

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістра

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ДЛЯ ВОЛОНТЕРСЬКОГО
КООРДИНАТОРА

RESEARCH AND SOFTWARE IMPLEMENTATION OF A
RECOMMENDATION SYSTEM FOR A VOLUNTEER COORDINATOR

Виконав: студент(ка) 2 курсу, групи КНТ-121м

Спеціальності 121 Інженерія

програмного забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Дьячков В.В.

(прізвище та ініціали)

Керівник Федорончак Т.В.

(прізвище та ініціали)

Рецензент Голуб Т.В.

(прізвище та ініціали)

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

Інститут, факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти магістр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

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

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Дьячкова Віталія Володимировича
 (прізвище, ім'я, по батькові)

- Тема проєкту (роботи) Дослідження та програмна реалізація рекомендаційної системи для волонтерського координатора. Research and Software Implementation of a Recommendation System for a Volunteer Coordinator
- керівник проєкту (роботи) Федорончак Тетяна Василівна, к.т.н., доцент,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
 затверджені наказом закладу вищої освіти від “04” листопада 2022 року №363
- Строк подання студентом проєкту (роботи) 1 грудня 2022 року
- Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
- Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми 3. Опис програми 4. Тестування і результати 5. Керівництво програміста 6. Керівництво системного програміста
- Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-6 Основна частина	Федорончак Т.В., доцент		
Нормоконтролер	Липовець М.В., асистент		

7. Дата видачі завдання “ 1 ” вересня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання
			ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка архітектури програми.	4–5 тижні	Розділ 2
4	Опис програми.	6-7 тиждень	Розділ 3
5	Тестування і результати	8–9 тижні	Розділ 4
6	Оформлення пояснювальної записки та документів до неї.	10-13 тижні	Розділ 5, 6
			Додатки
7	Нормоконтроль та рецензування.	14–15 тижні	
8	Захист роботи.	16 тиждень	

Студент(ка)

_____ Дьячков В.В.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи)

_____ Федорончак Т.В.
(підпис) (прізвище та ініці)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
138 с., 42 рис., 7 табл., 3 дод., 26 джерел.

РЕКОМЕНДАЦІЙНА СИСТЕМА, ВОЛОНТЕРСТВО,
ВОЛОНТЕРСЬКИЙ КООРДИНАТОР, ПІДСИСТЕМА, ВЕБЗАСТОСУНОК,
ШТУЧНИЙ ІНТЕЛЕКТ, ЗНАХОДЖЕННЯ БЛИЗЬКОСТИ ТЕКСТІВ,
РЕКОМЕНДАЦІЇ ЗАСНОВАНІ НА КОНТЕНТІ, PYTHON, NLP, BACKEND,
WEB DEVELOPMENT, DJANGO, DJANGO REST FRAMEWORK, CELERY,
SPACY, POSTGRESQL, DOCKER, UWSGI, NGINX.

Об'єкт дослідження – розподілення ресурсів волонтерської організації між наявними запитами на допомогу.

Мета дипломної роботи – дослідження і програмна реалізація рекомендаційної системи для волонтерського координатора.

З мети випливає і завдання до проєкту – створення програмного забезпечення, що задовольнило би потреби волонтерів і отримувачів допомоги у ефективному встановленні комунікації один з одним.

Предмет дослідження – ті чинники, що впливають на рішення, що приймає координатор волонтерської організації під час визначення, який саме ресурс організації буде направлено на задовільнення того чи іншого запита на допомогу.

Актуальність цього дослідження і розробки полягає в наявності великого попиту на надання і отримання допомоги в період війни, а також потреба автоматизувати процеси пошуку отримувачів допомоги і волонтерів за заданими критеріями.

Результат виконання роботи – створена рекомендаційна система, що надає персоналізовані рекомендації користувачам, базуючись на залишених ними запитах і пропозиціях про надання допомоги.

