МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Литвиненко О.Є.
«___» _____2021 р.

ДИПЛОМНИЙ ПРОЕКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА
ЗА НАПРЯМОМ ПІДГОТОВКИ 123 «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

Тема:«Програма генерації рекомендацій покупцю під час придбання нерухомості»

Виконавець:_____Степанов А. В.
(студент, група, прізвище, ім'я, по-батькові)

Керівник: _____д.т.н., доц. Вавіленкова А. І.
(науковий ступінь, вчене звання, прізвище, ім'я, по-батькові)

Нормоконтролер: _____ Тупота Є.В.
(підпис)

КИЇВ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *кібербезпеки, комп'ютерної та програмної інженерії*
Кафедра *комп'ютеризованих систем управління*
Напрям (спеціальність) 123 *«Комп'ютера інженерія»*
(шифр, найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.Є. Литвиненко
«_____» _____ 2021 р.

ЗАВДАННЯ
на виконання дипломного проекту
Степанова Артема Валерійовича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломного проекту «Програма генерації рекомендацій покупцю під час придбання нерухомості» затверджена наказом ректора від « 4 » лютого 2021 р. №135/ст

2. Термін виконання проекту: з 17 травня 2021 р. по 20 червня 2021 р.

3. Вихідні дані до проекту: існуючі рекомендаційні сервіси, бібліотека *LightFM, Athena* база даних, веб-фреймворк *FastAPI,* функція *WARP*-втрати

4. Зміст пояснювальної записки:

1) Сервіси надання рекомендацій для покупців

2) Технології та алгоритми формування даних і генерації рекомендацій

3) Програмний модуль генерації рекомендацій покупцю під час придбання нерухомості

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) Схема алгоритму навчання моделі з урахуванням функції *WARP*-втрати

2) *UML*- діаграма прецедентів для інтерпретації принципу роботи додатку

3) Результати роботи додатку генерації рекомендацій

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1.	Аналіз джерел за темою дослідження. Розроблення плану дипломного проекту	17.05.2021-21.05.2021	
2.	Затвердження плану дипломного проекту та проведення консультацій з науковим керівником, щодо наповнення пояснювальної записки	22.05.2021-23.05.2021	
3.	Дослідження сайтів, які використовують рекомендаційні системи	24.05.2021-27.05.2021	
4.	Розробка архітектури додатку та дослідження формул генерації рекомендацій	28.05.2021-31.05.2021	
5.	Створення веб-додатку, який генерує рекомендації для переданого користувача	01.06.2021-09.06.2021	
6.	Написання пояснювальної записки	10.06.2021-14.06.2021	
7.	Підготовка демонстраційного матеріалу	15.06.2021 20.06.2021	

7. Дата видачі завдання: « 17 » травня 2021 р.

Керівник дипломного проекту _____ д.т.н., доц. Вавіленкова А.І.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Степанов А.В
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Програма генерації рекомендацій покупцю під час придбання нерухомості»: 58 сторінок, 34 рисунки, 5 таблиць, 15 літературних джерела.

РЕКОМЕНДАЦІЙНА СИСТЕМА, ВЕБ-ДОДАТОК, *LIGHTFM*, *FASTAPI*

Об’єктом дослідження є процес генерації рекомендацій покупцю під час придбання нерухомості.

Предметом дослідження є технології розробки додатків для генерації рекомендацій для певного користувача, базуючись на його історії перегляду.

Метою даного дипломного проекту є проектування та розробка програмного додатку на базі веб-фреймворку *FastAPI* та бібліотеки *LightFM*, який генерує рекомендації.

Методи дослідження – Оркестратор задач (*Airflow*), програмний інструмент для побудови рекомендацій (*LightFM*), алгоритм навчання моделі з урахуванням функції *WARP*-втрати, фреймворк, мова програмування *Python3*, середовище для програмування *IntelliJ IDEA*.

Здійснено огляд сайтів, які використовують рекомендаційні системи, та сайти нерухомості. Розглянуто типи рекомендаційних систем. Здійснено дослідження та порівняння фреймворків та бібліотек, які допомогли б у розробці. Створено та реалізовано структуру роботи програми. Реалізовано програмний додаток, який повертає рекомендації для користувача.

Матеріали дипломного проекту рекомендується використовувати при проведенні досліджень рекомендаційних систем, у навчальному процесі фахівців з системного програмування, а також у сферах, пов’язаних з продажою продуктів і нерухомості.

Прогнозні припущення про розвиток об’єкту та предмету дослідження – застосування запропонованого алгоритму в системах генерації рекомендацій, базуючись на історії переглядів користувачів.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

HTML – *Hypertext Markup Language*

S3 – *Simple Storage Servise*

AWS – *Amazon Web Services*

SQL – *Structed Query Language*

UML – *Unified Modeling Language*

БД – База Даних

WARP – *Weighted-Approximate-Rank-Pairwise*

UX – *User Experience*

UI – *User Interface*

ВСТУП

Актуальність теми. Подання рекомендацій користувачу, є дуже актуальною темою сьогодні. Ціль кожної компанії – це продати якнайбільше продуктів користувачу, а рекомендації користувачу можуть як раз підштовхнути користувача зробити ту чи іншу покупу, або хоча б задуматись над нею. Оскільки у сфері продажу нерухомості таких рекомендацій майже не існує, то дана проблема набуває ще більшої актуальності, бо пропозиції можуть дуже сильно перевищувати попит, і користувач може не побачити релевантні оголошення. Крім цього рекомендації підвищують якість *UX* (досвід користувача), тобто користувач проводить більше часу на сайті, що теж підвищує вірогідність покупки того чи іншого товару. Також рекомендації можуть бути корисними у розсилці, тобто користувачу приходить на пошту набір оголошень або продуктів, які могли б його зацікавити. Всі дані, які використовує додаток для генерації оголошень, беруться з самого сайту де проводиться створення додатку, це може бути переходи на сторінки, лайки, зберігання у обране, придбання та використанні фільтри під час пошуку.

Об’єкт дослідження – процес генерації рекомендацій покупцю під час придбання нерухомості.

Предмет дослідження – технології розробки додатків для генерації рекомендацій для певного користувача, базуючись на його історії перегляду.

Мета дипломного проекту – проектування та розробка програмного додатку на базі веб-фреймворку *FastAPI* та бібліотеки *LightFM*, який генерує рекомендації.

Методи дослідження: оркестратор задач (*Airflow*) – призначенний для виконання великих послідовностей задач за розкладом; програмний інструмент для побудови рекомендацій (*LightFM*) – призначений для обробки вхідних даних та генерації рекомендацій, використовуючи вхідні дані; алгоритм навчання моделі з урахуванням функції *WARP*-втрати – призначений для навчання моделі і оптимізації її результатів; фреймворк *FastAPI* – призначений для зручного створення веб-додатку; мова програмування *Python3* – призначена для реалізації

дипломного проекту; середовище для програмування *IntelliJ IDEA* – призначене для зручного проведення розробки додатків.

Практична цінність. У даному дипломному проекті розроблено програму генерації рекомендацій користувачу, його практична цінність полягає в тому, що, використовуючи цю програму можна згенерувати рекомендації для будь-якого сайту нерухомості.

Новизна дипломного проекту полягає у використанні функції *WARP*-втрати для отпимізації рекомендацій щодо оголошень нерухомості.

Особистий внесок випускника. Всі результати, представлені у дипломному проекті, отримані випускником особисто.

Практичні значення отриманих результатів у дипломному проекті дають змогу використовувати отримані дані при проведенні досліджень та проектувань рекомендаційних систем, навчальному процесі фахівців з системного програмування та у сферах пов’язаних з продажом товарів, продуктів, нерухомості і тд.

Апробація отриманих результатів. Теоретичні аспекти отриманих у дипломному проекті результатів проходили апробацію на міжнародній науково-технічній конференції “Інтелектуальні технології лінгвістичного аналізу”

Публікації. Степанов А.В. Інструмент забезпечення анонімності в інтернеті (*TOR* мережа): міжнар. науково-техн. конф. «Інтелектуальні технології лінгвістичного аналізу», 20 – 21 жовтня 2020 р.: тези доп. – К., 2020. – С. 15.

Прогнозні припущення про розвиток об’єкту та предмету дослідження – застосування запропонованого алгоритму в системах генерації рекомендацій, базуючись на історії переглядів користувачів. Зараз проект запускається локально, наступним кроком у розробці цього додатку буде запуск на хмарній обчислювальній машині, щоб доступ до рекомендацій був увесь час. Також до проекту було б доцільно додати більше метрик рекомендацій, та підключити його до *back-end* частини сайту агенції нерухомості.

РОЗДІЛ 1
СЕРВІСИ НАДАННЯ РЕКОМЕНДАЦІЙ ДЛЯ ПОКУПЦІВ

1.1. Застосування сервісів надання рекомендацій для покупців

Рекомендаційні системи використовуються у різних сферах: рекомендація плейлісту для відео та аудіо сервісів, рекомендація продуктів у інтернет магазинах та рекомендація різного роду контенту у соціальних мережах. Наведені системи можуть працювати з використанням одного типу інформації на вході, наприклад, аудіодоріжка, яка сподобалась користувачу або декількох типів інформації: аудіо та відеоконтент, відгуки та оцінки користувачів або навіть атрибути продукту, який був придбаний клієнтом.

Майже кожна велика продуктова компанія має у своєму арсеналі рекомендаційну систему. Зазвичай рекомендації таких систем відображаються на сайті разом зі словами: “Рекомендуємо”, “Інші користувачі дивились також”, “Вас може зацікавити”, “Покупці, які дивились цей товар, також цікавились цим”, “Більше товарів для вивчення” і тд.

Задачі, які вирішують рекомендаційні системи:

– Покращення *UX* (*user experience*), завдяки наданню користувачу схожих продуктів, які його б зацікавили;
– Збільшення часу сесії користувача на сайті;
– Збільшити час сесії клієнта на сайті.

Наведені рішення комплексно приводять до загальної цілі рекомендаційних систем – покупка товару, а це завжди актуально для будь-якої компанії.

Типи рекомендаційних систем:

– Колаборативна фільтрація (*collaborative filtering*);
– Ті, які базуються на контенті (*content-based*);
– Ті, які базуються на знаннях (*knowledge-based*);
– Гібридні (*hybrid*).

Колаборативна фільтрація (*collaborative filtering*) – це тип рекомендаційних систем, у яких рекомендації базуються на історії оцінок користувачів про товар, це називається явна оцінка (*explicit*), або взаємодію користувача з ним (кліки, переходи на сторінки, покупка товару) – неявна оцінка (*implicit*). Використовуючи ці дані, система вираховує ‘схожих’ користувачів. Утворюється матриця взаємодій (рис. 1.1), базуючись на оцінках користувачів. Кожному користувачу присвоюється вага з урахуванням схожості його оцінок відносно активного користувача, потім знаходиться група людей, які ‘схожі’ на нього. Наприкінці підбирається відповідний контент, базуючись на оцінках групи користувачів та їх ваг.

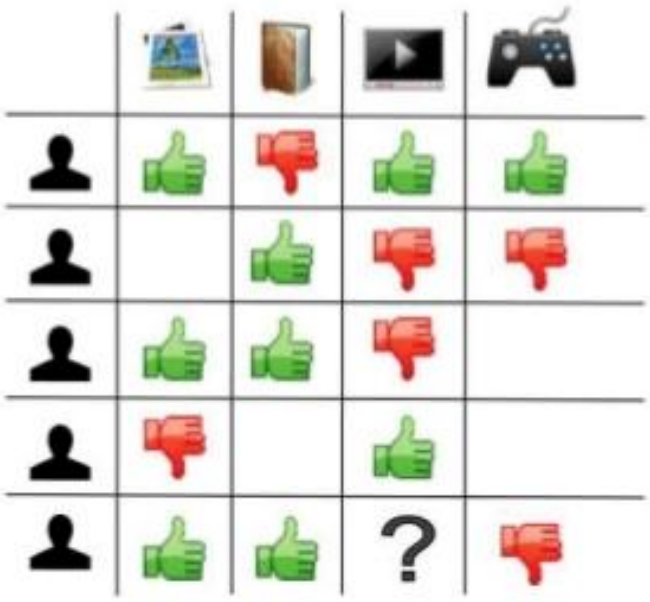

Рис. 1.1. Приклад матриці колаборативної фільтрації

Ті, які базуються на контенті (*content-based*) (рис. 1.2) – це тип рекомендаційних систем, у яких рекомендації базуються на особливостях товару та вподобаннях користувача для того щоб показати йому схожі товари.

Використовуючи ці дані, система знаходить схожі товари, які вже були оцінені користувачем. Для кожного користувача вираховується його вектор інтересі, потім знаходять *n*-близьких векторів товарів та рекомендують їх користувачу.

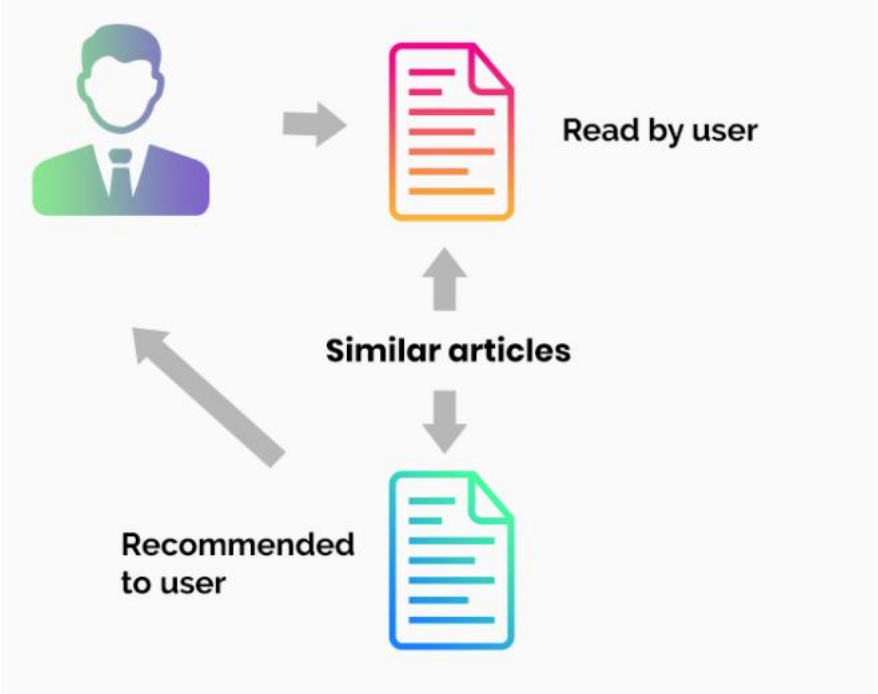

Рис. 1.2. Спрощена схема роботи *content-based* рекомендаційної системи

Ті, які базуються на знаннях (*knowledge-based*) – це рекомендаційні системи, які базуються на явних знаннях про асортимент товару та предметну область. Ця система не використовує особливості товару, або інтереси користувача. Такі системи часто дуже ефективні, але дуже складні у розробці. Приклад рекомендацій: при покупці ігрової консолі, система рекомендує користувачу докупити бездротовий геймпад або диск (рис. 1.3, рис 1.4).

