            }
        }

            List<AgentMatch> resultValues =
new ArrayList<>(result.values());
        resultValues.addAll(featuredNew.val
ues());

                List<AgentMatch> res =
resultValues.stream()
            .sorted((o1, o2) -> {
                AgentType o1AgentType =
o1.getAgentType();
                AgentType o2AgentType =
o2.getAgentType();

                    if (o1AgentType ==
o2AgentType) {
                        return
Long.compare(o2.getPoints(), o1.getPoints());
                } else {
                        return
Integer.compare(o2AgentType.getRank(),
o1AgentType.getRank());
                }
            })
            .collect(toList());

        concierge.ifPresent(res::add);

        return res;
    }

                    private void
updateRecoSetIdInActiveReferralsIfNeeded(Map
<Long,                      AgentMatch>
agentMatchWithActiveReferrals,
List<AgentMatch> newAgentMatch, Long
recoSetId) {
                        List<Long>
newAgentMatchAgentIds                       =
newAgentMatch.stream().map(AgentMatch::getA
gentId).collect(toList());
                List<Long> referralIds =
agentMatchWithActiveReferrals.values().stream()
                        .filter(a -> !
newAgentMatchAgentIds.contains(a.getAgentId()
))
            .map(AgentMatch::getAgentRefer
ralId)
            .collect(toList());
        if (referralIds.size() == 0) {
            agentReferralClient.updateRecoSe
tId(new      UpdateRecoSetIdRequest(referralIds,
recoSetId));
        }
    }

        private Map<Long, AgentMatch>
getAgentMatchWithActiveReferrals(List<Agent
Match> lastAgentMatch) {
                Set<Long> referralIds =
lastAgentMatch.stream()
                        .filter(a -> !
a.getAgentId().equals(CONCIERGE_ID))
            .map(AgentMatch::getAgentRefer
ralId).collect(Collectors.toSet());
```

```
                List<Long> agentIdsWithActiveReferral = agentReferralClient.findAllAgentReferralByIds(new AllReferralsByIdsRequest(referralIds)).stream()
                        .filter(referral -> ACTIVE_STATUSES.contains(referral.getReferralStatus()) && ACTIVE_SUB_STATUSES.contains(referral.getSubStatus()))
                        .map(agentReferral -> agentReferral.getAgent().getPartyId())
                        .collect(toList());

                return lastAgentMatch.stream()
                        .filter(a -> agentIdsWithActiveReferral.contains(a.getAgentId()))
                        .collect(Collectors.toMap(AgentMatch::getAgentId, a -> a));
            }

            private static Set<String> getActiveStatusOrSubStatus(Map<String, Set<String>> allStatuses) {
                return allStatuses.entrySet().stream()
                        .filter(es -> es.getKey().equals(ReferralStatusEnum.CurrentStatus.ACTIVE.name()))
                        .map(Map.Entry::getValue)
                        .flatMap(Collection::stream)
                        .collect(Collectors.toSet());
            }

            public AgentRecoSet getLatestBuySideAgentReco(Long consumerId, RecommendationSource recoSource) {
                AgentRecoSet lastAgentRecoSet = Optional.ofNullable(customAgentRecoSetRepository.findLatestBuySideRecoSet(consumerId))
                        .map(agentRecoSetMapper::mapEntity)
                        .orElseThrow(() -> new NotFoundException("There is no latest buy-side agent reco for consumer - " + consumerId));

                return getMergedAgentRecoSet(lastAgentRecoSet, recoSource);
            }

        //    private static <T> Predicate<T> distinctByKey(Function<? super T, Object> keyExtractor) {
        //        Map<Object, Boolean> map = new ConcurrentHashMap<>();
        //        return t -> map.putIfAbsent(keyExtractor.apply(t), Boolean.TRUE) == null;
        //    }

            private Optional<QuestionResponse> findAddressQuestionResponse(QuestionSet questionSet) {
                log.debug("AgentRecoService.findAddressQuestionResponse consumerId: {}, questionSet: {}", questionSet.getConsumerId(), questionSet.getQuestionSetId());
                return questionSet.getQuestionResponses()
                        .stream()
                        .filter(r -> ADDRESS_QUESTION.contains(r.getQuestionId()))
                        .findFirst();
            }

            public Set<ConsumerIdsBuySide> findConsumerAndBuySideForRefreshing(Double frequencyRate, Instant startFrom) {