ABSTRACT

Explanatory note to the diploma qualifying work of the master degree: 138 pages, 42 figures, 7 tables, 3 appendixes, 26 sources.

RECOMMENDER SYSTEM, VOLUNTEERING, VOLUNTEER COORDINATOR, SUBSYSTEM, WEB-APPLICATION, ARTIFICIAL INTELLIGENCE, FINDING TEXT SIMILARITY, CONTENT-BASED RECOMMENDATIONS, PYTHON, NATURAL LANGUAGE PROCESSING, BACKEND, DJANGO, DJANGO REST FRAMEWORK, CELERY, SPACY, POSTGRESQL, DOCKER, UWSGI, NGINX.

The object of the research is the distribution of resources of a volunteer organization between available help requests.

The purpose of this work is to research the possible implementation ways and to implement the recommender system for a volunteer coordinator.

The main project task is to create a software that will satisfy the needs of volunteers and help seekers to effectively establish communication with each other.

The subject of this research is the counters that influence the decision that is made by manual volunteer coordinator when they are determining which volunteer organization resource will be direct to satisfy which human need.

The relevance of these research and developments is in the presence of a large demand for providing and receiving aid during the war, as well as an urgent need to automate the processes of finding aid recipients and volunteers according to specified criteria.

The result of this work is a recommender system that generates personalized recommendations for users based on their help requests and proposals.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	7
Вступ.....	9
1 Аналіз предметної області.....	12
1.1 Опис предметної області.....	12
1.2 Огляд систем-аналогів	18
1.3 Висновки за розділом	24
2 Розробка архітектури програми	25
2.1 Структурна схема програми	25
2.2 Опис компонентів розроблюваної системи.....	28
2.3 Вибір інструментарію розробки програмного забезпечення	35
3 Опис програми	43
3.2 Основні рішення щодо реалізації компонентів системи.....	54
3.3 Підсумки опису програми.....	61
4 Тестування і результати.....	62
4.1 Опис методики проведення тестування	62
4.2 Результати тестування	64
5 Керівництво програміста	70
6 Керівництво системного програміста.....	74
Висновки	77
Перелік джерел посилання	79
Додаток А Технічне завдання	82
Додаток Б Текст програми.....	87
Додаток В Слайди презентації	129

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	–	Application Programming Interface;
CRUD	–	Create, Read, Update, Delete;
CSRF	–	Cross-Site Request Forgery;
DRF	–	Django REST Framework;
GUI	–	Graphical User Interface;
HTTP	–	HyperText Transport Protocol;
HTTPS	–	HyperText Transport Protocol Secure;
JSON	–	JavaScript Object Notation;
MVC	–	Model-View-Controller;
NDCG	–	Normalized Discounted Cumulative Gain;
NLP	–	Natural Language Processing;
ORM	–	Object-Relational Mapping;
REST	–	Representational State Transfer;
SQL	–	Structured Query Language;
TDD	–	Test Driven Development;
TOML	–	Tom's Obvious, Minimal Language;
XML	–	Extensible Markup Language;
XSS	–	Cross-Site Scripting;
YAML	–	Yet Another Markup Language;
UX	–	User Experience;
БД	–	база даних;
ВПО	–	внутрішньо переміщена особа;
ГО	–	громадська організація;
ОПМ	–	обробка природної мови;

- ОС – операційна система;
- ПЗ – програмне забезпечення;
- ПММ – паливно-мастильні матеріали;
- СКБД – система керування базами даних;
- УІ – універсальний ідентифікатор.

ВСТУП

Волонтерство існувало з давніх часів і завжди було проявом гуманності, доброти і найліпших людських чеснот. Небайдуже ставлення професіоналів і людей з активною громадянською позицією до проблем суспільства і готовність добровільно працювати на його благо завжди було одним з важливих факторів розвитку цього самого суспільства.