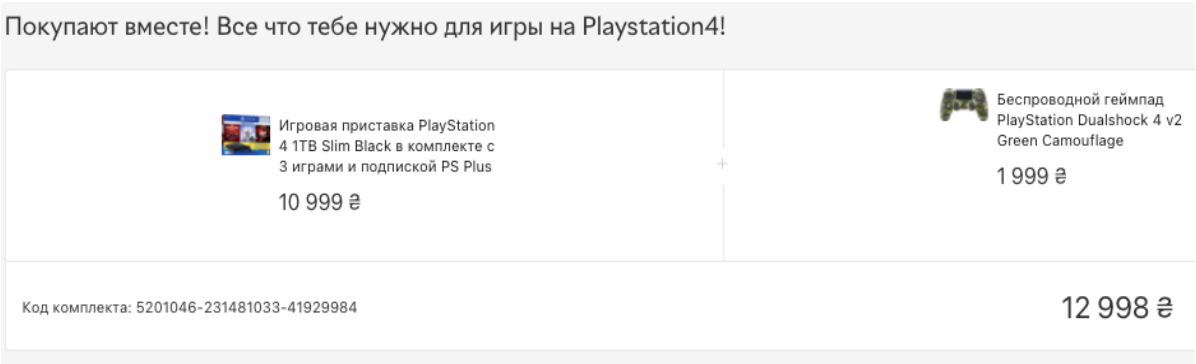

Рис. 1.3. Рекомендація покупки комплекту з геймпадом на сайті *rozetka.com.ua*

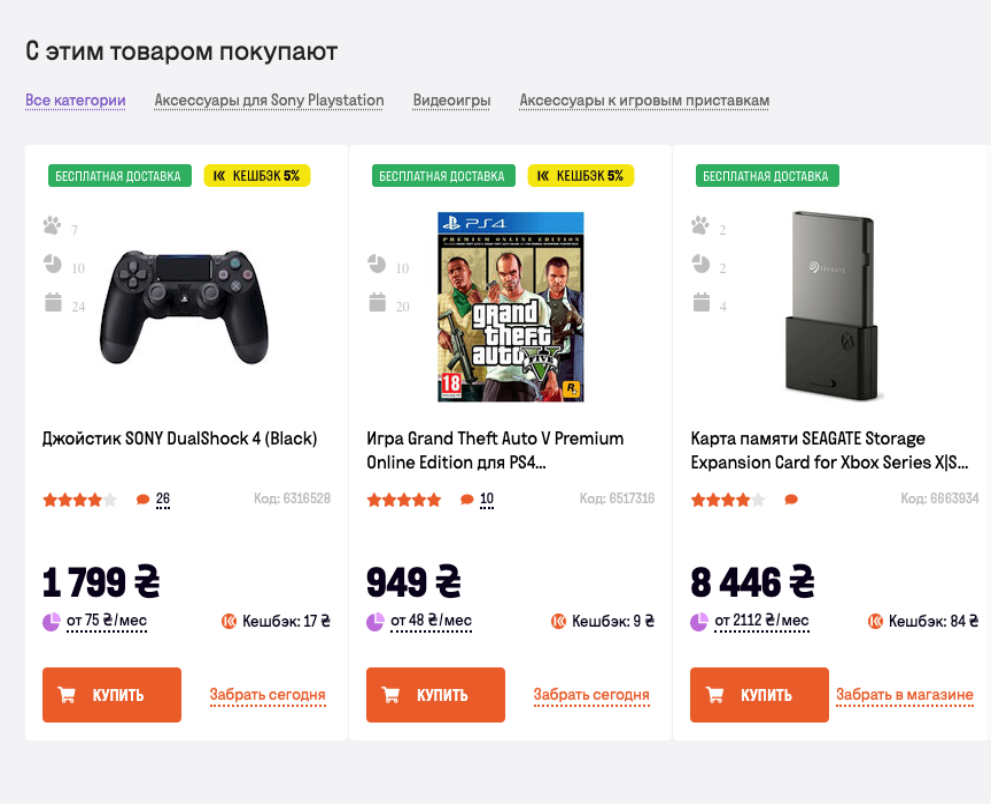

Рис. 1.4. Рекомендація покупки комплектів на сайті *foxtrot.com.ua*

Гібридні (*hybrid*) – це рекомендаційні системи, які вміщують в себе декілька типів для мінімізації недоліків того чи іншого типу. Великі компанії використовують такі системи, бо їх можна налаштовувати як заманеться, наприклад, *Netflix* використовує 27 алгоритмів у своїй системі. У гібридних системах немає універсального методу, все обмежується можливостями та фантазіями розробників. Але є декілька популярних типів комбінування:

– Реалізація колаборативних та контентних алгоритмів та об’єднання їх рекомендацій;
– Включення деяких контентних правил у колаборативний метод;
– Включення деяких колаборативних правил у контентний метод;
– Побудування загальної моделі, яка включає в себе правила обох метрик.

1.2. Аналіз сервісів генерації рекомендацій для покупців

1.2.1. *rozetka.com.ua*

rozetka.com.ua – найбільший інтернет магазин в Україні, який продає багато товарів з різних категорій від комп’ютерних комплектуючих до їжі (рис. 1.5, рис. 1.6).

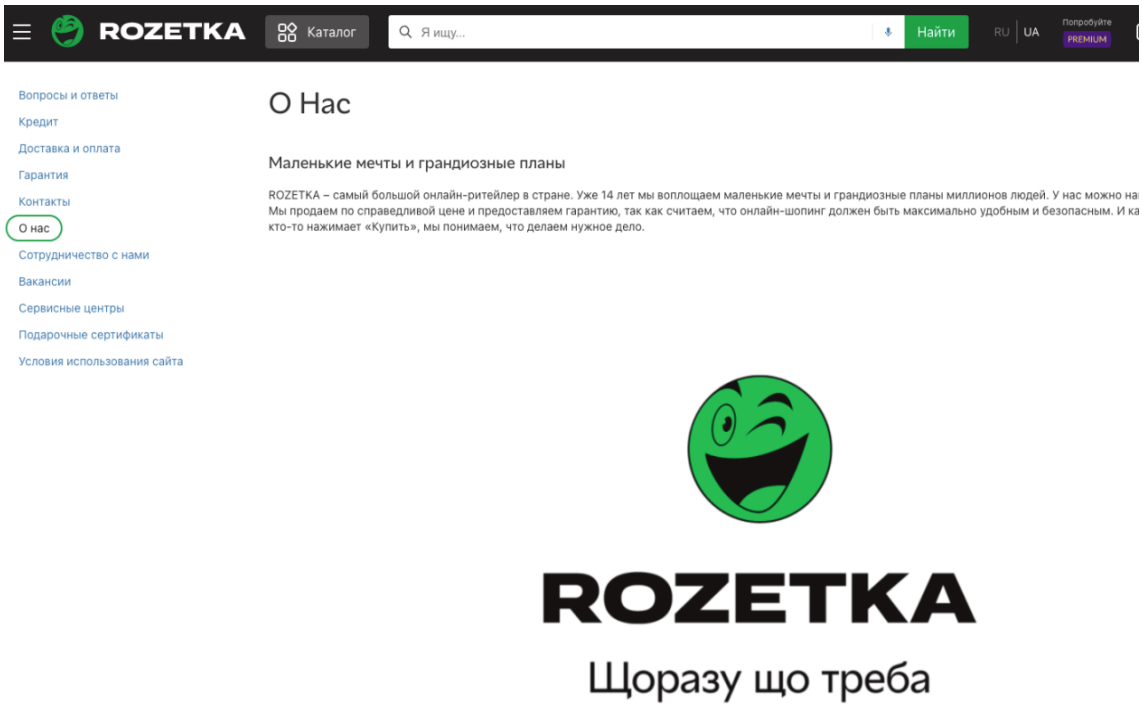

Рис. 1.5. Сторінка “О нас” на *rozetka.com.ua*

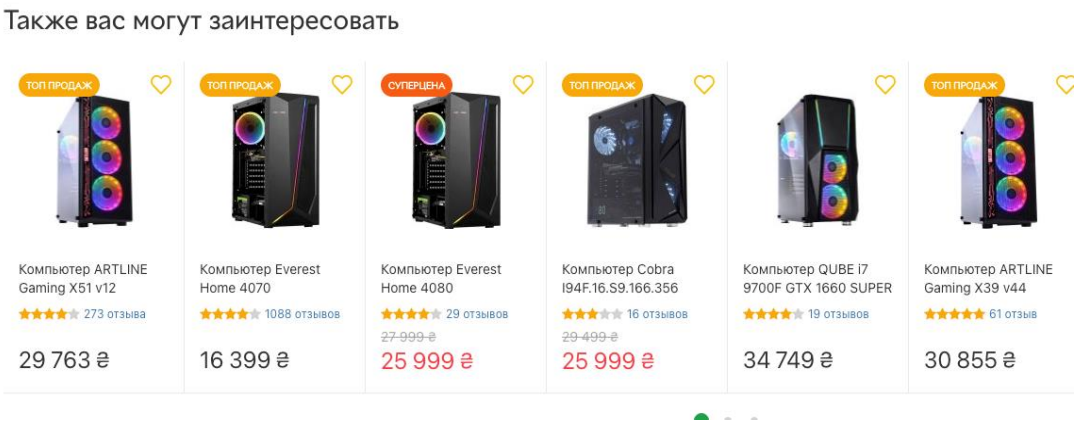

Рис. 1.6. Частина інтерфейсу, яка відображає рекомендації на *rozetka.com.ua*

Переваги: персоналізовані рекомендації для кожної категорії товару, точні рекомендації відносно товару, що був переглянутий користувачем та зручне розташування панелі рекомендацій.

Недоліки: сайт завантажується дуже довго, а панель з рекомендаціями ще довше.

1.2.2. *amazon.com*

amazon.com– найбільший інтернет магазин продажу різноманітних товарів у світі. Це найбільший у світі ринок електронної комерції, постачальник *AI*-асистентів і платформа хмарних обчислень, що вимірюється доходом і ринковою капіталізацією (рис 1.7, рис 1.8).

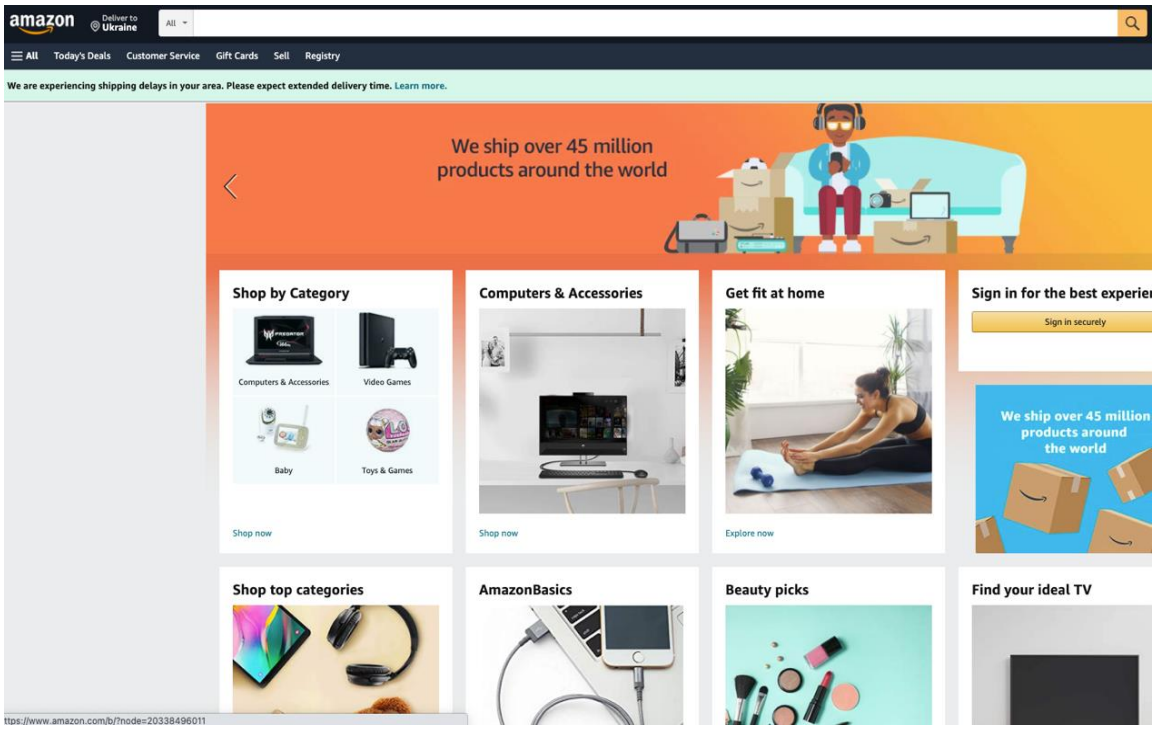

Рис. 1.7. Головна сторінка *amazon.com*

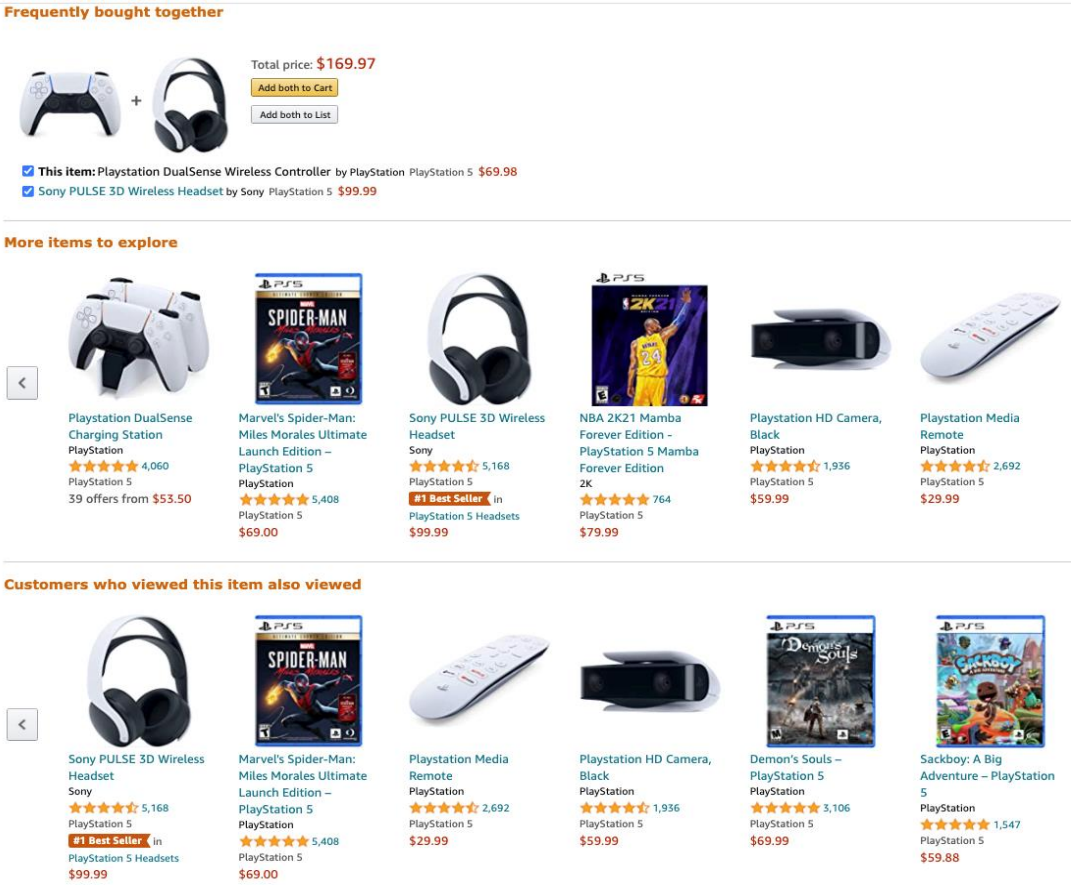

Рис. 1.8. Частина інтерфейсу, яка відображає рекомендації на *amazon.com*

Переваги: персоналізовані рекомендації для кожної категорії товару, точні рекомендації відносно товару, що був переглянутий користувачем. Швидко завантажується сайт та рекомендації.

Недоліки: перенавантажений інтерфейс, який впливає на зручність використання сайту.

1.2.3. *youtube.com*

youtube.com – найбільший та найпопулярніший відеохостінг, який надає користувачам змогу зберігати та показувати відео (рис. 1.9).

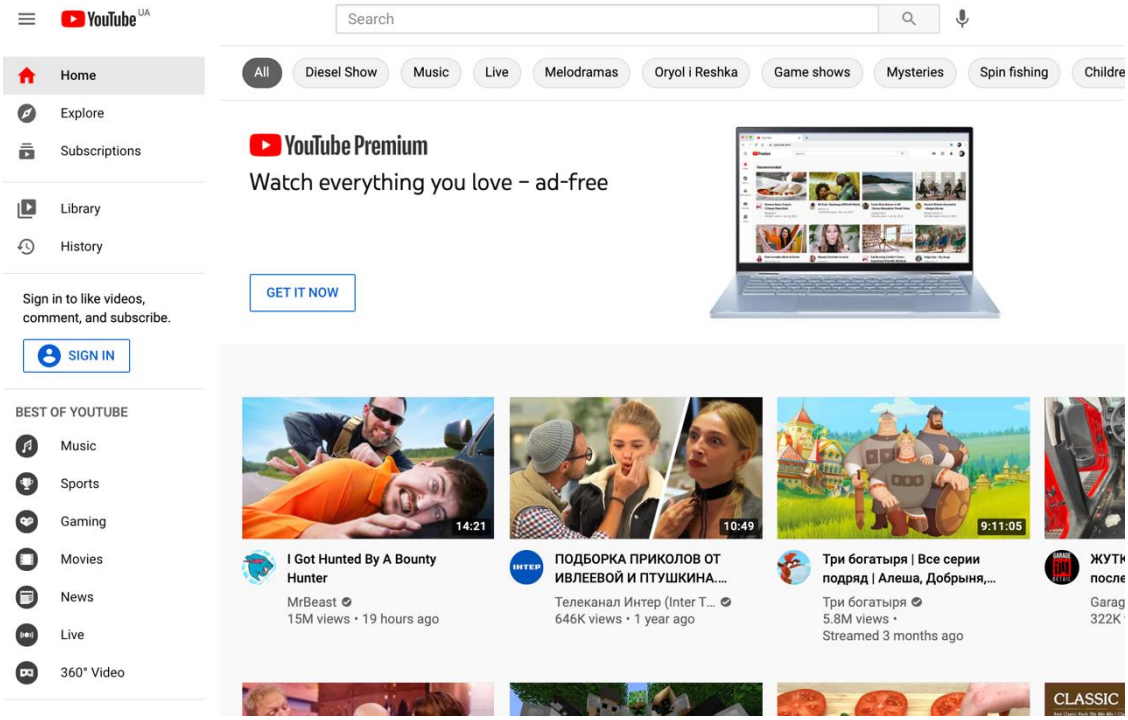

Рис. 1.9. Головна сторінка *youtube.com*, яка і являється сторінкою рекомендацій

Переваги: точні рекомендації відносно переглянутих відео користувачем, швидко завантажується сайт та рекомендації і зручна нескінченна сторінка рекомендацій.