                Date dateTo = Date.from(Instant.now().minus(Duration.ofDays(7)));

                Set<ConsumerIdsBuySide> sortedConsumersForUpdate = customAgentRecoSetRepository.findConsumerAndBuySideByCriteria(Date.from(startFrom), dateTo);

                if ((sortedConsumersForUpdate.size() * frequencyRate) == 0) {
                    return Collections.emptySet();
                }

                List<ConsumerIdsBuySide> result = getCheckedConsumerIdsBuySides(new ArrayList<>(sortedConsumersForUpdate), frequencyRate);

                return new HashSet<>(result);
            }

            private List<ConsumerIdsBuySide> getCheckedConsumerIdsBuySides(List<ConsumerIdsBuySide> consumerIdsBuySides, Double frequencyRate) {
                List<Long> consumerIdsWithoutProperty = consumerIdsBuySides.stream()
```

```
                .filter(a -> !a.getBuySide() &&
a.getPropertyId() == null)
                .map(ConsumerIdsBuySide::getCo
nsumerId)
                .collect(toList());

                        Map<Long, Long>
consumerIdPropertyId =
consumerClient.findConsumersBySort(new
AllConsumersByIdsRequest(new
HashSet<>(consumerIdsWithoutProperty)), new
PageRequest(0, Integer.MAX_VALUE, new
Sort("none")))
                .stream()
                    .filter(a -> a.getPropertyId() !=
null)
                .collect(Collectors.toMap(Consum
er::getConsumerId, Consumer::getPropertyId));

            consumerIdsBuySides
                .forEach(a -> {
                        if (!a.getBuySide() &&
a.getPropertyId() == null) {
                    a.setPropertyId(consumerIdPr
opertyId.getOrDefault(a.getConsumerId(), null));
                    }
                });

            List<ConsumerIdsBuySide> result =
consumerIdsBuySides.stream()
                .filter(a -> !(!a.getBuySide() &&
a.getPropertyId() == null))
                .collect(toList());

                    if (result.isEmpty()) return
Collections.emptyList();

                    List<ConsumerIdsBuySide>
consumerIdsBuySidesFromReport =
reportClient.getConsumersForGenerateFeaturedR
ecos(new ArrayList<>(result));

                                        return
consumerIdsBuySidesFromReport.isEmpty()
                ? Collections.emptyList()
                                        :
consumerIdsBuySidesFromReport.subList(0, (int)
(consumerIdsBuySides.size() * frequencyRate));
            }

                public Map<Long, Long>
getCountOfAgentInReco(List<Long> referralIds)
{
            List<AgentRecoSetEntity> recos =
agentRecoSetRepository.findFirstByConsumerId
AndBuySide(referralIds);

                                Map<Long, Long>
countOfAgentByReferralId = new HashMap<>();

                for (AgentRecoSetEntity reco :
recos) {
                        List<Long> refIds =
reco.getMatchAgents().stream()
                .map(AgentMatch::getAgentRef
erralId)
                .collect(toList());