З початком повномасштабного вторгнення Росії в Україну, волонтерська робота є актуальною як ніколи раніше [1]: допомога внутрішньо переміщеним особам (ВПО), а також людям, що постраждали або втратили майно в ході бойових дій, постачання харчів, ліків, вивезення людей з тимчасово окупованих територій і, звісно, допомога збройним силам оборони України, – все це стало повсякденною рутиною для волонтерських організацій і благодійних фондів в нашій державі.

З різким збільшенням кількості запитів на волонтерську допомогу постала проблема автоматизації робочих процесів всередині волонтерського руху [2]. Як і у всіх інших сферах людської діяльності, все що може бути віддано на аутсорс (від англ. *outsource* – підрядництво) машині згодом або перетворюються на мобільні, веб чи десктоп застосунки, або на роботизовані машини, що здатні виконувати повсякденні задачі, що раніше були покладені на плечі волонтерів.

Першим з процесів, що було автоматизовано у цій сфері, став збір і обробка запитів і пропозицій на надання волонтерської допомоги. Завдяки тому, що координаторам і менеджерам волонтерських організацій і штабів більше не треба було власноруч збирати і зберігати анкети з запитами людей, вони змогли приділяти більше робочого часу іншим, більш нагальним проблемам. Цю задачу, як правило, кожна волонтерська організація вирішує сама для себе, оскільки в кожній є свій унікальний перелік послуг, що вони реалізують, а також свої сталі процеси обробки запитів, що надходять.

Другою, популярною для вирішення проблемою, є проблема організації самих волонтерів в середині волонтерського штабу. Існує безліч як безкоштовних так і пропрієтарних застосунків, що дозволяють визначати в календарі якісь івенти (від англ. *event* – подія), назначати на них виконавців, розподіляти роботу між різними волонтерами тощо. Зазвичай, волонтерські організації просто використовують готові рішення для закриття описаної проблеми.

Прямо зараз ведеться розробка програмного продукту, мобільного застосунку, що ставить собі на меті закрити потребу однієї з волонтерських організацій в швидкій обробці запитів і волонтерських пропозицій. Проте, значну міру роботи штабу все ще складає координація волонтерської допомоги між тими, хто її потребує. Наявність рекомендаційної системи, що підказувала би співробітникам волонтерської організації, яким саме чином найкраще використати ресурси штабу, куди саме треба направити допомогу, значно пришвидшила би його роботу і звільнила би робочий час координаторів для вирішення інших проблем, зокрема, з організації робочого процесу і керування іншими волонтерами всередині їх структури.

Актуальність даної роботи визначається нагальною потребою однієї з волонтерських організацій в автоматизації низки своїх робочих процесів задля оптимізації своєї роботи та більш ефективного надання допомоги потребуючим того людям або організаціям, структурам тощо.

Отож, об'єктом дослідження є розподілення ресурсів волонтерської організації між наявними запитами на допомогу. Предметом дослідження будуть ті чинники, що впливають на рішення, що приймає координатор волонтерської організації під час визначення, який саме ресурс організації буде направлено на задовільнення того чи іншого запита на допомогу.

Метою цієї роботи є практична реалізація рекомендаційної системи, що стане майбутнім компонентом наявного волонтерського координатора. Для досягнення мети, необхідно провести дослідження наявних рішень поставленої проблеми та визначити оптимальний метод реалізації

рекомендаційної системи. Крім того, необхідно знайти таке технічне рішення, що дозволяло би якнайменшими зусиллями інтегрувати отриману підсистему до вже наявної, більшої системи волонтерського координатора.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

В даному розділі наводиться опис основних функціональних частин основного компоненту системи-волонтерського координатора, який на момент написання роботи знаходиться в стані розробки. Також, наводиться деталізований опис предметної області відповідно до того, як вона змодельована на платформі. Крім того, даний розділ містить результати огляду систем-аналогів, онлайн платформ волонтерських організацій і благодійних фондів. За результатами проведеного аналізу було складено технічне завдання до проєкту, що наведено у висновках до розділу.