Недоліки: рекомендації холодного старту являються не якісними.

1.2.4. *megogo.net*

megogo.net – медіасервіс для перегляду відео та телебачення, на його платформі представлено художні та документальні фільми, мультфільми, серіали та шоу, прямі трансляції ТБ і тд (рис. 1.10, рис. 1.11).

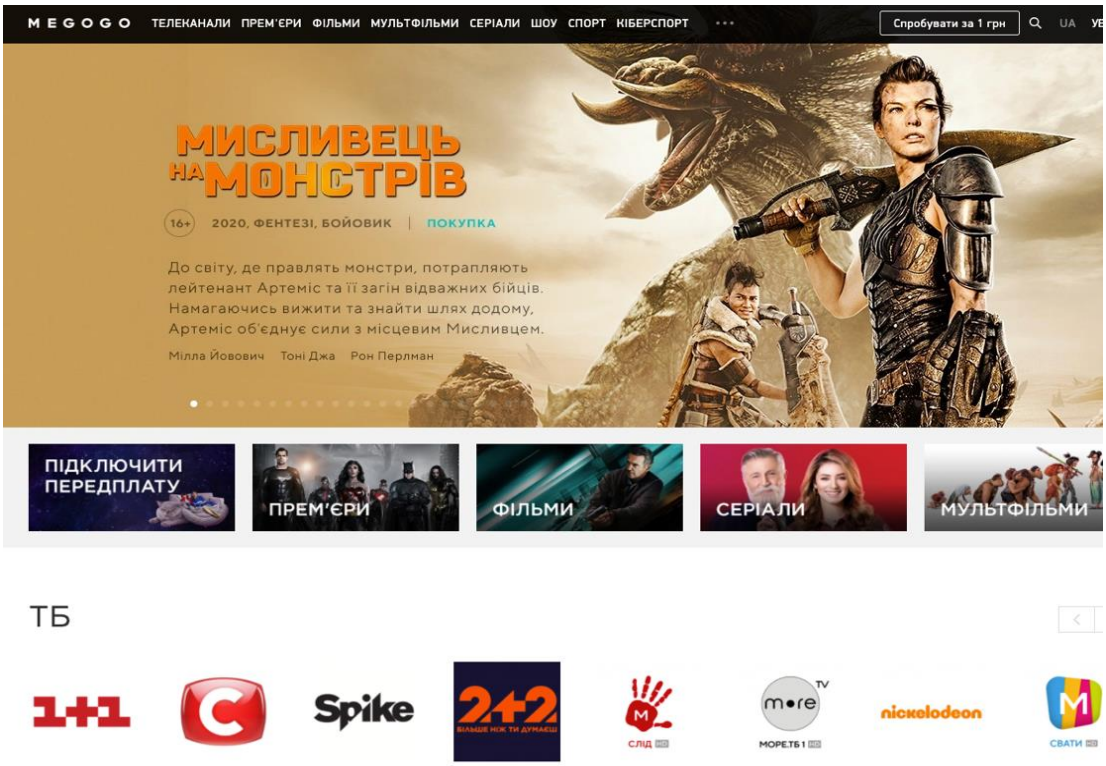

Рис. 1.10. Головна сторінка *megogo.net*

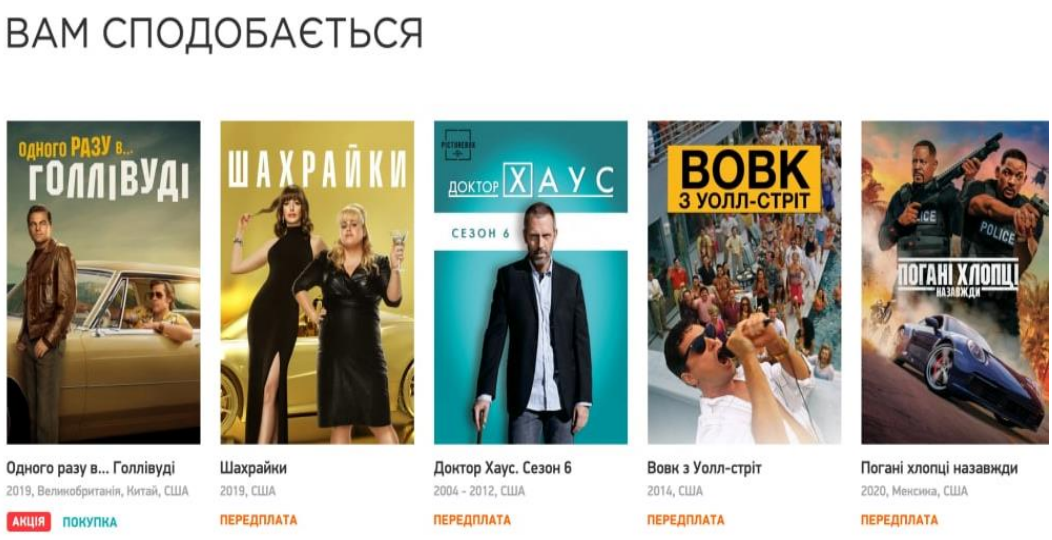

Рис. 1.11. Частина інтерфейсу, яка відображає рекомендації на *megogo.net*

Переваги: точні рекомендації відносно переглянутих фільмів, серіалів або передач користувачем, швидко завантажується сайт та рекомендації.

Недоліки: рекомендації холодного старту являються не якісними.

1.2.5. *zoopla.co.uk*

zoopla.co.uk – сайт для пошуку нерухомості у Британії (рис. 1.12, рис. 1.13).

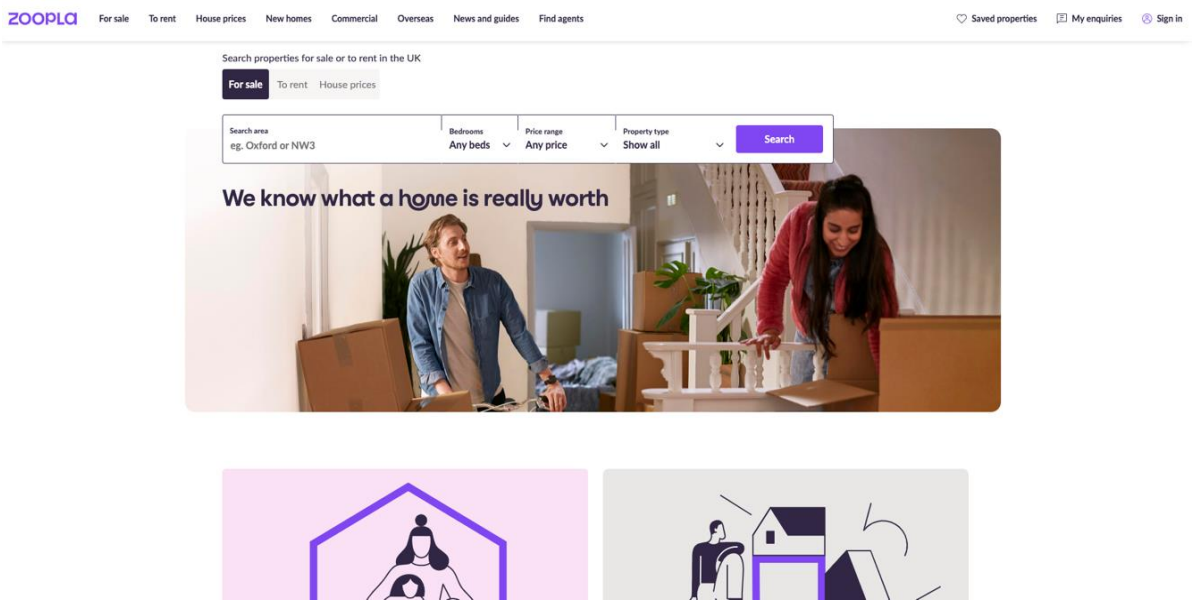

Рис. 1.12. Головна сторінка *zoopla.co.uk*

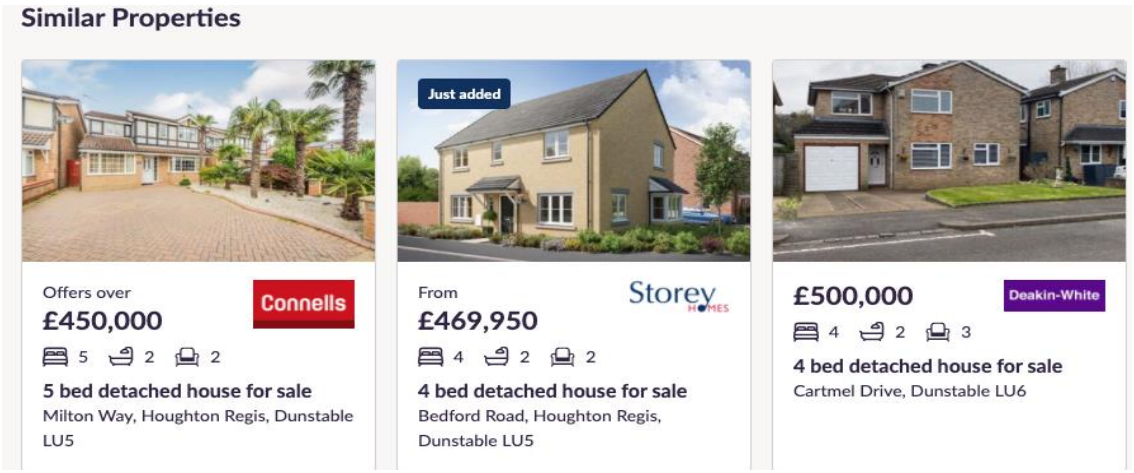

Рис. 1.13. Частина інтерфейсу, яка відображає рекомендації на *zoopla.co.uk*

Переваги: точні рекомендації відносно переглянутого оголошення користувачем і якісні рекомендації холодного старту.

Недоліки: рекомендації беруться відносно поточного оголошення, тобто згенеровані рекомендації це просто відфільтровані оголошення, які схожі на поточне, за увагу не береться історія пошуку користувача та інші атрибути. Сайт та рекомендації завантажуються дуже довго.

1.2.6. *onedome.com*

onedome.com – платформа для пошуку та придбання нерухомості (рис. 1.14, рис. 1.15).

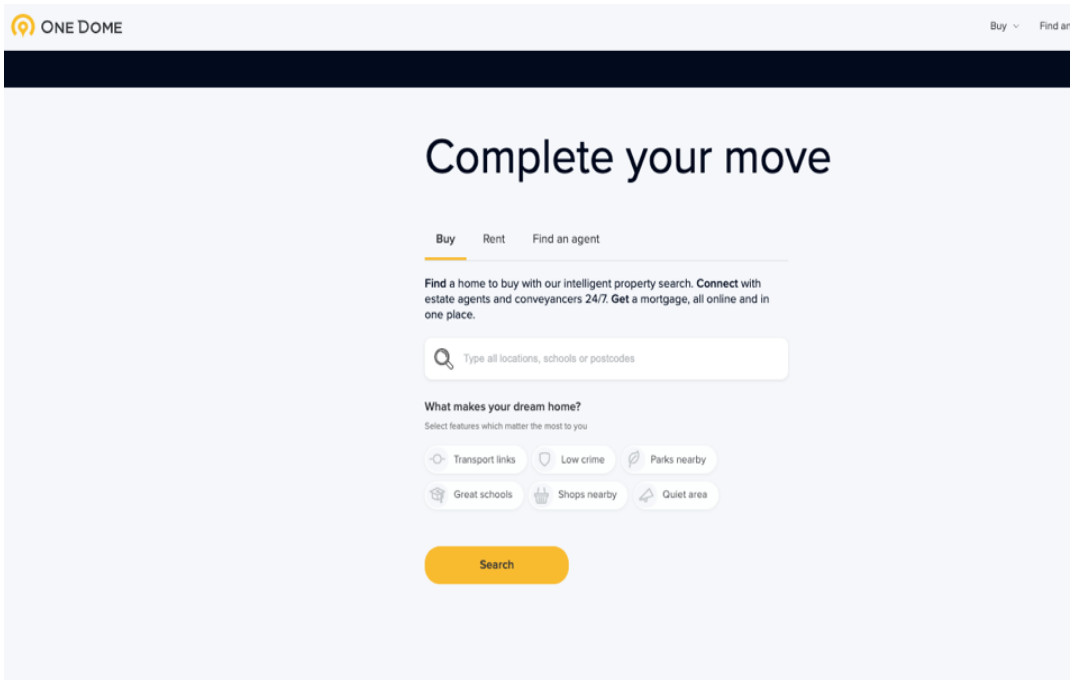

Рис. 1.14. Головна сторінка *onedome.com*

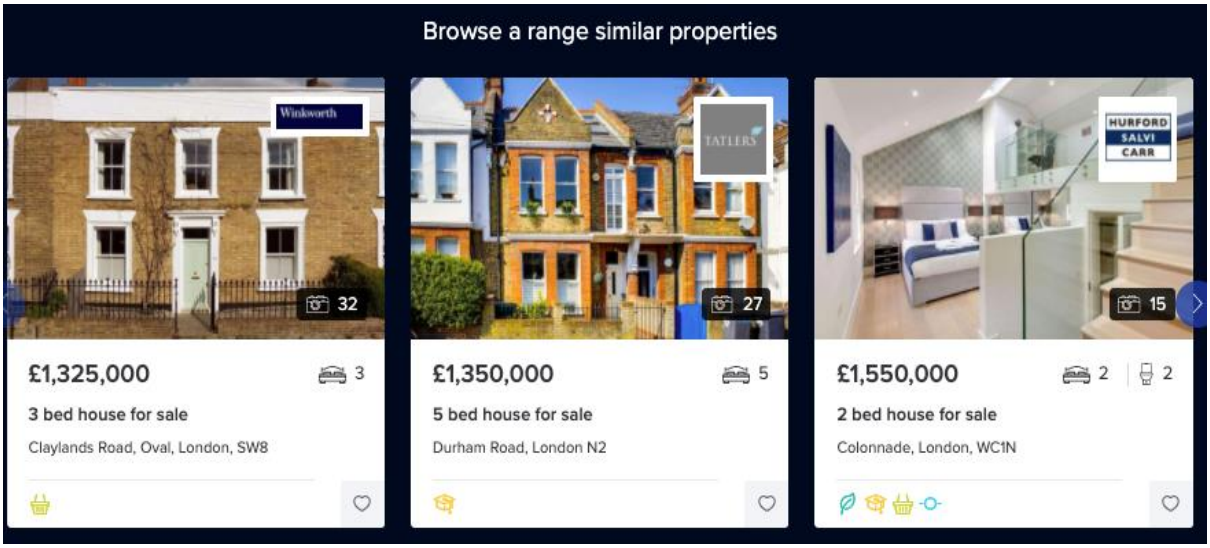

Рис. 1.15. Частина інтерфейсу, яка відображає рекомендації на *onedome.com*

Переваги: точні рекомендації відносно переглянутого оголошення користувачем, швидко завантажується сайт та рекомендації, рекомендації холодного старту - якісні.

Недоліки: рекомендації беруться відносно поточного оголошення, тобто згенеровані рекомендації це просто відфільтровані оголошення, які схожі на поточне, за увагу не береться історія пошуку користувача та інші атрибути.

Порівняння наведених сайтів наведено у таблиці 1.1.

Таблиця 1.1

Порівняння існуючих сервісів та їх рекомендацій

Назва сайту	Якісні рекомендації	Швидкість завантаження	Зручність	Якісний холодний старт
rozetka.com.ua	Так	Ні	Так	Так
amazon.com	Так	Так	Ні	Так
youtube.com	Так	Так	Так	Ні
megogo.net	Так	Так	Так	Ні
zoopla.co.uk	Так	Ні	Ні	Так
onedome.com	Так	Так	Ні	Так

На основі аналізу переваг те недоліків існуючих рекомендаційних систем, що застосовуються у різних сферах діяльності, у дипломному проекті було вирішено удосконалити роботу рекомендаційної системи *onedome.com* шляхом розробки програми, яка буде враховувати не тільки поточне оголошення, а й історію переглядів користувача та додаткові атрибути оголошень. Тобто необхідно розробити додаток, що буде прораховувати рекомендації раз у день, базуючись на історії пошуку користувача та додаткові атрибути оголошень. Додатково буде використовуватись вже існуючий ‘*similar properties*’ підхід для холодного старту.

Завдання для вирішення у дипломному проекті:

– ознайомитися з існуючими реалізаціями та підходами щодо створення рекомендацій;

– вивчити алгоритми генерації даних для формування рекомендацій;
– дослідити структуру бази даних для генерації рекомендацій;
– створити програму генерації рекомендацій покупцю під час придбання нерухомості;
– проаналізувати та оцінити метрики розробленої програми.

1.3. Висновки до розділу

У першому розділі дипломного проекту розглянуто існуючі рекомендаційні системи та наведена їх класифікація. Залежно від типу рекомендацій та принципу їх формування розрізняють системи колаборативної фільтрації; системи, що базуються на контенті; системи, що базуються на знаннях та гібридні рекомендаційні системи.