                        List<Long> common =
referralIds.stream()
                .distinct()
                .filter(refIds::contains)
                .collect(toList());

                common.forEach(com -> {
                    countOfAgentByReferralId.put(
com,
                                        (long)
Optional.ofNullable(reco.getMatchAgents()).map(
List::size)
                        .map(a -> a.equals(0) ? 0 :
a - 1)
                        .orElse(0));
                });
            }

            return countOfAgentByReferralId;
        }

                        private void
sendRealogyLead(List<RankedRecommendation
TypeAgent> rankedAgents, Consumer consumer)
{
            log.debug("AgentRecoService.sendR
ealogyLead consumer: {}",
consumer.getConsumerId());
            try {
                        final Long
NATHAN_GARZA_RELO_DIR_ID = 388L;
                        boolean
oneOfNotSgdAgentsBelongToReloAndOffice =
rankedAgents
                .stream()
                    .flatMap(rankedAgent ->
rankedAgent.getAgentsRecoPoints().stream())
                    .filter(rankedAgent -> !
FEATURED_AGENT.equals(rankedAgent.getAg
entType()))
                .anyMatch(agentRecoPoints ->
NATHAN_GARZA_RELO_DIR_ID.equals(agen
tRecoPoints.getReloDirectorId()) &&
agentBelongsTo_iLeads(agentRecoPoints));
```

```
                long featuredAgentsCount = rankedAgents
                        .stream()
                        .flatMap(rankedAgent -> rankedAgent.getAgentsRecoPoints().stream())
                        .filter(rankedAgent -> FEATURED_AGENT.equals(rankedAgent.getAgentType()))
                        .count();

                if (oneOfNotSgdAgentsBelongToReloAndOffice && featuredAgentsCount < 2)
                    realogyService.sendAsyncRealogyLeadRequest(consumer, false);
                } catch (Exception e) {
                    log.error("Error sending lead request to realogy for consumer {}. Error {}", consumer.getConsumerId(), e.getMessage());
                    e.printStackTrace();
                }
            }

            private boolean agentBelongsTo_iLeads(AgentRecoPoints agentRecoPoints) {
                final String ILEADS = "ileads";
                if (
                    agentRecoPoints.getHsfIdentifiers() != null
                        && agentRecoPoints.getHsfIdentifiers().getAffiliateId() != null
                        && agentRecoPoints.getHsfIdentifiers().getAffiliateId().toLowerCase().contains(ILEADS)
                    )
                    return true;

                if (
                    agentRecoPoints.getBrokerageInfo() != null
                        && agentRecoPoints.getBrokerageInfo().getBrokerageName() != null
                        && agentRecoPoints.getBrokerageInfo().getBrokerageName().toLowerCase().contains(ILEADS)
                    )
                    return true;

                return false;
            }

            private Map<AvmProviderType, AvmDataResponse> findAvmByProviders(CompletableFuture<List<AvmResponse>> futureAvm) {
                if (futureAvm == null)
                    return null;

                try {
                    List<AvmResponse> result = futureAvm.get();
                    return ListUtils.groupByWithOverwrite(result.stream().map(AvmResponse::getResponse)
                        .filter(Objects::nonNull)
                        .filter(v -> v.getValue() != null)
                        .collect(toList()), AvmDataResponse::getAvmProvider);
                } catch (Exception ignore) {
                    return Collections.emptyMap();
                }
            }

            @Data
            @AllArgsConstructor
            private static class AgentPointsType {
                private Long points;
                private AgentType agentType;
                private Boolean polePositionByAdmin;
                private Boolean roiBoosted;
            }
        }

        package com.sold.ms.reco.service;

        import com.sold.ms.agent.api.client.AgentClient;
        import com.sold.ms.agent.api.client.AgentTerritoryClient;
        import com.sold.ms.agent.rest.dto.Agent;
        import com.sold.ms.agent.rest.dto.AgentTerritoryRecommendationType;
        import com.sold.ms.agent.rest.dto.AgentTerritoryRequest;
        import com.sold.ms.agent.rest.dto.portal.AgentInfo;
        import lombok.RequiredArgsConstructor;
        import lombok.extern.slf4j.Slf4j;
        import org.springframework.beans.factory.annotation.Qualifier;
        import org.springframework.stereotype.Service;
```