1.1 Опис предметної області

Опис предметної області наводиться відповідно до того, як вона змодельована у системі “Волонтерський координатор”. На момент написання роботи, система знаходиться на етапі розробки, тому кінцевий варіант моделі предметної області може зазнати змін, в тому числі запропонованих в результаті проведеного в даному розділі аналізу.

Проєкт “Волонтерського координатора” ставить собі на меті створити платформу для ефективного пошуку і отримання волонтерської допомоги для осіб, що її потребують, та реалізації цієї допомоги особами, що мають можливості та бажання її надати.

1.1.1 Користувачі системи

В системі наявні декілька категорій користувачів, що знаходяться у постійній взаємодії між собою. Детальний опис категорій користувачів наведено у табл. 1.1.

Таблиця 1.1 – Категорії користувачів

Назва категорії	Короткий опис	Системні задачі
Отримувачі допомоги	Люди, що потребують волонтерської допомоги (ВПО, постраждалі, сили оборони України)	Створення запитів на надання допомоги, пошук та вибір підходящих пропозицій на надання допомоги
Волонтери	Особи, що можуть надати допомогу	Створення пропозицій про надання тих чи інших послуг на волонтерських засадах, комунікація з отримувачами допомоги для її реалізації
Адміністратори	Співробітники волонтерської організації	Модерація запитів та пропозицій про допомогу, що надходять від інших користувачів системи, допомога волонтерам з організації процесу надання допомоги і розподілення їх ресурсів між різними отримувачами

1.1.2 Запити і пропозиції на надання допомоги

Комунікація між різними користувачами системи відбувається за рахунок створення запитів і пропозицій на надання допомоги, чи заявок. Отримувачі допомоги мають можливість переглядати список пропозицій, волонтери навпаки – список заявок. Підібравши оптимальну і найбільш відповідну для себе заявку, користувачі мають можливість зв'язатися один з одним за вказаними в заявці контактами, а саме номером телефону, месенджерами, прив'язаними до номеру або адресою електронної пошти.

Запити і пропозиції на надання допомоги заводяться в системі відповідно до визначених в ній категорій. Кожна категорія має власний унікальний перелік атрибутів (деталей про допомогу), що користувач має заповнити при оформленні заявки. Повний список категорій і їх атрибутів наведено в табл. 1.2.

Таблиця 1.2 – Атрибути запитів і пропозицій відповідно до категорій

Назва категорії	Атрибут	Значення	Примітка
Атрибути, спільні для всіх категорій			
	Заголовок	Текст	
	Детальний опис	Текст	
	Чи є терміновим	Так / Ні	
	Автор	Користувач	Визначається автоматично в момент створення
Транспортні категорії			
Перевезти вантаж	Звідки	Місто	
	Куди	Місто	
	Вага	Ціле число, кг	
	Довжина	Дробне число, м	Тільки для запитів
	Ширина	Дробне число, м	Тільки для запитів
	Висота	Дробне число, м	Тільки для запитів

Продовження таблиці 1.2

Назва категорії	Атрибут	Значення	Примітка
Транспортні категорії			
Перевезти людей	Звідки	Місто	
	Куди	Місто	
	Кількість	Ціле число, осіб	
	Діти	Так / Ні	
	Багаж	Так / Ні	Тільки для запитів
	Домашні тварини	Так / Ні	
Перегнати авто	Звідки	Місто	
	Куди	Місто	
	Паливо	Дизель/Бензин	Тільки для запитів
	Коробка передач	Ручна/Автомат.	Тільки для запитів
Житло			
Житло в Україні	Де	Місто	
	Кількість людей	Ціле число, осіб	
	Кількість кімнат	Ціле число, кімнат	
	Тип приміщення	Дім / Квартира / Хостел	
	Підселення	Так / Ні	
	Строк	Доба / Тиждень / Довгостроково	
Гуманітарна допомога			
Їжа і продукти	Де	Місто	
	Кількість людей	Ціле число, осіб	Тільки для запитів
	Строк	Ціле число, днів	Від 1 до 15