Було розглянуто найпопулярніші сайти різної тематики, у яких присутні рекомендаційні системи, здійснено аналіз їх переваг та недоліків. Для порівняння було обрано маркетплейси *rozetka.com.ua, amazon.com,* медіасервіси *youtube.com, megogo.net* та сайти пошуку нерухомості *zoopla.co.uk, onedome.com.*

Виходячи з побажань замовника та базуючись на аналізі поточного стану предметної області було постановлено задачу: розробити додаток, що буде прораховувати рекомендації раз у день, базуючись на історії пошуку користувача та додаткові атрибути оголошень. Додатково буде використовуватись вже існуючий ‘*similar properties*’ підхід для холодного старту.

Наведений проект розробляється як частина сайту нерухомості та доповнює його.

РОЗДІЛ 2
ТЕХНОЛОГІЇ ТА АЛГОРИТМИ ФОРМУВАННЯ ДАНИХ І ГЕНЕРАЦІЇ РЕКОМЕНДАЦІЙ

2.1. Структура бази даних

Для створення рекомендацій потрібно мати деякий набір даних, зокрема матрицю взаємодій користувачів та оголошень. Рекомендації будуть базуватись на взаємодіях користувача з оголошенням, а саме переходах на детальну сторінку оголошення на сайті onedome.com, та на атрибутах оголошень. Наведені дані зберігаються у *Amazon Athena* (рис. 2.1) базі даних. *Amazon Athena* – це інтерактивна служба запитів, яка дозволяє аналізувати дані у *Amazon S3*, використовуючи стандартний *SQL*. *Amazon S3* – це служба зберігання об'єктів, іншими словами це хмарове сховище даних. Тобто сервіси записують дані у сховище *S3*, а за допомогою *Athena* можна дістати та відфільтрувати потрібні нам події та записи.

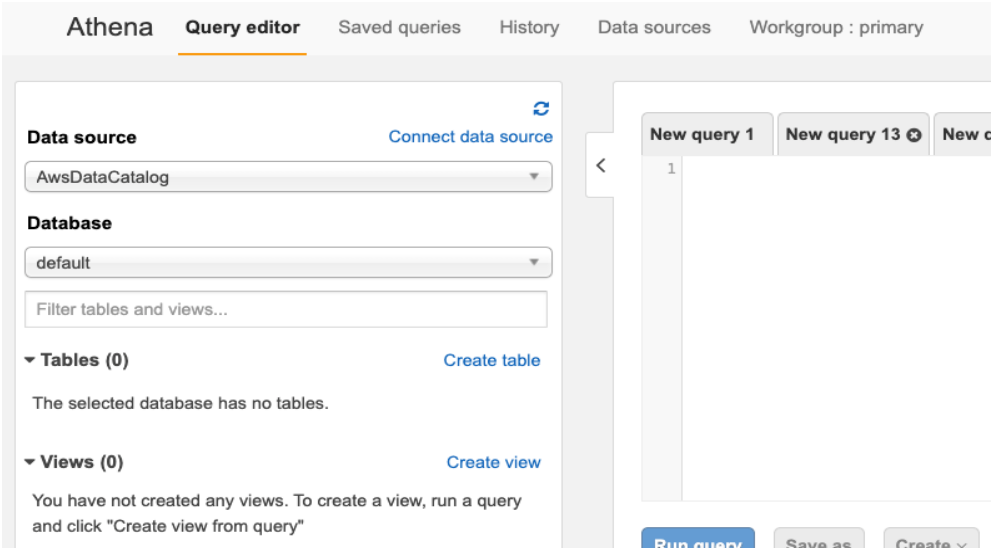

Рис. 2.1. Веб-інтерфейс *Amazon Athena*

Кафедра КСУ				НАУ 21 21 30 000 ПЗ			
Виконав	*Степанов А.В.*			Технології та алгоритми формування даних і генерації рекомендацій	*Літера*	*Аркуш*	*Аркушів*
Керівник	*Вавіленкова А.І.*				*Д*	*22*	*58*
Консульт.					СП-435 123		
Норм. контр.	*Тупота Є.В.*						
Зав. Каф.	*Литвиненко О.Є.*						

Буде використано три таблиці, опис полів яких наведено у табл.2.1, табл. 2.2 та табл.2.3:

– *listings* (рис. 2.2) – зберігає всі оголошення та їх атрибути за кожний день;

– *log_events_pq* (рис. 2.3) – зберігає події та їх назви;

– *log_properties_pq* (рис. 2.4) – зберігає атрибути, які записуються з тою чи іншою подією;

Рис. 2.2. Структура таблиці *listings Amazon Athena*

Рис. 2.3. Структура таблиці *log_events_pq Amazon Athena*

Рис 2.4. Структура таблиці *log_properties_pq Amazon Athena*

Таблиця 2.1

Опис використаних полів таблиці *listings Amazon Athena*

Поле	Опис
uuid	Унікальний ідентифікатор оголошення
action_type	Тип оголошення (продаж/оренда)
address_postal_code	Поштовий індекс оголошення
price	Ціна оголошення
property_type	Тип оголошення (дім, квартира, пентхаус і тд)
number_of_bedrooms	Кількість спальних кімнат
stc	Маркер, який ідентифікує статус оголошення (продано/актуально)
listing_source	Джерело оголошення
indices	Оцінки *Locality Reality*

Таблиця 2.2

Опис використаних полів таблиці *log_events_pq Amazon Athena*

Поле	Опис
log_uuid	Унікальний ідентифікатор події
timestamp	Часова мітка, яка вказує коли відбулась подія
cid	Ідентифікатор користувача

Таблиця 2.3

Опис використаних полів таблиці *log_properties_pq Amazon Athena*

Поле	Опис
property_name	Назва атрибуту події
property_value	Значення атрибуту події

Для формування запитів у БД формуємо окрему функцію, яка буде приймати на вхід запит та об’єкт підключення:

```
def fetchall_athena(query_string, db):
    query_id = athena_client.start_query_execution(
        QueryString=query_string,
        QueryExecutionContext={
            'Database': db
        },
        ResultConfiguration={
            'OutputLocation': ''
        }
    )['QueryExecutionId']
    query_status = None
    while query_status == 'QUEUED' or query_status == 'RUNNING' or query_status is None:
```

```
        query_status = athena_client.get_query_execution(QueryExecutionId=query_id)['QueryExecution']['Status']['State']
        if query_status == 'FAILED' or query_status == 'CANCELLED':
            raise Exception('Athena query with the string "{}" failed or was cancelled'.format(query_string))
        time.sleep(3)

    results_paginator = athena_client.get_paginator('get_query_results')
    results_iter = results_paginator.paginate(
        QueryExecutionId=query_id,
        PaginationConfig={
            'PageSize': 1000
        }
    )
    results = []
    data_list = []

    for results_page in results_iter:

        for row in results_page['ResultSet']['Rows']:
            data_list.append(row['Data'])

    for datum in data_list:
        results.append([x['VarCharValue'] if 'VarCharValue' in x else '' for x in datum])
    result = pd.DataFrame.from_records(results[1:], columns=results[0])

    return result
```

Наведений код відправляє запит у *Athena* та чекає поки він завершиться, після чого ітерує через результат та конвертує дані. Результатом цієї функції являється таблиця у вигляді об’єкту бібліотеки *pandas*.

Для фільтрації та вилучення потрібних даних відправляємо, наведеній вище, функції два *SQL* запити:

```
SELECT le.log_uuid, le.timestamp, le.cid, lp.property_name, lp.property_value, le.event_name
    FROM telemetry.log_events_pq le
    LEFT JOIN telemetry.log_properties_pq lp ON le.log_uuid = lp.log_uuid
    WHERE le.event_name IN ('USER_VIEW_PROPERTY_PAGE_NEW_HOME', 'USER_VIEW_PROPERTY_PAGE')
    AND le.cid != ''
```

```
SELECT l.uuid, l.action_type, l.address_postal_code, l.location_point, l.price, l.property_type, l.number_of_bedrooms, l.stc,
    CASE WHEN l.listing_source = 'HomeBuilders' THEN true ELSE false END AS new_build,
    json_extract(l.indices, '$.green') AS green_index, json_extract(l.indices, '$.grocery') AS grocery_index,
    json_extract(l.indices, '$.education') AS education_index, json_extract(l.indices, '$.quiet') AS quiet_index,
    json_extract(l.indices, '$.transport') AS transport_index, json_extract(l.indices, '$.safety') AS safety_index
FROM listings AS l
INNER JOIN (
    SELECT uuid, max(date) AS max_date
    FROM listings
    GROUP BY uuid
) AS l2  ON l.uuid = l2.uuid AND l.date = l2.max_date
```

Перший запит вилучає всі події з ім’ям *USER_VIEW_PROPERTY_PAGE_NEW_HOME*, *USER_VIEW_PROPERTY_PAGE* – події переходу користувача на детальну сторінку оголошення, а другий вилучає всі унікальні оголошення з атрибутами, які перелічені у табл. 2.1.

2.2. Технології та інструменти використані для розробки

2.2.1. Аналіз оркестраторів задач

Для відпрацювання великої задачі за розкладом, потрібен оркестратор задач. Було виокремлено наступні оркестратори задач (рис. 2.5): *Luigi*, *Airflow*, *Kubeflow*.

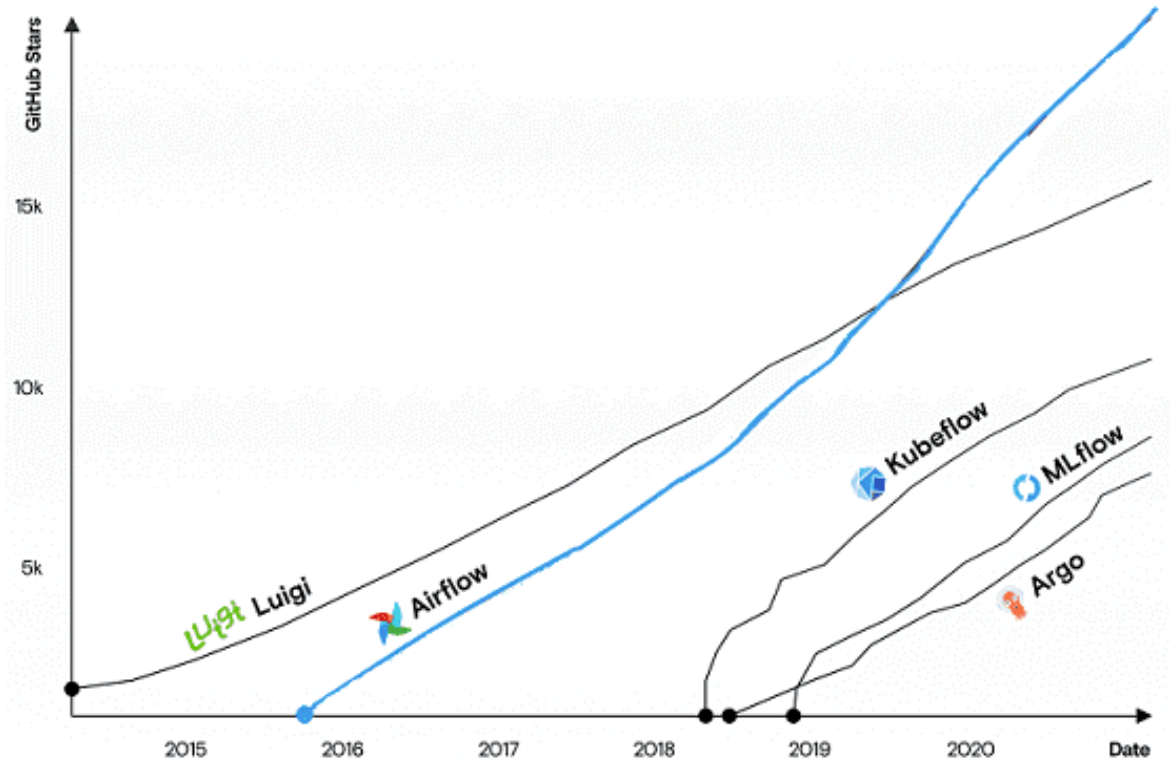

Рис. 2.5. Графік популярності оркестраторів задач

Luigi (рис. 2.6, рис. 2.7) – оркестратор задач, ціль якого є вирішити всі проблеми, які зазвичай пов’язані з довгими пакетними процесами. Завдяки йому, можна створити будь-яку задачу та пов’язати її з іншою, утворивши ланцюг задач.

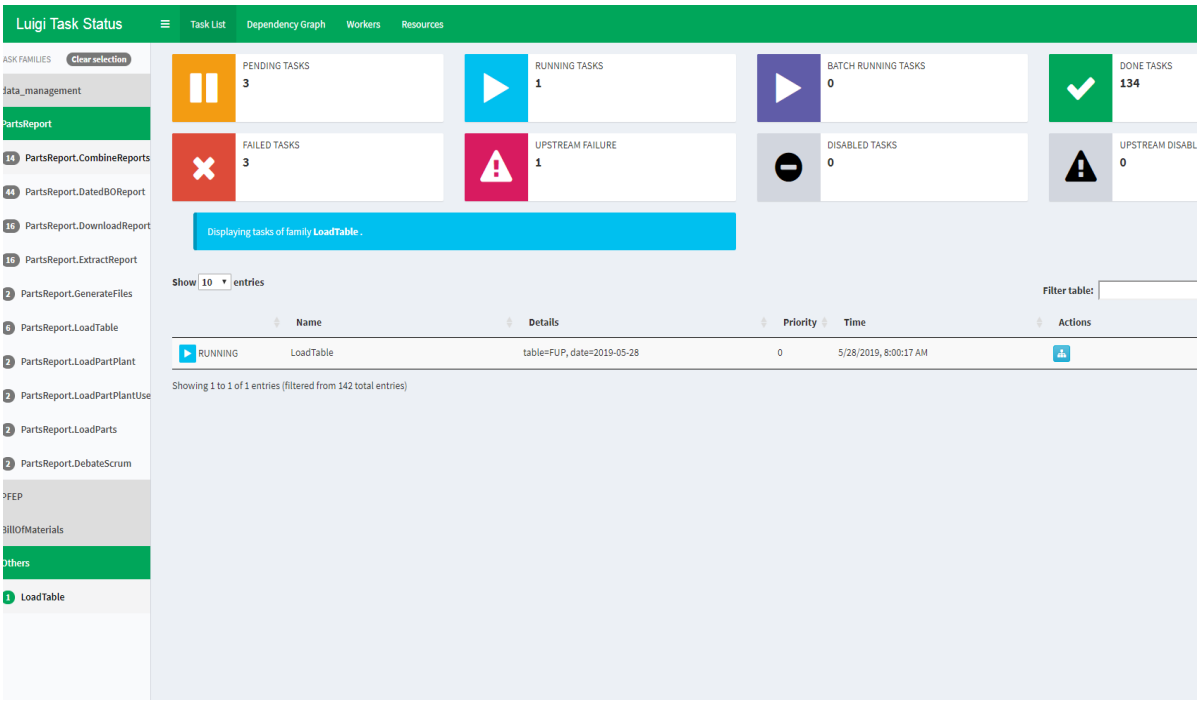

Рис. 2.6. Веб-інтерфейс головної сторінки *Luigi*

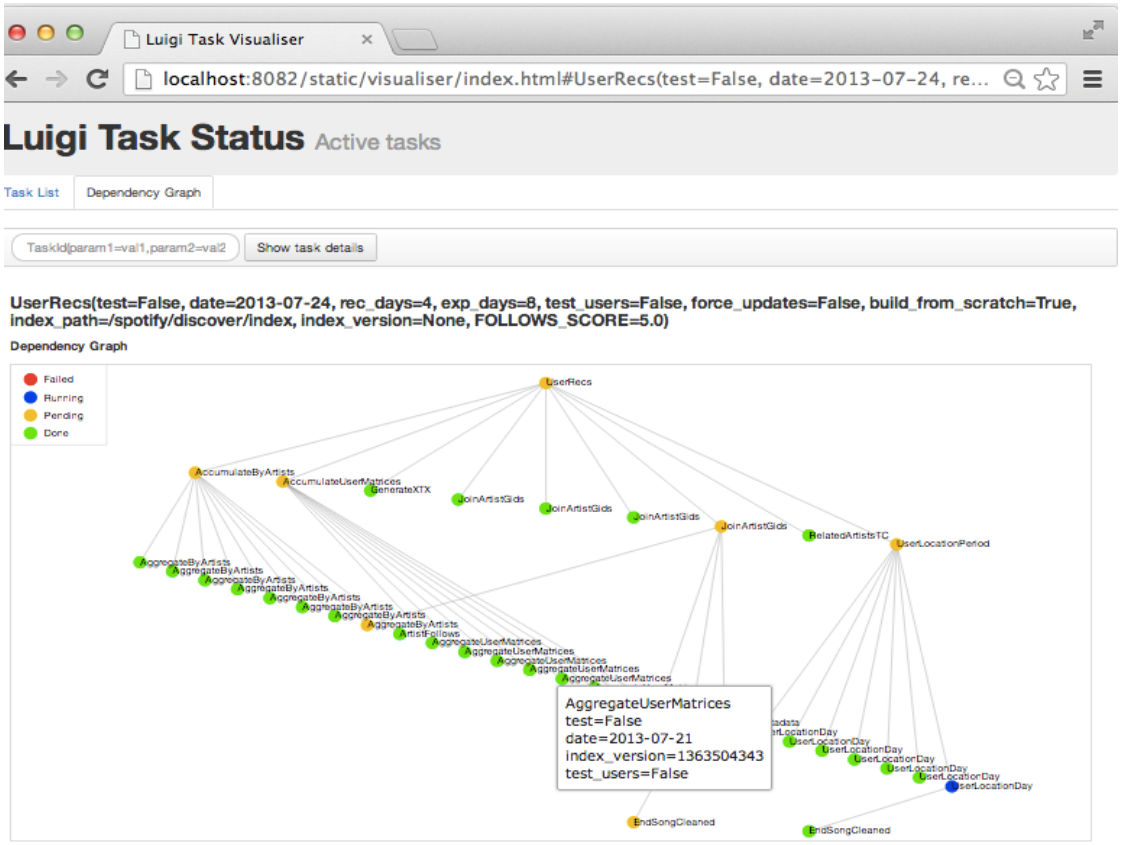

Рис. 2.7. Веб-інтерфейс ієрархії задач *Luigi*

Airflow (рис. 2.8, рис. 2.9) – оркестратор задач для розробки планування та моніторінгу робочих процесів. Для опису процесів використовується мова *Python*. Підтримує всі популярні обробники даних та бази даних.