```
import java.util.List;
import java.util.Map;
import java.util.Optional;

import static com.sold.ms.agent.rest.dto.RecommendAgentExchange.RecommendAgentRequest;
import static com.sold.ms.agent.rest.dto.RecommendAgentExchange.RecommendAgentResponse;

@Slf4j
@Service
@RequiredArgsConstructor
public class AgentClientService {

    @Qualifier("basicAgentTerritoryClientV1")
    private final AgentTerritoryClient agentTerritoryClient;
    private final AgentClient agentClient;

    public Optional<AgentTerritoryRecommendationType> findAgentTerritoryByZip(String zipCode, Boolean buySide, String filterState) {
        log.debug("AgentTerritoryClient.getAgentTerritory - {}", zipCode);

        try {
            return Optional.ofNullable(agentTerritoryClient.getAgentTerritory(new AgentTerritoryRequest(zipCode, buySide, filterState)));
        } catch (Exception e) {
            log.debug("Can not find agent territory by zip code - {}", zipCode, e);
        }
        return Optional.empty();
    }

    public Optional<RecommendAgentResponse> findAgentTerritoryByZips(List<String> zipCodes, Boolean buySide) {
        log.debug("AgentTerritoryClient.getAgentTerritories - {}", zipCodes);

        try {
            return Optional.ofNullable(agentTerritoryClient.getAgentTerritoriesZipCodes(new RecommendAgentRequest(zipCodes, buySide)));
        } catch (Exception e) {
            log.debug("Can not find agent territories by zip codes - {}", zipCodes, e);
        }
        return Optional.empty();
    }

    public Optional<Map<Long, AgentInfo>> getAllAgentsInfoByIds(List<Long> ids) {
        try {
            return Optional.ofNullable(agentClient.findAllAgentsInfoByIds(ids));
        } catch (Exception e) {
            log.debug("Can not find agents info by ids {}", ids, e.getMessage());
        }
        return Optional.empty();
    }

    public Optional<Agent> getAgentById(Long id) {
        try {
            return Optional.ofNullable(agentClient.getAgentById(id));
        } catch (Exception e) {
            log.debug("Can not find agent info by id {}", id, e.getMessage());
        }
        return Optional.empty();
    }

}

package com.sold.ms.reco.domain;