Продовження таблиці 1.2

Назва категорії	Атрибут	Значення	Примітка
Гуманітарна допомога			
Одяг і взуття	Де	Місто	
Медикаменти	Де	Місто	
Товари, послуги і ресурси			
Послуги і людські ресурси	Де	Місто	
Харчова сировина	Де	Місто	
Нехарчові товари	Де	Місто	
Будматеріали та інструменти	Де	Місто	
Інше			
Доставка паливно-мастильних матеріалів (ПММ)	Де	Місто	
	Тип	Дизель / Бензин	
	Обсяг	Ціле число, л	

Як вже зазначалось раніше, після заповнення всіх атрибутів запиту чи пропозиції на надання допомоги, заявка повинна пройти попередню модерацію адміністраторами системи перш ніж потрапити до загальної видачі всім іншим користувачам платформи волонтерського координатора. Адміністратори перевіряють правильність і коректність заповнення заявки, в разі потреби контактують з автором заявки задля уточнення її актуальності.

В разі схвалення адміністратором, заявка відображається для інших користувачів: запити на надання допомоги – волонтерам, пропозиції на надання допомоги - відповідно, людям що її потребують. Саме на цьому етапі буде доречним залучення рекомендаційної системи аби оптимізувати процес пошуку відповідної заявки користувачами, скоротити час від її попадання в

пошукову видачу до втілення в життя. Слід зазначити, що функціональність з пошуку запитів і пропозицій на надання допомоги ще не є імplementованою на момент написання роботи, а тому фінальний її варіант може зазнати змін, в тому числі запропонованих з урахуванням пропозицій, що можуть спростити розробку і покращити ефективність майбутньої рекомендаційної системи, описаних пізніше в цьому розділі.

Після того як користувач знайде заявку, що відповідає його потребам, він може зв'язатися з її автором для того, аби отримати чи надати описану допомогу.

Після безпосереднього надання або отримання допомоги, необхідно підтвердження отримувача допомоги про те, що заявка тепер може бути закритою.

В разі відхилення заявки, вона отримує відповідний статус “Відхилено”. Автор заявки отримує сповіщення про причини такого вердикту і, в разі потреби, може подати повторну заявку з урахуванням усіх зауважень адміністратора.

Життєвий цикл заявок запиту і пропозиції на надання допомоги зображено на рис. 1.1 у вигляді блок-схеми.