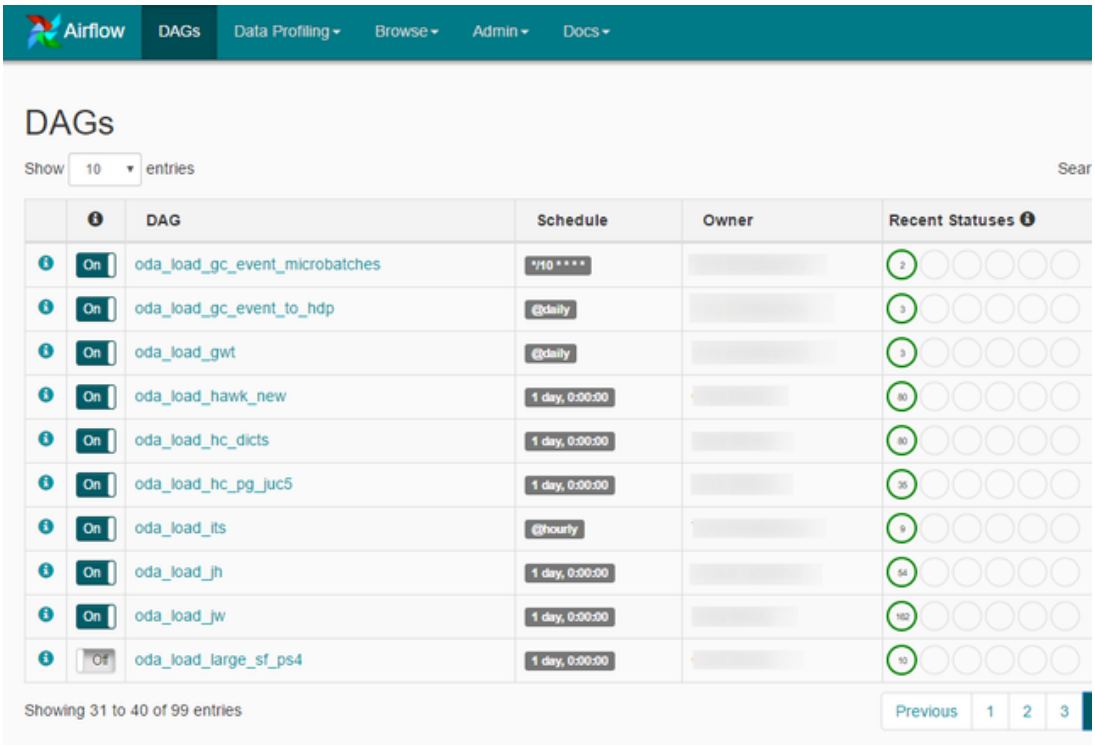

Рис. 2.8. Веб-інтерфейс головної сторінки *Airflow*

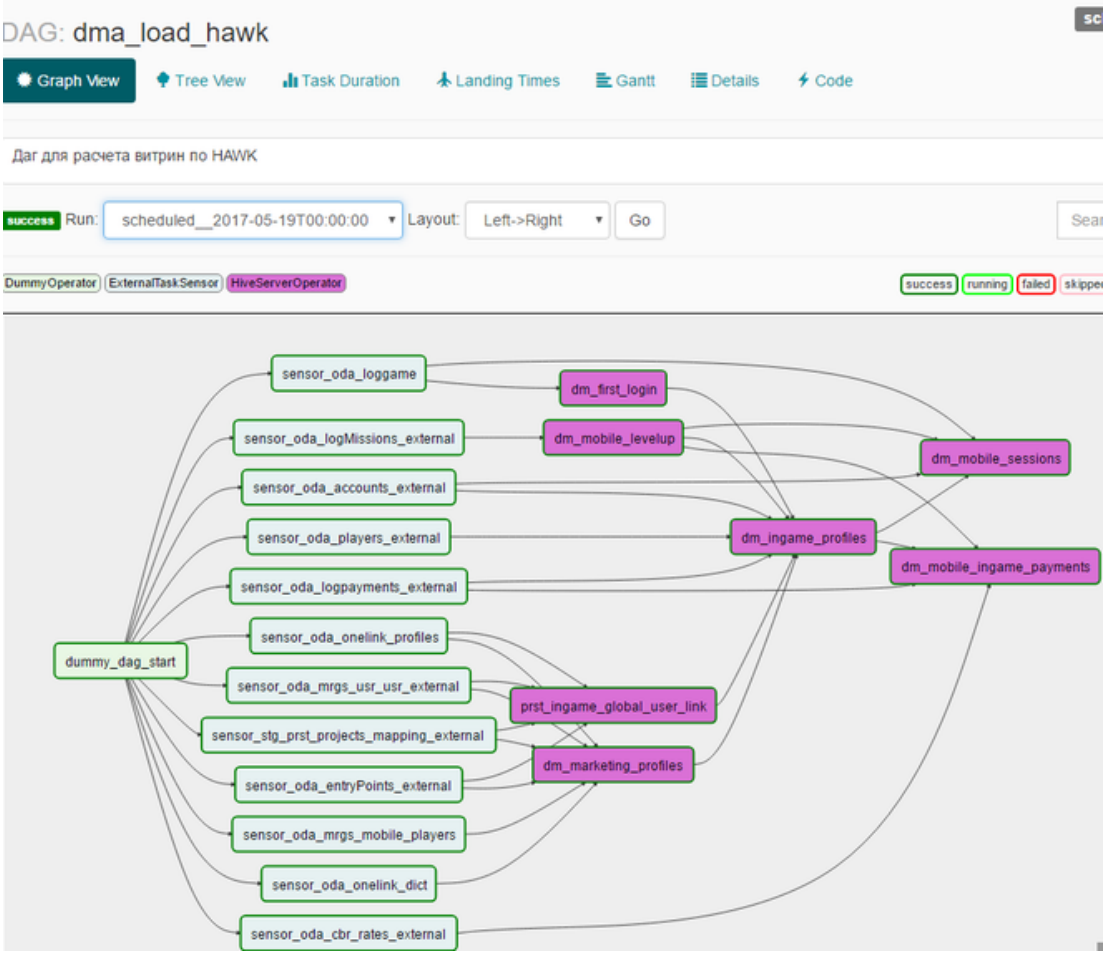

Рис. 2.9. Веб-інтерфейс ієрархії задач *Airflow*

Kubeflow – оркестратор задач у якому всі задачі контейнеризуються та запускаються як самостійний модуль, фокусується на операціях машинного навчання, слідкування за експериментами, налаштування гіперпараметрів та розгортання моделі.

Виходячи з потреб дипломного проекту, тобто планування та моніторінгу робочих процесів під час придбання нерухомості для подальшої генерації рекомендацій, було обрано *Airflow*, так як це найпопулярніший інструмент та його дуже активно підтримують розробники. Крім цього він має дуже зручний інтерфейс та документацію.

2.2.2. Програмні інструменти для побудови рекомендацій вирази

Існує багато бібліотек для побудови різноманітних рекомендацій. Для реалізації дипломного проекту було виокремлено такі бібліотеки: *Lenskit, Crab, Surprise, TensorRec, LightFM, Case Recommender*.

Lenskit – набір інструментів для побудови, пошуку та навчання про рекомендаційні системи. Допомагає будувати надійні та відтворювані експерименти.

Crab – гнучкий та швидкий рекомендаційний двигун, який вміщує в собі класичні рекомендаційні алгоритми з базових пакетів *numpy, Scipy, Matplotlib*.

Surprise – набір інсрументів для побудови та аналізу рекомендаційних систем. Працює з явною оцінкою (*explisit*). Має в собі різноманітні готові для використання алгоритми передбачення, такі як базові алгоритми, методи сусідства, методи факторизації і тд.

TensorRec – рекомендаційна система яка дозволяє швидко розробляти рекомендаційні алгоритми та налаштовувати їх, використовуючи *TensorFlow*. На вхід приймає три частини даних: особливості користувачів, особливості продукту, взаємодії. Вона використовує ці дані для передбачення оцінки, навчаючись на порівняні оцінок на виході з реальними взаємодіями.

LightFM – імплементація популярних рекомендаційних алгоритмів явної та неявної оцінки у одній бібліотеці. Легка у використанні та дуже швидка завдяки написанні на *Cpython* та підтримки багатопоточності.

Case Recommender – імплементація популярних рекомендаційних алгоритмів у одному фреймворку. Цей фреймворк має різноманітні типи продуктової рекомендації, передбачення оцінки та валідації метрик.

Для реалізації дипломного проекту було обрано бібліотеку *LightFM*, через швидкість та легкість використання, крім цього це практично єдина бібліотека яка підтримує навчання ранжуванню з *WARP* втратою (*Weighted-Approximate-Rank-Pairwise*).

Для створення веб-додатку, який буде повертати топ 10 рекомендацій заданому користувачу, потрібен асинхронний та швидкий веб-фреймворк. Було виокремлено наступні веб-фреймворки: *Tornado, FastAPI, Sanic*.

Tornado – асинхронний веб-фреймворк, який надає базові можливості створення асинхронного додатку. Був створений у 2009, і не являється веб-структурою, це колекція асинхронних модулів, які використовуються для створення веб-фреймворку.

FastAPI – один з найновіших та найшвидших веб-фреймворків, який має приємний синтаксис та дуже багато документації. Крім цього підтримує автоматичну генерацію документації для створених кінцевих точок додатку.

Sanic – теж один з найновіших веб-фреймворків, який підтримує *Python* не нижче 3.6, має простий синтаксис. Розробка схожа на розробку *Flask*.

В подальшому у дипломному проекті використовується бібліотека *FastAPI*, через швидкість та легкість використання.

2.3. Алгоритм формування рекомендацій для покупця

Для формування рекомендацій буде використовуватись бібліотека *LightFM* з *loss* функцією *WARP*.

2.3.1. Модель *LightFM*

Модель *LightFM*, являється гібридною рекомендаційною системою прихованої оцінки, вона вивчає вбудовування (*embeddings*, приховані представлення у багатошаровому просторі), для користувачів та продуктів таким чином, що кодує переваги користувачів поверх продуктів. Помножені разом, ці подання дають оцінки для кожного продукту для поточного користувача. Продукти, чиї оцінки найвищі, швидше за все зацікавлять клієнта.

LightFM використовує модель матричної факторизації, яка розкладає матрицю взаємодій на дві матриці (користувач та продукт), так що при їх множенні зберігається вихідна матриця взаємодій. На рис. 2.10 ілюструється це поняття графічно. Факторизовані матриці відомі як вбудування користувачів та продуктів, вони мають однакову кількість рядків (прихована векторна розмірність), але різну кількість стовпчиків залежно від кількості користувачів або продуктів.

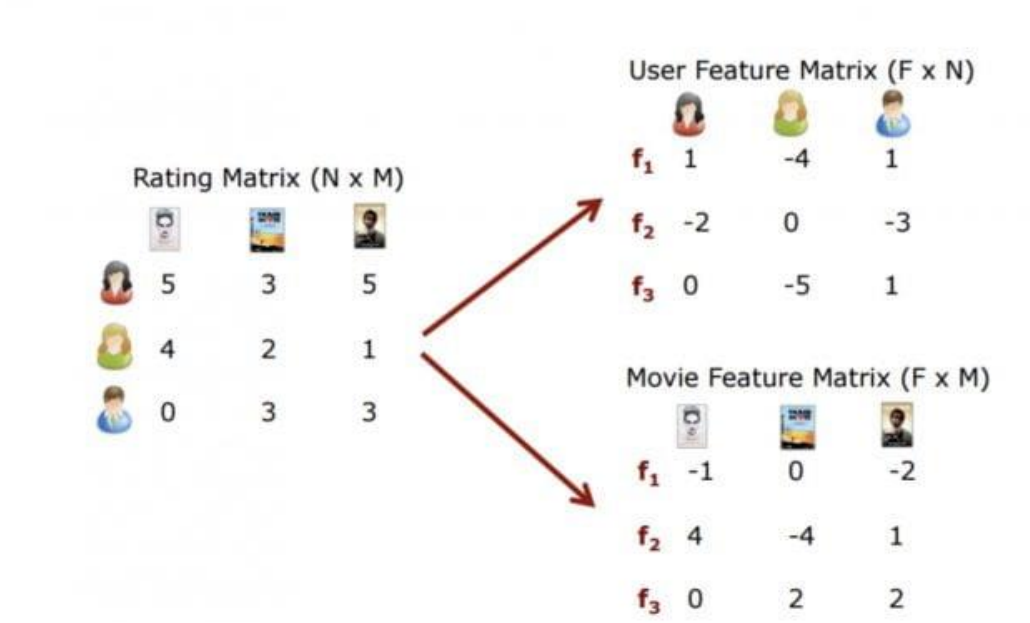

Рис. 2.10. Матрична факторизація

Приховані вбудування можуть фіксувати приховані особливості щодо атрибутів користувачів та продуктів, що відображають їх вподобання. Вбудування прораховується для кожної особливості – e_j (де j=1,k, k – кількість особливостей) і потім вони підсумовуються для отримання представлень користувачів та продуктів. Нехай e_j^U - вбудування користувача, e_j^I – вбудування продукту, J_u –

множина номерів ознак користувачів та J_i – множина номерів ознак продуктів. Тоді користувача можна охарактеризувати параметром:

$$q_u = \sum_{j \in J_u} e_j^U,$$

а продукт:

$$p_i = \sum_{j \in J_i} e_j^I,$$

де значення e_j є нормалізованим та знаходиться у діапазоні [0, 1].

Кожна ознака описується терміном скалярного зміщення b_j^U – для користувача та b_j^I – для продукту. Зміщення користувача та продукту прораховується сумою зміщень ознак.

Формула підсумування зміщення для користувача:

$$b_u = \sum_{j \in f_u} b_j^U,$$

та для продукту:

$$b_i = \sum_{j \in f_i} b_j^I,$$

Прогноз (рекомендація) для користувача u та продукту i прораховується добутком параметрів, що характеризують користувача та продукт, скоригованим відповідними зміщеннями:

$$r_{ui} = f(q_u * p_i + b_u + b_i),$$

де f – функція втрати (для дипломного проекту була обрана *WARP*-функція);

i=1,X;

X – кількість запропонованих продуктів;

u=1,N;

N – кількість клієнтів, для яких генеруються рекомендації.

Чим вище значення r_{ui} тим більша вірогідність того, що даний продукт сподобається користувачу.

2.3.2. *WARP* втрата

Даний метод відбирає елементи вихідного вектору моделі, поки не знайде пару, яка, як відомо, неправильно оцінена, а потім застосовує оновлення до цих двох неправильно позначених прикладів.

З рис. 2.11 видно, що у цільовому векторі користувач взаємодіяв з продуктом ‘*Milky Way*’, а вихідний вектор був побудований тою чи іншою моделлю або нейронною мережею. Використовуючи даний метод, потрібно порівнювати оцінки, про які в нас є дані, та оцінки, про які немає даних у вихідному векторі.

Рис. 2.11. Приклад використання *WARP* втрати

Тобто порівнюємо оцінки продуктів ‘*Milky Way*’ та ‘*Kit Kat*’: 0.59 < 0.63.

В даному випадку виходить, що передбачення неправильне, бо виходячи з цієї логіки користувач швидше за все придбає ‘*Kit Kat*’ ніж ‘*Milky Way*’, що не є вірно. Вираховуємо *WARP* втрату за формулою:

$$Loss = (r_{ui}^{-} - r_{ui+1}^{+}) = (0.63\text{-}0.59) = 0.04$$

Для того, щоб не переглядати всі пари, що зайняло би багато ресурсів, можна переглядати не всі, а випадкові пари. Якщо було оброблено багато пар до того моменту, коли знайшлась неправильна пара, можна казати, що передбачення працює правильно та точно, але з іншої сторони, якщо перша пара була не правильною, то можна казати що передбачення працює неточно. Тому домножуємо втрату на:

$$\ln\left(\frac{X-1}{N}\right),$$

де X – загальна кількість оцінок; тобто запропонованих продуктів (5 у прикладі);

N – кількість пар, необхідних для пошуку прикладу, де модель помилилась (1 у прикладі).