import com.sold.ms.data.rest.dto.AvmDataResponse;
import com.sold.ms.reco.constant.RecoConstants;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.bson.types.ObjectId;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;
import org.springframework.data.mongodb.core.mapping.Field;
```

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

@Document(collection = RecoConstants.PARTNER_RECO_SET_COLLECTION)
@Data
@AllArgsConstructor
@NoArgsConstructor
public class PartnerRecoSetEntity extends AbstractRecoSet {

    private static final long serialVersionUID = 1L;

    @Id
    private ObjectId id;

    @Indexed
    @Field("partner_reco_set_id")
    private Long partnerRecoSetId;

    @Field("avm")
    private Collection<AvmDataResponse> avm;

    @Field("avg_avm")
    private Double avgAvm;

    @Field("partner_recommendations")
    private List<PartnerRecommendationEntity> partnerRecommendations = new ArrayList<>();


}

package com.sold.ms.reco.domain;

import com.sold.ms.reco.constant.PartnerType;
import com.sold.ms.reco.constant.RecoConstants;
import com.sold.ms.reco.domain.eligibility.AbstractDoubleEligibilityRule;
import com.sold.ms.reco.domain.eligibility.IEligibilityRule;
import com.sold.ms.reco.domain.eligibility.IQuizEligibilityRule;
import com.sold.ms.reco.domain.vo.PartnerFee;
import com.sold.ms.reco.domain.vo.fee.calculation.AbstractBuySidePartnerFee;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.bson.types.ObjectId;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import java.util.List;
import java.util.Set;

@Document(collection = RecoConstants.PARTNER_DEFINITION_COLLECTION)
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class PartnerDefinitionEntity {

    private static final long serialVersionUID = 1L;

    @Id
    private ObjectId id;
    private PartnerType partnerType;
    private String partnerName;
    private String partnerId;
    private PartnerFee partnerFee;
    private AbstractBuySidePartnerFee buySidePartnerFee;
    private List<AbstractDoubleEligibilityRule<Double, String>> homeValueRules;
    private List<AbstractDoubleEligibilityRule<Double, String>> buyBudgetValueRules;
    private List<AbstractDoubleEligibilityRule<String, String>> propertyTypeAndPlaceRules;
    private List<AbstractDoubleEligibilityRule<String, Double>> lotSizeRules;
    private List<AbstractDoubleEligibilityRule<Number, String>> yearBuiltRule;
    private IQuizEligibilityRule quizEligibilityRule;

    private Set<String> buySideStates;
```

```
        private Set<String> sellSideStates;

        private Boolean buySidePartner;
        private Boolean publicPartner;
        private Boolean activePartner;

        @Deprecated
            private IEligibilityRule<Double>
bedroomsRule;

                                        private
List<AbstractDoubleEligibilityRule<Double,
String>> bedroomsRules;

        @Deprecated
            private IEligibilityRule<Double>
bathroomsRule;

                                        private
List<AbstractDoubleEligibilityRule<Double,
String>> bathroomsRules;

    }

    package com.sold.ms.reco.domain;

    import
com.sold.ms.commons.rest.dto.AddressInfo;
    import
com.sold.ms.reco.constant.RecoConstants;
    import
com.sold.ms.reco.rest.dto.AgentMatch;
    import lombok.AllArgsConstructor;
    import lombok.Builder;
    import lombok.Data;
    import lombok.EqualsAndHashCode;
    import lombok.NoArgsConstructor;
    import org.bson.types.ObjectId;
    import
org.springframework.data.annotation.Id;
    import
org.springframework.data.mongodb.core.index.Co
mpoundIndex;
    import
org.springframework.data.mongodb.core.index.Co
mpoundIndexes;
    import
org.springframework.data.mongodb.core.index.In
dexed;
    import
org.springframework.data.mongodb.core.mapping
.Document;
    import
org.springframework.data.mongodb.core.mapping
.Field;
    import java.util.ArrayList;
    import java.util.List;

    @Document(collection =
RecoConstants.AGENT_RECO_SET_COLLECT
ION)
    @Data
    @Builder
    @AllArgsConstructor
    @NoArgsConstructor
    @EqualsAndHashCode(callSuper = true)
    @CompoundIndexes({
            @CompoundIndex(name =
"consumer_buy_date_idx", def = "{'consumer_id':
1, 'buySide': 1, 'createdDate': -1}")
    })
    public class AgentRecoSetEntity extends
AbstractRecoSet {