Це має сенс, оскільки доводиться брати більше пар для пошуку, щоб втрата була мінімальною. Тепер формула *WARP* виглядає так:

$$Loss = \ln\left(\frac{X-1}{N}\right)(r_{ui} - r_{ui+1}),$$

де r_{ui} та r_{ui+1} оцінки з вихідного вектору.

Після цього тільки оцінки неправильної пари будуть оновлюватись, базуючись на *WARP* втрату.

Виходячи з наведеного прикладу, можна сформувати алгоритм навчання моделі (рис. 2.12), використовуючи *WARP* втрату:

– Для позитивної пари (користувач, продукт) обираємо негативний приклад серед інших продуктів. Прораховуємо передбачення на парах і якщо не вийшло порушення порядку (тобто модель передбачила більшу оцінку для позитивної пари), продовжуємо порівнювати поки не станеться порушення.

– Якщо порушення отримали при першої спроби, то можна зробити великий градієнтний крок, так як це означає що модель дуже часто проставляє негативні приклади вище позитивних і потрібно сильно змінювати її ваги. Якщо пошук виконувався довго, то робимо маленький крок, так як модель вже достатньо гарно навчена.

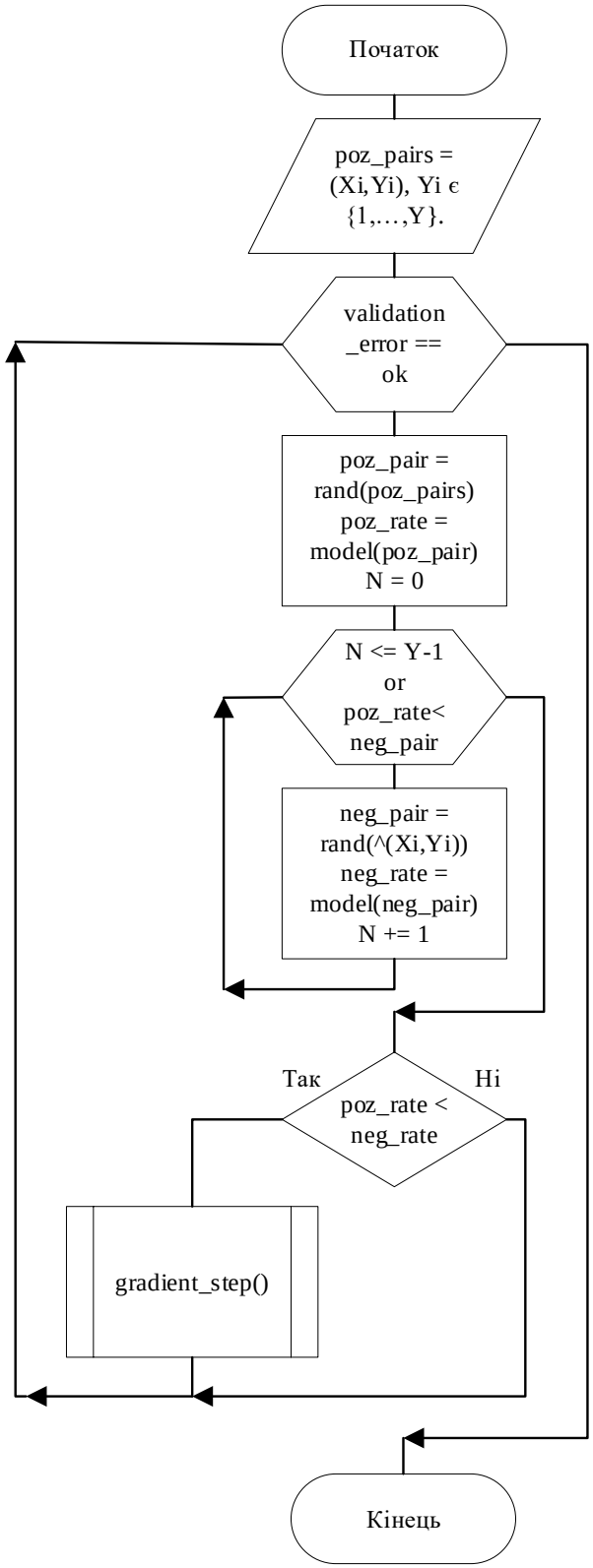

Рис. 2.12. Схема алгоритму навчання моделі з врахуванням функції *WARP*-втрати

2.4. Висновки до розділу

У другому розділі дипломного проекту проведено аналіз БД *AWS Athena*, таблиць та даних, які використовуються для генерації рекомендацій. Створено універсальну функцію, яка дістає дані з *Athena*, використовуючи передані об’єкти підключення та запиту.

Виокремлено три основних задачі, які можуть бути вирішені за допомогою фреймворків та готових інструментів. Було розглянуто та проаналізовано найпопулярніші оркестратори задач: *Luigi*, *Airflow*, *Kubeflow*. Та обрано *Airflow*, так як це найпопулярніший інструмент і його дуже активно підтримують розробники та він має дуже зручний інтерфейс і документацію. Проаналізовано бібліотеки для побудови рекомендацій: *Lenskit*, *Crab*, *Surprise*, *TensorRec*, *LightFM*, *Case Recommender*. Та обрано *LightFM*, через швидкість і легкість використання та підтримку *WARP* втрати. Також розглянуто фрейморки, за допомогою яких можна реалізувати асинхронний додаток: *Tornado*, *FastAPI*, *Sanic*. Та обрано *FastAPI* через швидкість та легкість використання.

Описано та розроблено алгоритм генерації рекомендацій, у якому буде використовуватись інструментарій бібліотеки *LightFM* – прорахування вбудувань на основі особливостей та підсумування їх для отримання кінцевого представлення користувачів та продуктів, представлення користувачів та продуктів у свою чергу виражаються у вигляді представлень їх особливостей: вбудовування прораховується для кожної особливості, після чого вони підсумовуються разом, щоб отримати кінцеві представлення користувачів та продуктів. Та описано алгоритм використання *WARP* втрати для оптимальної і швидкої генерації топ 10 рекомендацій.

РОЗДІЛ 3
ПРОГРАМНИЙ МОДУЛЬ ГЕНЕРАЦІЇ РЕКОМЕНДАЦІЙ ПОКУПЦЮ ПІД ЧАС ПРИДБАННЯ НЕРУХОМОСТІ

3.1. Структура програми генерації рекомендацій покупцю

На рис. 3.1 наведена структура програми генерації рекомендацій покупцю.

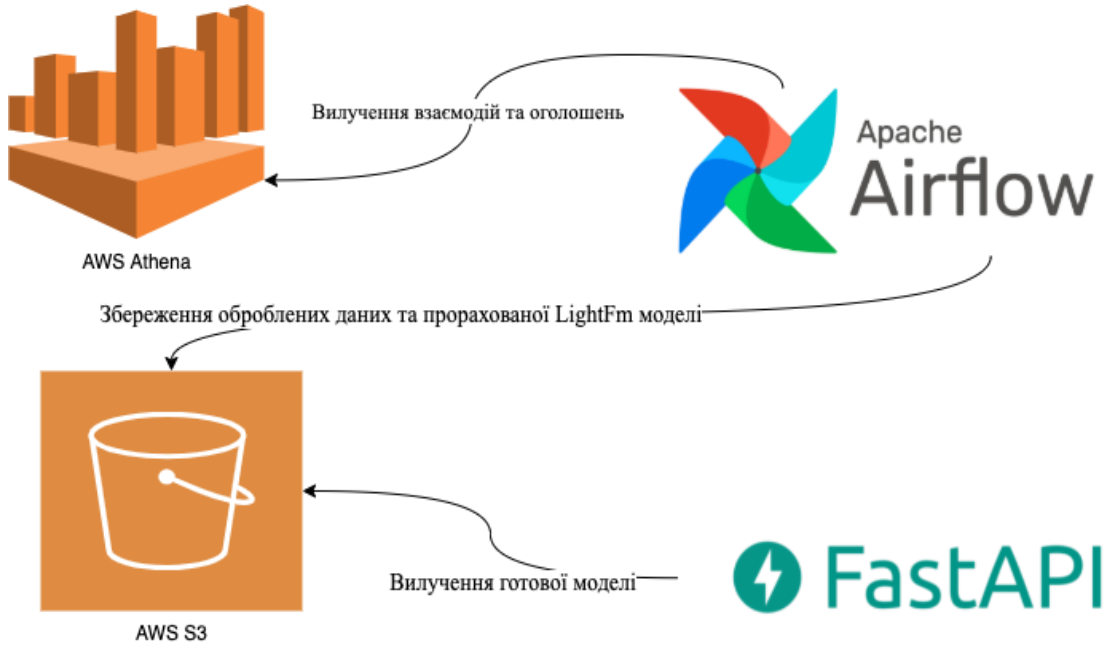

Рис. 3.1. Структура програми генерації рекомендацій покупцю

Загальна робота програми може бути описана чотирма компонентами: *Airflow, AWS Athena, AWS S3* та *FastAPI*. Оркестратор задач *Airflow* вилучає дані про взаємодії користувачів та оголошення з *Athena* бази даних, оброблює їх, обраховує *LightFM* модель та зберігає все у *S3*. *FastAPI* у свою чергу під час запуску вилучає оброблені дані та *LightFM* модель, після чого прослуховує

назначений порт для генерації рекомендацій для переданого користувача. Реалізація програми базується на мові програмування *Python*, бо наведені фреймворки та бібліотеки працюють саме на ньому.

Діаграма прецедентів всіх чотирьох компонент, на основі чого і працює програмний модуль, зображено на рис. 3.2.

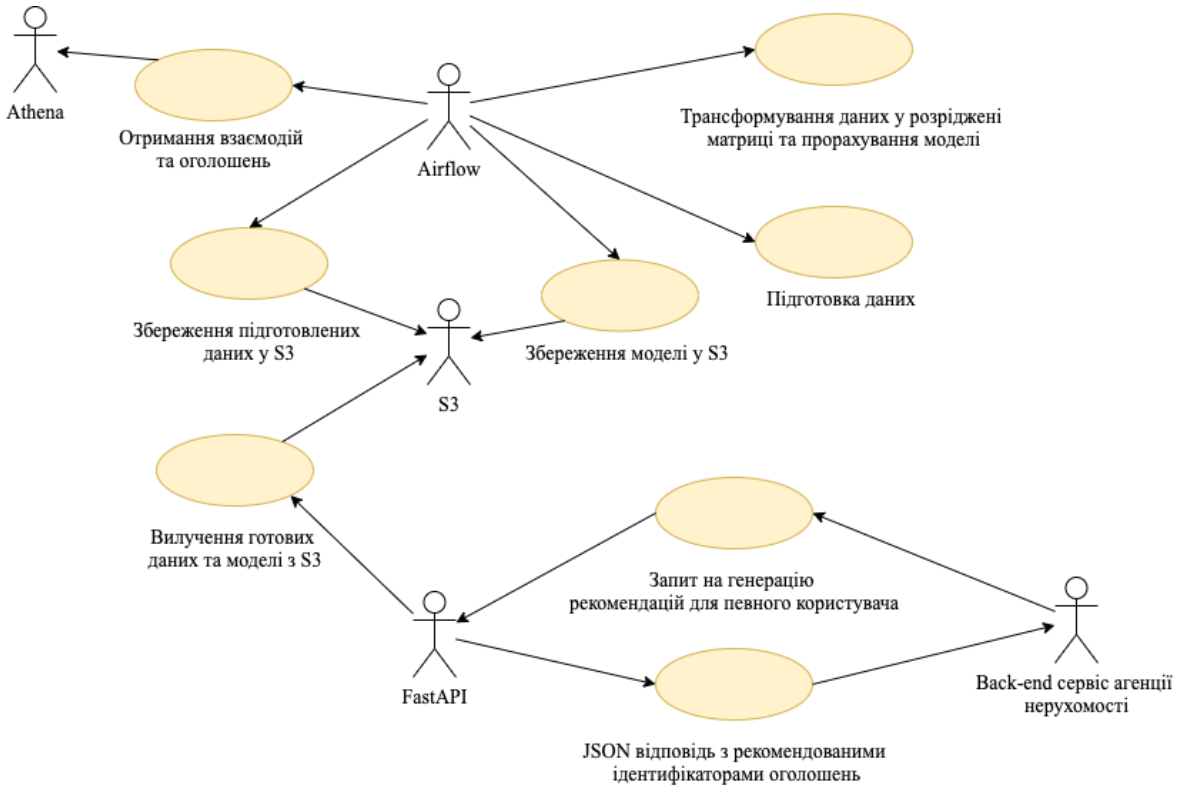

Рис. 3.2. *UML* – діаграма прецедентів для інтерпретації принципу роботи додатку

3.1.1. Підготовка та обробка даних у оркестраторі задач *Airflow*

Для створення задач та їх послідовності потрібно створити *DAG* (*Directed Acyclic Graph*) – спрямований асинхронний граф, який являється базовою одиницею оркестрування задачами та їх послідовністю. Для створення графу використовується наступний код:

```
dag = DAG(
    'Firefly',
    description='Data preparation and model calculation for recommendation system',
```

```
    schedule_interval='0 6 * * *',
    start_date=datetime(2021, 5, 20)
)
```

Створюється екземпляр класу *DAG*, та йому передаються аргументи для його ініціалізації: назва графу, опис графу, *CRON* вираз, який регулює частоту запуску графу, та дата початку створення графу.

Задачу прорахування рекомендаційної моделі можна розділити на 2 етапи: підготовка даних та прорахування самої моделі. Для цього створимо 2 об’єкти задач:

```
prepare_data = PythonOperator(
    task_id='prepare_data',
    execution_timeout=timedelta(minutes=30),
    python_callable=load_and_prepare_data,
    retries=0,
    dag=dag,
)
calculate_model = PythonOperator(
    task_id='calculate_model',
    execution_timeout=timedelta(minutes=20),
    python_callable=generate_model,
    retries=0,
    dag=dag,
)
```

Ці об’єкти створюються за допомогою класу *PythonOperator*, це означає, що ці задачі будуть виконуватись на мові *Python* у вигляді функцій, які можна викликати. У наведений клас, для його ініціалізації, передаються наступні аргументи: назва задачі, ліміт на виконання у хвилинах, функція, яку потрібно визвати під час виконання цієї задачі, кількість повторів, якщо задача завершилась помилкою, та об’єкт графу. Послідовність виконання задач описується операторами >> – задача виконується після закінчення попередньої та [ ] – задачі виконуються паралельно. В нашому випадку дві задачі *prepare_data* та

calculate_model повинні виконуватись послідовно (рис. 3.3), тоді запишемо їх виконання так:

prepare_data >> calculate_model

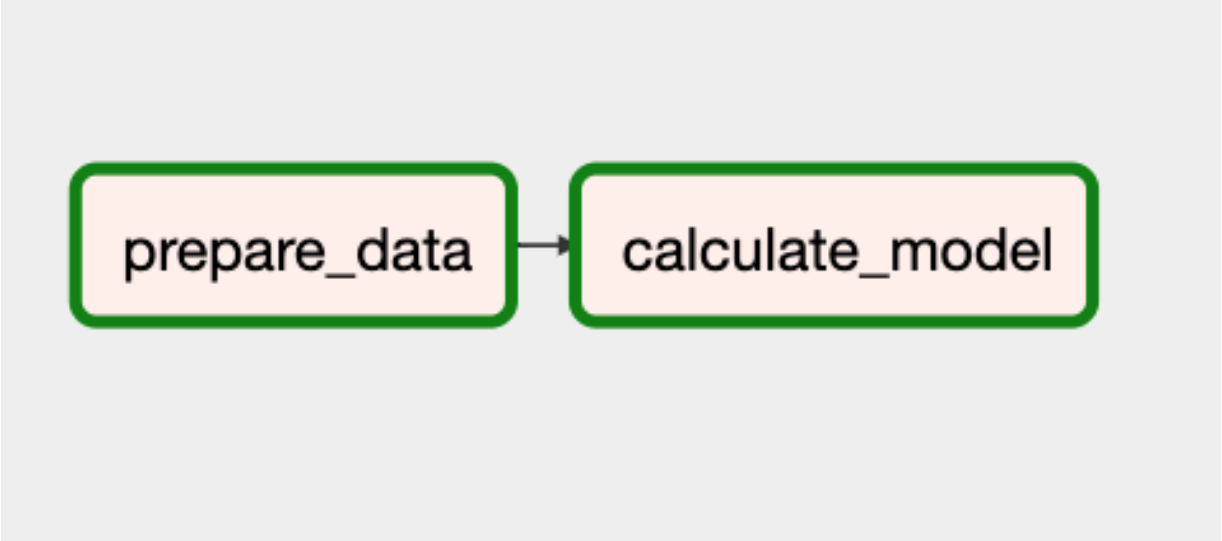

Рис. 3.3. Графічне представлення графу, у інтерфейсі *Airflow*

Функція *prepare_data.*

Щоб використовувати дані для генерації рекомендацій, їх потрібно обробити та правильно підготувати. Нехай вилучені дані з пункту 2.1 збережені у змінних *df_items* та *df_interactions*, оголошення та взаємодії користувача відповідно, які являються таблицями, представлені об'єктом типу *pandas.DataFrame*, то підготовка взаємодій буде виглядати наступним чином (функція *prepare_data* у Додатку А). Наведений код, для зручності, переводить стовпчик *timestamp* з рядкового типу даних у об'єкт *datetime*, за допомогою методу *to_datetime*(). Після цього відфільтровує рядки взаємодій, щоб залишити тільки записи, де *property_value* дорівнює *propertyId*, це можна зробити передавши об'єкту df_*interactions* у квадратних дужках логічний вираз *df_interactions*[‘*property_name*’]==*’propertyId’*. Видаляємо непотрібні стопвчики *property_name*, *log_uuid*, *event_name*, використовуючи метод *drop*(). Перейменуємо стовпчики *cid*, *property_value* на *user_id*, *item_id* відповідно і додаємо ще один стовпчик *score* та присвоюємо йому значення 1. Стовпчик *score* буде відповідати за пріоритетність взаємодії, де 1 це звичайний пріоритет, *a* >1 більш пріоритетний. Для цього потрібно знайти дублікати взаємодій між користувачем та оголошенням, методом *duplicated*(), який приймає на вхід список

з назв стовпчиків відносно яких потрібно перевірити дублікати, в нашому випадку це список з двох змінних [‘*user_id*’, ‘*item_id*’]. Видаляємо всі взаємодії з головного об’єкту *df_interactions*, щоб в майбутньому повернути їх з підвищеним значенням *score*, передавши йому у квадратних дужках знак ~ (інвертує значення у списку логічних змінних) та список логічних змінних, який був результатом методу *duplicated*(). Для підвищення пріоритетності взаємодій, які повторюються, нова пріоритетність прораховується наступним чином:

$$score = log_{10}\left(\sum_{j \in P} old_{score\,j} * 10\right),$$

де *old_score* – попередній пріоритет взаємодії, який дорівнює одиниці;

P – множина дублікатів взаємодій.