            private static final long
serialVersionUID = 1L;

        @Id
        private ObjectId id;

        @Field("agent_reco_set_id")
        @Indexed
        private Long agentRecoSetId;

        private Boolean buySide;

        private String additionalZip;

        @Field("agent_match_ids")
            private List<AgentMatch>
matchAgents = new ArrayList<>();

            public AgentRecoSetEntity(ObjectId
id, Long agentRecoSetId, AddressInfo
addressInfo,
                                Long propertyId, Long
questionSetId, Long consumerId,
                                    Boolean buySide,
List<AgentMatch> matchAgents, Boolean
isActiveOpportunity) {
                super(questionSetId, consumerId,
addressInfo, propertyId, isActiveOpportunity);
                this.id = id;
                                this.agentRecoSetId =
agentRecoSetId;
                this.buySide = buySide;
                this.matchAgents = matchAgents;
            }
    }

    package com.sold.ms.reco.domain;
```

```
import com.sold.ms.commons.auditing.AbstractAuditEntity;
import com.sold.ms.commons.rest.dto.AddressInfo;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Field;

import java.util.Optional;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class AbstractRecoSet extends AbstractAuditEntity {

    @Field("question_set_id")
    private Long questionSetId;

    @Indexed
    @Field("consumer_id")
    private Long consumerId;

    @Field("address_info")
    private AddressInfo addressInfo;

    @Field("property_id")
    private Long propertyId;

    @Field("active_opportunity")
    private Boolean isActiveOpportunity;

    public Boolean getIsActiveOpportunity() {
        return Optional.ofNullable(isActiveOpportunity).orElse(true);
    }
}
```

```
package com.sold.ms.reco.domain.eligibility;


import com.fasterxml.jackson.annotation.JsonSubTypes;
import com.fasterxml.jackson.annotation.JsonSubTypes.Type;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import com.fasterxml.jackson.annotation.JsonTypeInfo.As;
import com.fasterxml.jackson.annotation.JsonTypeInfo.Id;

@JsonTypeInfo(
  use = Id.CLASS,
  include = As.PROPERTY,
  property = "_class"
)
@JsonSubTypes({
        @Type(value = BetweenEligibilityRule.class),
        @Type(value = ContainsEligibilityRule.class),
        @Type(value = DoesNotContainsEligibilityRule.class),
        @Type(value = DoubleAndEligibilityRule.class),
        @Type(value = EqualsEligibilityRule.class),
        @Type(value = GreaterEligibilityRule.class),
        @Type(value = GreaterOrZeroEligibilityRule.class),
        @Type(value = GreaterNotNullEligibilityRule.class),
        @Type(value = LessEligibilityRule.class),
        @Type(value = AlwaysTrueEligibilityRule.class),
        @Type(value = AlwaysFalseEligibilityRule.class),
        @Type(value = AlwaysTrueDoubleEligibilityRule.class),
        @Type(value = AlwaysFalseDoubleEligibilityRule.class),
        @Type(value = AlwaysTrueIfNotNullEligibilityRule.class),
        @Type(value = MockQuizEligibilityRule.class),
        @Type(value = QuizResponseRule.class),
        @Type(value = MinNormalizationSavingsRule.class),
        @Type(value = DefaultNormalizationSavingsRule.class),

})
```

```
public interface IEligibilityRule<T> {
   Boolean isEligible(T value);

   String getRuleDescription();
}

package com.sold.ms.reco.graphql;

import
com.sold.ms.commons.domain.ThreadLocalHolde
r;
import
com.sold.ms.commons.graphql.ValidationService;
import
com.sold.ms.commons.rest.dto.AddressInfo;
import
com.sold.ms.commons.rest.dto.Paginated;
import
com.sold.ms.reco.constant.internal.ServiceSource
Constant;
import
com.sold.ms.reco.domain.ApiRecoInfo;
import
com.sold.ms.reco.rest.dto.AgentRecoSet;
import
com.sold.ms.reco.rest.dto.PartnerRecoSet;
import
com.sold.ms.reco.service.AgentPartnerRecoServi
ce;
import
com.sold.ms.reco.service.AgentRecoService;
import
com.sold.ms.reco.service.partners.OpendoorServi
ce;
import
com.sold.ms.reco.service.RefreshAgentRecoServi
ce;
import
io.leangen.graphql.annotations.GraphQLArgumen
t;
import
io.leangen.graphql.annotations.GraphQLMutation
;
import
io.leangen.graphql.annotations.GraphQLNonNull;
import
io.leangen.graphql.annotations.GraphQLQuery;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import
org.springframework.data.domain.PageRequest;