З рис. 3.4 видно, що завдяки логарифму сума пріоритетів дублікатів помножених на 10 буде нормалізоване, тобто якщо користувач передивлявся одне і те ж оголошення більше одного разу, то кожний додатковий перегляд буде менше впливати на *score*, що не призведе до надлишкового значення пріоритету.

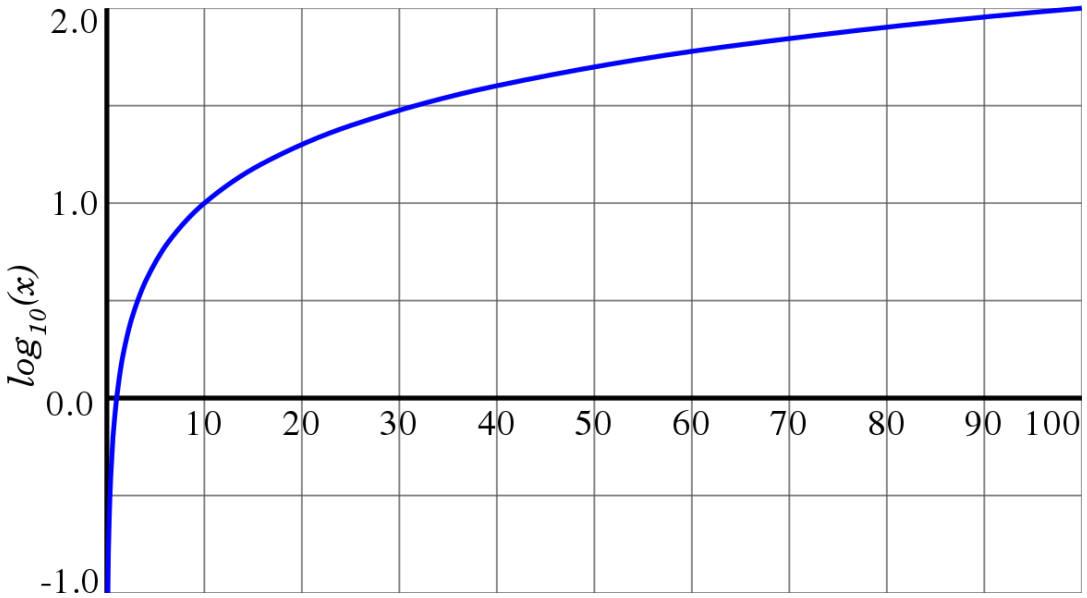

Рис. 3.4. Графік функції десяткового логарифму

Щоб прорахувати нове значення пріоритету, використовуємо метод *log*10(), бібліотеки *numpy*, передавши йому агреговану суму стовпчика *score* помножену на 10. Коли нове значення прораховане його можна додати у головний об’єкт

df_interactions, методом *append*(). Об’єкт взаємодій готовий для роботи з *LightFM*, тому його потрібно зберегти на *S3*. Для того щоб зберегти об’єкти у файл існує зручна бібліотека *pickle*, яка конвертує змінні у файл. Створюємо двійковий потік та зберігаємо в нього байти файлу, який потрібно зберегти у *S3*, після цього викликаємо функцію *upload_file_s3*(), яка зберігає файли у *S3* та приймає на вхід назву файлу, двійковий потік, назву папки у *S3* та логічне значення яке регулює чи відкритий переданий двійковий потік. Функція *upload_file_s3*() виглядає наступним чином:

```
def upload_file_s3(file_name, file_obj, bucket_name=REPORTS_BUCKET_S3, encoded=False):
    if not encoded:
        file = json.dumps(file_obj).encode('utf-8')
        file_buffer = io.BytesIO(file)
    else:  # buffer was sent directly
        file_buffer = file_obj
    log.info(f"Uploading file '{file_name}' to S3 bucket '{bucket_name}'")
    try:
        s3_client.put_object(
            ACL='bucket-owner-full-control',
            Body=file_buffer,
            Key=file_name,
            Bucket=bucket_name
        )
    except ClientError as e:
        log.error(f"Failed to upload file '{file_name}' to S3 bucket '{bucket_name}': {e}")
        return None
```

Дана функція кодує та відкриває двійковий потік якщо значення переданого аргументу *encoded* є хибним. Після цього викликається метод *put_object*(), офіційної бібліотеки Амазону, який передає у *S3* файл, його назву, папку та налаштовує доступ до нього.

Підготовка оголошень виглядає наступним чином:

```
df_items.replace('', np.nan, inplace=True)
df_items.rename(columns={"uuid": "id"}, inplace=True)
pickle_buffer = io.BytesIO()
df_items.to_pickle(pickle_buffer)
pickle_buffer.seek(0)
upload_file_s3(
   "firefly/items_preprocessed.pickle",
   pickle_buffer.getvalue(),
   bucket_name=DATA_BUCKET_S3,
   encoded=True
)
```

Замінюються всі пусті поля на об’єкт *np.nan*, методом *replace*(), для подальшої зручності в роботі з даними. З такою ж цілю перейменуються стовпчик *uuid* на *id*, методом *rename*(). Після чого створюється двійковий потік та передається, вище описаній, функції *upload_file_s3*() для зберігання оброблених даних у *S3*.

Функція *calculate_model.*

Оброблені дані потрібно трансформувати у розріджені матриці з типом даних *float*, та виконати прорахунки з пункту 2.3.1 та пункту 2.3.2. Для цього у бібліотеці *LightFM* наявний класи *Dataset* і клаас *LightFM*, перший допомагає трансформувати дані, а другий прорахувати рекомендації.

Трансформуємо дані (функція *calculate_model* у Додатку А). Створюється екземпляр класу *Dataset* та викликається метод *fit*(), який ініціалізує набір даних та приймає на вхід унікальні ідентифікатори користувачів та оголошень, *df_interactions*[‘*user_id’*].*unique*() та *df_interactions*[‘*item_id*’].*unique*() відповідно. Потім виконується підготовка ознак оголошень, для цього потрібно передати унікальні значення кожної ознаки у *Dataset*. Тобто обираються унікальні значення ознаки *action_type*, *price*, *property_type*, *number_of_bedrroms*, *stc*, *new_build*, *green_index*, *grocery_index*, *education_index*, *quiet_index*, *transport_index* та *safety_index* методом *unique*(), додатково перед цим замінюються нулями пусті

значення, методом *fillna*(), бо клас *Dataset* не оброблює змінні типу *np.nan* або *None*. Коли всі унікальні значення ознак отримані, їх потрібно помістити у один список, це робиться завдяки ітерації через кожну ознаку та додаванням її унікальних значень у загальний список, методом *append*(). Потім ці ознаки передаються у метод *fit_partial*() класу *Dataset*, який оновлює набір даних цими ознаками. Так як *LightFM* у середині *Dataset* індексує користувачів і оголошення своїми ідентифікаторами типу *Int*, то потрібно створити мапінг, у якому ключ зіставляється якомусь значенню, за допомогою методу *mapping*() класу *Dataset*, завдяки якому можна буде дізнатись який ідентифікатор має користувач або оголошення у *LightFM*. Для зручності створюється зворотній мапінг, за допомогою строкового виразу, в якому виконується ітерація через оригінальний мапінг та ключ і значення міняються місцями. Отримуємо розріджену матрицю взаємодій та пріоритетів взаємодій, за допомогою методу *build_interactions*() класу *Dataset*, який на вхід приймає список кортежів ідентифікатора користувача, ідентифікатора оголошення та пріоритету їх взаємодії. Отримуємо розріджену матрицю ознак оголошень, за допомогою методу *build_item_features*() класу *Dataset*, який на вхід приймає список ідентифікаторів оголошень та зіставлених до них кортеж значень ознак.

Створюємо *LightFM* модель та навчаємо її:

```
lfm_model = LightFM(loss='warp')
lfm_model.fit_partial(
    train_mat,
    sample_weight=train_mat_weights,
    item_features=train_items_features,
    num_threads=2,
    epochs=21,
)
```

Наведений код створює об’єкт класу *LightFM*, та ініціалізує його, використовуючи аргумент *loss*=’*warp*’, це означає що всі прорахунки будуть коригуватись функцією втрат з пункту 2.3.2. Після цього викликаємо метод *fit_patial*(), який виконає прорахунки з пункту 2.3.1 та пункту 2.3.2, він приймає на

вхід розріджену матрицю взаємодій, матрицю пріоритетів, матрицю ознак оголошень, кількість паралельних процесів для прорахунків та кількість епох – кількість ітерацій на одному й тому ж вхідному набору даних.

Коли модель прорахована, її та допоміжні об’єкти потрібно зберегти у *S3* для подальшого використання. Це робиться наступним кодом:

```
pickle_buffer = io.BytesIO()
pickle.dump(lfm_model, pickle_buffer, protocol=pickle.HIGHEST_PROTOCOL)
pickle_buffer.seek(0)
upload_file_s3(
    "firefly/lightfm_model.pickle",
    pickle_buffer.getvalue(),
    bucket_name=DATA_BUCKET_S3,
    encoded=True
)

upload_file_s3('firefly/lightfm_mapping.json', lightfm_mapping, bucket_name=DATA_BUCKET_S3)

pickle_buffer = io.BytesIO()
sp.save_npz(pickle_buffer, train_items_features)
pickle_buffer.seek(0)
upload_file_s3(
    "firefly/train_items_features.npz",
    pickle_buffer.getvalue(),
    bucket_name=DATA_BUCKET_S3,
    encoded=True
)
```

Створюються двійкові потоки для кожного об’єкту та передаються, вище описаній, функції *upload_file_s3*() для зберігання оброблених даних у *S3*.

3.1.2. Вилучення готових даних та запуск додатку *FastAPI*

Для вилучення готових даних та прорахуванню рекомендацій, створюємо клас:

```
class Model:
    def __init__(self):
        self.lightfm_mapping = json.loads(load_file_from_s3('firefly/lightfm_mapping.json').decode('utf-8'))
        self.train_items_features = sp.load_npz(io.BytesIO(load_file_from_s3("firefly/train_items_features.npz")))
        self.lfm_model = pickle.load(io.BytesIO(load_file_from_s3('firefly/lightfm_model.pickle')))
        self.interactions = pickle.load(io.BytesIO(load_file_from_s3('firefly/interactions_preprocessed.pickle')))
        self.df_items = pickle.load(io.BytesIO(load_file_from_s3('firefly/items_preprocessed.pickle')))
        self.items_mapping = pd.DataFrame(self.lightfm_mapping['items_inv_mapping'].values(), columns=['id'])
```

Клас *Model* при ініціалізації створює 6 атрибутів, за допомогою яких буде виконуватись генерація рекомендацій. Ці атрибути скачуються з *S3*, за допомогою функції *load_file_from_s3*(), та перетворюються у об’єкти методом *load()* бібліотеки *pickle*. Функція *load_file_from_s3()* виглядає наступним чином:

```
def load_file_from_s3(file_name, bucket_name=DATA_BUCKET_S3):
    log.info(f"Opening file '{file_name}' from S3 bucket '{bucket_name}'")
    try:
        fileobj = s3_client.get_object(
            Bucket=bucket_name,
            Key=file_name
        )
    except ClientError as e:
```

```
            log.error(f"Failed to open file '{file_name}' from S3 bucket '{bucket_name}': {e}")
            return None
        else:
            return fileobj['Body'].read()
```

Вона приймає на вхід назву файлу та папки звідки потрібно його дістати. Викликає метод *get_object*(), офіційної бібліотеки Амазону, який повертає двійковий потік вказаного файлу. Після цього функція повертає двійкове значення файлу, методом *read*().

Додаємо у створений клас новий метод *predict*(), який буде повертати топ 10 рекомендованих оголошень для вказаного користувача:

```
    def predict(self, user_id):
        row_id = self.lightfm_mapping['users_mapping'].get(user_id)
        if row_id:
            log.info(f'Recommendations for user {user_id}, row number - {row_id}')
            items_to_rec = list(self.items_mapping[~self.items_mapping['id'].isin(
                list(self.interactions[self.interactions['user_id'] == user_id]['item_id'])
            )].index)   # exclude already viewed items
            viewed_items = self.df_items[self.df_items['id'].isin(
                list(self.interactions[self.interactions['user_id'] == user_id]['item_id'])
            )]

            prediction = self.lfm_model.predict(
                row_id,
                items_to_rec,
                item_features=self.train_items_features,
                num_threads=4
            )
```

```
            top_cols = heapq.nlargest(TOP_N, range(len(prediction)), prediction.take)
            # get top 10 scores and their indexes

            log.info(f'Max score of prediction: {np.max(prediction)}')

            recs = pd.DataFrame({'col_id': top_cols}, dtype=str)
            recs['item_id'] = recs['col_id'].map(self.lightfm_mapping['items_inv_mapping'].get)
            predicted_items = self.df_items[self.df_items['id'].isin(
                list(recs['item_id'])
            )]
            inv_map = {v: int(k) for k, v in recs['item_id'].to_dict().items()}
            predicted_items.sort_values(by='id', key=lambda x: x.map(inv_map), inplace=True)
            return viewed_items, predicted_items, recs['item_id']
        return None
```

Цей метод приймає на вхід ідентифікатор користувача, перевіряє чи існує цей користувач у *LightFM*, якщо ні, то повертаємо пусте значення. Якщо користувач існує, то для створення рекомендацій, фільтруємо вже переглянуті оголошення, щоб не показувати їх користувачу. Викликаємо метод *predict*() для генерації рекомендацій, та передаємо йому ідентифікатор користувача, список оголошень, з яких потрібно згенерувати рекомендацію, розріджену матрицю ознак оголошень та кількість паралельних потоків для обчислень. Повернений об’єкт являється списком оцінок для пар користувач та оголошення, але нам потрібні ідентифікатори десяти оголошень з найбільшою оцінкою, це робиться за допомогою методу *nlargest*() бібліотеки *heapq*. Цей метод приймає на вхід кількість оголошень які потрібно дістати, ітератор списку рекомендацій та сам список. Після цього створюємо таблицю рекомендованих оголошень з їх ознаками, за допомогою бібліотеки *pandas*. Результатом методу *predict*() являється 3 таблиці:

переглянуті оголошення, рекомендовані оголошення з ознаками та рекомендовані оголошення без ознак.

Для створення додатку *FastAPI* запишемо наступний код:

```
api = FastAPI()
@api.on_event("startup")
def startup_event():
    log.info('Start Firefly-API')
    api.state.model = Model()
```

Наведений код ініціалізує додаток, створенням об’єкта класу *FastAPI*. Та виклик декоратора *on_event*(“*startup*”), який запускає функцію *startup_event*() під час запуску додатку. Функція *startup_event*(), у свою чергу ініціалізує об’єкт класу *Model*() та додає його у стан додатку, тобто цей об’єкт буде доступний з будь-якого місця додатку.