import
org.springframework.stereotype.Service;

import java.time.Instant;
import java.util.List;
import java.util.Optional;

import
static
com.sold.ms.reco.constant.internal.ServiceSource
Constant.*;
import
static
com.sold.ms.referral.rest.dto.RecommendationSo
urce.CONSUMER;

@Slf4j
@Service
@RequiredArgsConstructor
public class GraphQLAgentRecoService {

    private final ValidationService
validationService;
    private final AgentRecoService
agentRecoService;
    private final RefreshAgentRecoService
refreshAgentRecoService;
    private final AgentPartnerRecoService
agentPartnerRecoService;
    private final OpendoorService
opendoorService;

    @GraphQLMutation(name =
"generateSellSideAgentMatches")
    public AgentRecoSet
generateAgentMatches(@GraphQLNonNull
@GraphQLArgument(name = "consumerId")
Long consumerId) {
        log.debug("Requesting new agent
match set for consumer {}", consumerId);
        this.setServiceSource(FRESH_AGE
NT_RECO);
        return
agentRecoService.generateFreshAgentReco(consu
merId, false, CONSUMER);
    }

    @GraphQLMutation(name =
"generateBuySideAgentMatches")
    public AgentRecoSet
generateBuySideAgentMatches(@GraphQLNonN
ull @GraphQLArgument(name = "consumerId")
Long consumerId) {
        log.debug("Requesting new buy-side
agent recommendations for consumer {}",
consumerId);
        this.setServiceSource(FRESH_AGE
NT_RECO);
        return
agentRecoService.generateFreshAgentReco(consu
merId, true, CONSUMER);
    }
```

```
                @GraphQLQuery(name = "getLatestBuySideAgentReco")
                public AgentRecoSet retrieveLatestBuysideAgentReco(@GraphQLNonNull @GraphQLArgument(name = "consumerId") Long consumerId) {
                    log.debug("Requesting latest Buyside agent reco for consumer {}", consumerId);
                    this.setServiceSource(AGENT_RECO_REFRESH);
                    return agentRecoService.getLatestBuySideAgentReco(consumerId, CONSUMER);
                }

                @GraphQLQuery(name = "getLatestSellSideAgentReco")
                public AgentRecoSet generateLatestAgentReco(@GraphQLNonNull @GraphQLArgument(name = "consumerId") Long consumerId) {
                    log.debug("Requesting latest sell side agent reco set for consumer {}", consumerId);
                    this.setServiceSource(AGENT_RECO_REFRESH);
                    return agentRecoService.getLatestAgentReco(consumerId, CONSUMER);
                }

                @GraphQLQuery(name = "findAllAgentRecoSetsByConsumerId")
                public Paginated<AgentRecoSet> findAllAgentRecoSetsByConsumerId(@GraphQLNonNull @GraphQLArgument(name = "consumerId") Long consumerId,

@GraphQLNonNull @GraphQLArgument(name = "buySide")Boolean buySide,

@GraphQLArgument(name = "pageNumber") Integer pageNumber,

@GraphQLArgument(name = "pageSize") Integer pageSize) {
                    PageRequest pageRequest = new PageRequest(Optional.ofNullable(pageNumber).orElse(0),
Optional.ofNullable(pageSize).orElse(20));
                    log.debug("Searching all agentRecoSets by consumerId {}", consumerId);
                    return Paginated.wrap(agentRecoService.findAllAgentRecoSetsByConsumerId(consumerId, buySide, pageRequest));
                }

                @GraphQLQuery(name = "findAgentRecoSetById")
                public AgentRecoSet findAgentRecoSetById(@GraphQLNonNull @GraphQLArgument(name = "recoSetId") Long recoSetId) {
                    log.debug("Finding agent reco set by id {}", recoSetId);
                    return agentRecoService.findAgentRecoSet(recoSetId);
                }

                @GraphQLMutation(name = "refreshAgentRecosBasedOnFrequencyInput")
                public List<Long> refreshAgentRecosBasedOnFrequencyInput(@GraphQLNonNull @GraphQLArgument(name = "frequencyRate") Double frequencyRate,
                                                                         @GraphQLNonNull @GraphQLArgument(name = "startFrom") Instant startFrom) {
                    log.debug("Refreshing AgentRecos based on frequency input {} and date start from {}", frequencyRate, startFrom);
                    this.setServiceSource(AUTO_REFRESH_SYSTEM);
                    return refreshAgentRecoService.refreshAgentRecosBasedOnFrequencyInput(frequencyRate, startFrom);
                }
```