Створимо демонстраційну сторінку, яка буде показувати рекомендації:

```
@api.get("/predict/{user_id}", response_class=HTMLResponse)
def predict(user_id):
    viewed_items, predicted_items, recs = api.state.model.predict(user_id)
    if recs is None:
        raise HTTPException(status_code=404, detail=f"User {user_id} absent in model")
    else:
        return f"""<html>
            <h3> Viewed Listings </h3>
            <div> {viewed_items.to_html()} </div>
            <h3> Recommended Listings </h3>
            <div> {predicted_items.to_html()} </div>
            <h3> Json Response </h3>
            <div> {recs.to_json()} </div>
        </html>"""
```

Створення сторінки виконується декоратором *.get*(), який приймає на вхід шлях до сторінки та тип відповіді на запит. Для повернення рекомендацій на

сторінці, викликається метод *predict*(), класу *Model*, з стану додатка. Після чого виконується перевірка на пусті значення повернених об’єктів, якщо вони не пусті, то повертаємо *HTML* відповідь.

3.2. Демонстрація та тестування роботи програми генерації рекомендацій покупцю під час придбання нерухомості

Для запуску додатку потрібно ввести у командний рядок наступну команду:
uvicorn app.main:api
Після чого виведеться інформація про вдалий запуск додатку (рис. 3.5).

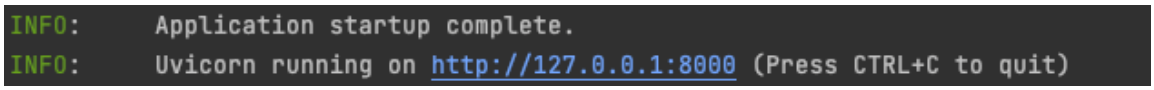

Рис. 3.5. Інформація про запуск додатку *FastAPI*

Якщо перейти на вказану адресу з пункту 3.1.2, наприклад *http://127.0.0.1:8000/predict/2983c666-fc16-4140-b276-328cac7c34ee*, то можна отримати відповідь від серверу з рекомендованими оголошеннями (рис. 3.6).

Viewed Listings

	id	action_type	address_postal_code	price	property_type	number_of_bedrooms	stc	new_build	quiet_index	green_
65032	A5A672FC46FD51_10836102	SALE	LS9 0EN	160000	HOUSE_DETACHED	3.0	true	false	10.0	3.0
117480	F4CD171AEEAB99_NTLLL99293697	SALE	LS8 3BX	190000	HOUSE_SEMI_DETACHED	3.0	true	false	10.0	5.0
373963	A53304C33BA608_PUDSP212799	SALE	LS13 4DS	169950	HOUSE_TERRACED	3.0	true	false	4.0	8.0

Recommended Listings

	id	action_type	address_postal_code	price	property_type	number_of_bedrooms	stc	new_build	quiet_index	green_
14749	VE_29116508	SALE	IP29 4RQ	400000	HOUSE_OTHER	3.0	true	false	3.0	9.0
317134	VE_30419163	SALE	BA14 7PU	225000	HOUSE_SEMI_DETACHED	3.0	true	false	10.0	6.0
15197	VE_30198564	SALE	NE23 3FS	339950	HOUSE_BUNGALOW	3.0	true	false	10.0	8.0
388541	A5F6B1DB87EFA8_100769065641	SALE	WF7 5PW	150000	HOUSE_SEMI_DETACHED	3.0	true	false	10.0	7.0
448387	F581284C9067F7_RS0022	SALE	RM13 9UU	345000	HOUSE_SEMI_DETACHED	3.0	true	false	2.0	7.0
213099	VE_26968710	SALE	TN33 0EU	280000	HOUSE_END_OF_TERRACE	3.0	true	false	1.0	10.0
278678	F476A8F77379F9_490538	SALE	TW11 8DH	1295000	HOUSE_SEMI_DETACHED	4.0	true	false	10.0	8.0
31067	13343_1479	RENT	W5 2PJ	1690	HOUSE_TERRACED	2.0	false	false	1.0	4.0
155023	A5EC7AFF225C51_SOU180070	SALE	EX2 7FX	340000	HOUSE_SEMI_DETACHED	3.0	true	false	4.0	4.0
142630	BABI0_3146	SALE	OX39 4EF	345000	HOUSE_SEMI_DETACHED	2.0	true	false	10.0	9.0

Json Response

{"0":"VE_29116508","1":"VE_30419163","2":"VE_30198564","3":"A5F6B1DB87EFA8_100769065641","4":"F581284C9067F7_RS0022","5":"VE_26968710","6":"F476A8F77379F

Рис. 3.6. Відповідь додатку за адресою
http://127.0.0.1:8000/predict/2983c666-fc16-4140-b276-328cac7c34ee

Для перевірки релевантності рекомендацій можна обрати 5 випадкових користувачів та порівняти рекомендовані оголошення з потребами користувача, виходячи з переглянутих оголошень (табл. 3.1).

Таблиця 3.1

Релевантність рекомендацій

№	Потреби користувача	Кількість переглянутих оголошень	Кількість релевантних оголошень	Кількість рекомендованих оголошень
1	Однокімнатний дім з приблизною ціною 100.000, з гарним показником навколишньої безпеки та озеленіння	1	5	10
2	Чотирикімнатний дім з приблизною ціною 240.000, з гарним показником безпеки, озеленіння та звукового забруднення	3	8	10
3	Двокімнатний дім з ціною 240.000, гарним показником звукового забруднення та середніми озелененням	1	5	10
4	Чотирикімнатний дім з приблизною ціною 490.000, з гарним показником	3	8	10

Закінчення таблиці 3.1

№	Потреби користувача	Кількість переглянутих оголошень	Кількість релевантних оголошень	Кількість рекомендованих оголошень
	навколишнього озеленіння та біля магазинів			
5	Трикімнатний дім з приблизною ціною 175.000, з гарним показником звукового забруднення та біля магазинів	1	5	10

З наведеної таблиці можна помітити, що кількість релевантних рекомендацій більша для тих користувачів, які взаємодіяли з більшою кількістю оголошень.

Прорахувати якість рекомендацій, якщо кількість рекомендованих оголошень дорівнює десяти, можна наступною формулою:

$$p = \frac{\sum_{j \in R} r_j}{n},$$

де R – множина перевірених користувачів;

r – кількість релевантних оголошень для користувача;

n – кількість перевірених користувачів;

p належить відрізку [0,10], де 10 це дуже точні рекомендації, а 0 - неточні. В нашому випадку p=6.2, це означає, що рекомендаційна система генерує достатньо гарні рекомендації.

Для тестування швидкодії програми можна написати наступний код:

```
counter = 50
time = 0
while counter > 0:
```

```
    response = requests.get('http://127.0.0.1:8000/predict/fff5af02-20bd-4d93-a070-6c315c88c580')
    time += response.elapsed.total_seconds()
    counter -= 1
print(time/50)
```

Наведений код відправляє запит додатку 50 разів підряд та сумує час відповіді програми на кожний запит. Час, за який відповів додаток, можна дістати за допомогою методу *total_seconds*(). Після завершення циклу, виводиться повідомлення про середній час відповіді додатку (рис. 3.7).

```
0.7883136999999999

Process finished with exit code 0
```

Рис. 3.7. Повідомлення про середній час відповіді додатку

Якщо середній час відповіді додатку менший або дорівнює 1 секунди, то можна вважати що додаток достатньо швидко генерує рекомендації та відповідає на запити.

3.3. Висновки до розділу

У третьому розділі дипломного проекту було розглянуто структуру програми: *AWS Athena*, *Airflow*, *S3* та *FastAPI*. Розроблено послідовність задач на *Airflow*, яка складається з двох задач: підготовки, обробки даних та прорахування моделі. Описано функції які представляють собою задачі у *Airflow*. Розроблено власний клас, який допомагає вилучати дані з *S3* та прораховувати рекомендації. Створено додаток, використовуючи *FastAPI*, який повертає рекомендації для вказаного користувача та відповідає на запити. Протестовано релевантність рекомендацій, використовуючи набір з п’яти користувачів, та перевірено швидкодію додатку, використовуючи розроблену програму для тестування.

ВИСНОВКИ

У дипломній роботі було розроблено програму генерації рекомендацій покупцю під час придбання нерухомості.

У першому розділі було розглянуто цілі та задачі, які вирішують рекомендаційні системи. Наведено класифікацію рекомендаційних систем: залежно від типу рекомендацій та принципу їх формування розрізняють системи колаборативної фільтрації, системи, що базуються на контенті, що базуються на знаннях та гібридні рекомендаційні системи. Було розглянуто популярні сайти різної тематики, у яких присутні рекомендаційні системи і здійснено аналіз їх переваг та недоліків. Для порівняння було обрано маркетплейси *rozetka.com.ua*, *amazon.com*, медіасервіси *youtube.com*, *megogo.net* та сайти пошуку нерухомості *zoopla.co.uk, onedome.com*. Було поставлено задачу: розробити додаток, що буде прораховувати рекомендації раз у день, базуючись на історії пошуку користувача та додаткові атрибути переглянутих оголошень.

У другому розділі дипломного проекту проведено аналіз існуючих таблиць та даних, які використовуються для генерації рекомендацій. Було створено універсальну функцію, яка вилучає дані з БД *AWS Athena*. Виокремлено три основних задачі, які можуть бути вирішені за допомогою фреймворків та готових інструментів. Розглянуто, проаналізовано та виконано порівняння найпопулярніших оркестраторів задач: *Luigi, Airflow, Kubeflow*, та обрано *Airflow*, через його переваги відносно інших. Проаналізовано бібліотеки для побудови рекомендацій: *Lenskit, Crab, Surprise, TensorRec, LightFM, Case Recommender*, та обрано *LightFM*, через велику швидкість, легкість використання і підтримку *WARP* втрати. Також розглянуто фреймворки, за допомогою яких можна реалізувати асинхронний додаток: *Tornado, FastAPI, Sanic*, та обрано *FastAPI* через швидкість та легкість використання. Описано та розроблено алгоритм генерації рекомендацій, у якому використовується інструментарій *LightFM*, та описано алгоритм *WARP* втрати, яка оптимізує результат рекомендацій.

У третьому розділі дипломного проекту було розглянуто структуру програми, а саме структуру взаємодій її компонент: *AWS Athena*, *Airflow, S3* та

FastAPI. Розроблено послідовність задач на *Airflow*, яка складається з двох задач: підготовки, обробки даних та прорахування моделі. Створено додаток, використовуючи *FastAPI*, який повертає рекомендації для вказаного користувача та відповідає на запити. Було протестовано релевантність рекомендацій, використовуючи набір з п’яти користувачів, та перевірено швидкодію додатку, використовуючи розроблену програму для тестування.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Scaling Up To Large Vocabulary Image Annotation* [Електронний ресурс*]*. – Режим доступу*: http://www.thespermwhale.com/jaseweston/papers/wsabie-ijcai.pdf*

2. *LightFM* documentation [Електронний ресурс]. – Режим доступу: *https://making.lyst.com/lightfm/docs/lightfm.html*

3. *FastAPI documentation* [Електронний ресурс]. – Режим доступу*: https://fastapi.tiangolo.com/*

4. *Airflow documentation* [Електронний ресурс]. – Режим доступу*: https://airflow.apache.org/docs/*

5. *Metadata Embeddings for user and item cold-start recommendations* [Електронний ресурс]. – Режим доступу*: http://ceur-ws.org/Vol-1448/paper4.pdf*

6. Володимир Гайджарж, Ігор Ізварін Бази даних в інформаційних системах – К.: Університет «Україна», 2010. – 415 с.

7. Казаков Д. І. Використання гібридних підходів у розробці рекомендацій – К.: Санкт-Петербурзький державний університет, 2019. – 33 с.

8. *LightFM Hybrid Recommendation system* [Електронний ресурс]. – Режим доступу: *https://www.kaggle.com/niyamatalmass/lightfm-hybrid-recommendation-system*

9. *“A Byte of Python”* [Електронний ресурс]. – Режим доступу: *https://wombat.org.ua/AByteOfPython/AByteofPythonRussian-2.02.pdf*

10. ДСТУ 3008-95 “Документація Звіти у сфері науки і техніки. Структура і правила оформлення”.

11. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.

12. ГОСТ 2.106-96 ЕСКД “Текстовые документы”.

13. *Richard Stevens W. UNIX Network Programming*. – *Prentice Hall,* 1990. – 784 с.

14. *Sean W. Linux Socket Programming. – Sams,* 2001. – 400 с.

15. *Ken Coar, Rich B. Apache Cooknook. – O’Reilly Media*, 2008. – 310 с.

ДОДАТОК А

ОПИС ФУНКЦІЙ У *AIRFLOW*

Функція *prepare_data*

```
df_interactions['timestamp'] = pd.to_datetime(df_interactions['timestamp'])
df_interactions = df_interactions[df_interactions['property_name'] == 'propertyId'].reset_index(drop=True)
df_interactions.drop(['property_name', 'log_uuid', 'event_name'], axis='columns', inplace=True)
df_interactions.rename(columns={"cid": "user_id", "property_value": "item_id"}, inplace=True)
df_interactions['score'] = 1
df_interactions['item_id'] = df_interactions['item_id'].astype(str) # some ids could be detected as int
duplicates = df_interactions.duplicated(subset=['user_id', 'item_id'], keep=False)
df_duplicates = df_interactions[duplicates].sort_values(by=['user_id', 'timestamp'])
df_interactions = df_interactions[~duplicates]
df_duplicates = df_duplicates.groupby(['user_id', 'item_id']).agg({'timestamp': 'max', 'score': 'sum'})
df_duplicates['score'] = np.log10(df_duplicates['score']*10)

df_interactions = df_interactions.append(df_duplicates.reset_index(), ignore_index=True, verify_integrity=True)
pickle_buffer = io.BytesIO()
df_interactions.to_pickle(pickle_buffer)
pickle_buffer.seek(0)
upload_file_s3(
    "firefly/interactions_preprocessed.pickle",
    pickle_buffer.getvalue(),
```

```
            bucket_name=DATA_BUCKET_S3,
            encoded=True
        )
```

Функція *calculate_model*

```
        dataset = Dataset()
        dataset.fit(df_interactions['user_id'].unique(),
df_interactions['item_id'].unique())
        num_users, num_items = dataset.interactions_shape()
        action_type_features = df_items['action_type'].unique()

        df_items['price'] = df_items['price'].fillna(0)
        price_features = df_items['price'].unique()

        property_type_features = df_items['property_type'].unique()

        df_items['number_of_bedrooms'] = df_items['number_of_bedrooms'].fillna(0)
        number_of_bedrooms_features = df_items['number_of_bedrooms'].unique()

        stc_features = df_items['stc'].unique()

        new_build_features = df_items['new_build'].unique()

        df_items['green_index'] = df_items['green_index'].fillna(0)
        green_index_features = df_items['green_index'].unique()

        df_items['grocery_index'] = df_items['grocery_index'].fillna(0)
        grocery_index_features = df_items['grocery_index'].unique()

        df_items['education_index'] = df_items['education_index'].fillna(0)
        education_index_features = df_items['education_index'].unique()
```

```
df_items['quiet_index'] = df_items['quiet_index'].fillna(0)
quiet_index_features = df_items['quiet_index'].unique()

df_items['transport_index'] = df_items['transport_index'].fillna(0)
transport_index_features = df_items['transport_index'].unique()

df_items['safety_index'] = df_items['safety_index'].fillna(0)
safety_index_features = df_items['safety_index'].unique()

item_features = []
for feature in (
    action_type_features, price_features, property_type_features,
number_of_bedrooms_features, stc_features,
    new_build_features, green_index_features, grocery_index_features,
education_index_features,
    quiet_index_features, transport_index_features, safety_index_features):
  item_features = np.append(item_features, feature)

dataset.fit_partial(
  users=(x for x in df_users['user_id']),
  items=(x for x in df_items['id']),
  item_features=item_features
)

num_users, num_items = dataset.interactions_shape()
lightfm_mapping = dataset.mapping()
lightfm_mapping = {
  'users_mapping': lightfm_mapping[0],
  'user_features_mapping': lightfm_mapping[1],
  'items_mapping': lightfm_mapping[2],
```

```
        'item_features_mapping': lightfm_mapping[3],
    }

    lightfm_mapping['users_inv_mapping'] = {v: k for k, v in lightfm_mapping['users_mapping'].items()}
    lightfm_mapping['items_inv_mapping'] = {v: k for k, v in lightfm_mapping['items_mapping'].items()}

    num_user_features = dataset.user_features_shape()
    num_show_features = dataset.item_features_shape()
    log.info('Num user features: {} -> {}  Num item features: {} -> {}.'.format(
        num_user_features[1] - num_users, num_user_features[1],
        num_show_features[1] - num_items, num_show_features[1]
    ))

    train_mat, train_mat_weights = dataset.build_interactions(
        df_to_tuple_iterator(df_interactions[['user_id', 'item_id', 'score']])
    )

    train_items_features = dataset.build_item_features(
        ((row['id'], [
            row['action_type'],
            row['price'],
            row['property_type'],
            row['number_of_bedrooms'],
            row['stc'],
            row['new_build'],
            row['green_index'],
            row['grocery_index'],
            row['education_index'],
            row['quiet_index'],
```

```
        row['transport_index'],
        row['safety_index']
    ]) for index, row in df_items.iterrows())
)
```