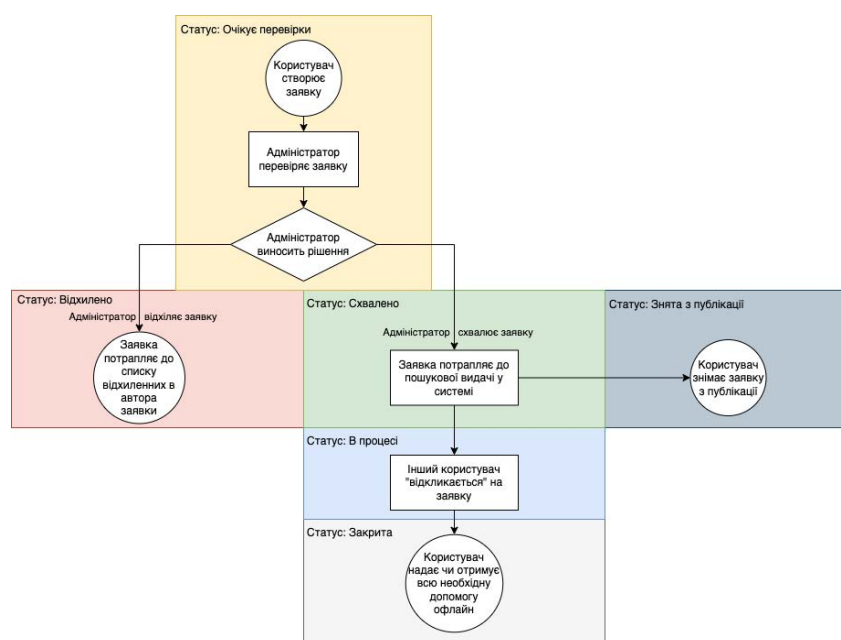


Рисунок 1.1 – Життєвий цикл запиту і пропозиції на надання допомоги

1.2 Огляд систем-аналогів

Огляд систем-аналогів ставить собі метою описати наявні в світі рішення з побудови рекомендаційної системи для онлайн-платформи волонтерського координатора, визначити основні критерії, за яким користувачі здійснюють пошук запитів і пропозицій на надання допомоги, що в майбутньому може стати вимогами до реалізації систему пошуку у системі “Волонтерський координатор” та стати підґрунтям для імплементації рекомендаційної системи. Результатом проведеного аналізу мають стати вимоги до побудови рекомендаційної системи, що стануть частиною технічного завдання до проєкту.

1.2.1 Огляд сайту “Української Волонтерської Служби”

Найбільшою і першою у пошуковій видачі є вебплатформа “Українською Волонтерською Служба” або “volunteer.country” [3]. Це сайт однойменної громадської організації (ГО). Її головною особливістю, зокрема, є можливість пошуку волонтерів для своїх власних проєктів. Запити на надання допомоги або, як вони називаються тут в системі, “можливості” наводяться списком (рис. 1.2) і доступні для перегляду будь-якими користувачами. Платформа також надає можливість пошуку “можливостей” на мапі (рис. 1.3), що є цікавою з точки зору UX (англ. “user experience” – дослівно “досвід користування платформою”) особливістю, що дозволяє користувачам досить швидко знаходити підходящі для себе можливості, що знаходяться близько до їх місцезнаходження, без того, щоб гортати сторінки за сторінкою заявок або налаштовувати відповідні фільтра за їх поточним місцем перебування та радіусом припустимого пошуку.

Зображений на рис. 1.2 фільтр з рядком для пошуку є єдиним способом звужити коло отриманих результатів. Рядок для пошуку здійснює повнотекстовий пошук за заголовком і детальним описом “можливості”.

Фільтри нижче надають змогу обрати тільки ті категорії волонтерської допомоги, що цікавлять користувача. Є можливість обрати декілька категорій одночасно.

Як видно з рис. 1.3, на платформі є можливість не тільки розміщувати одноразові пропозиції про допомогу чи оголошення про пошук волонтерів, але й виставляти та додавати на мапу адреси своїх волонтерських штабів та організацій, що функціонують на постійній основі.

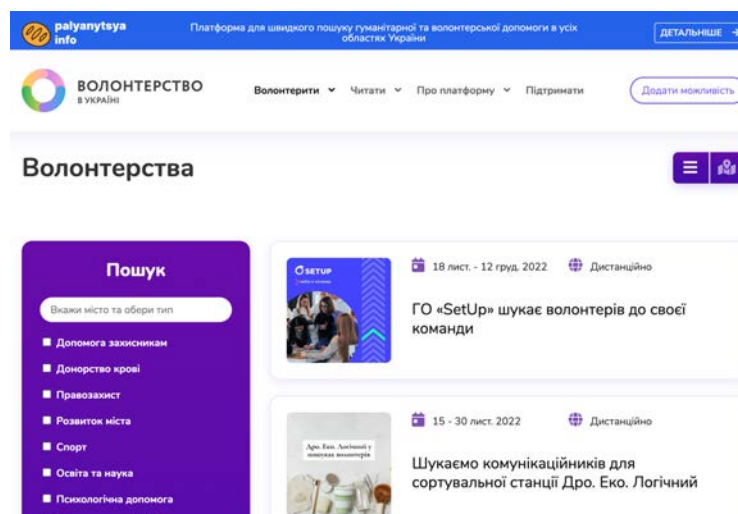


Рисунок 1.2 – Пошук можливостей на платформі “Української Волонтерської Служби”

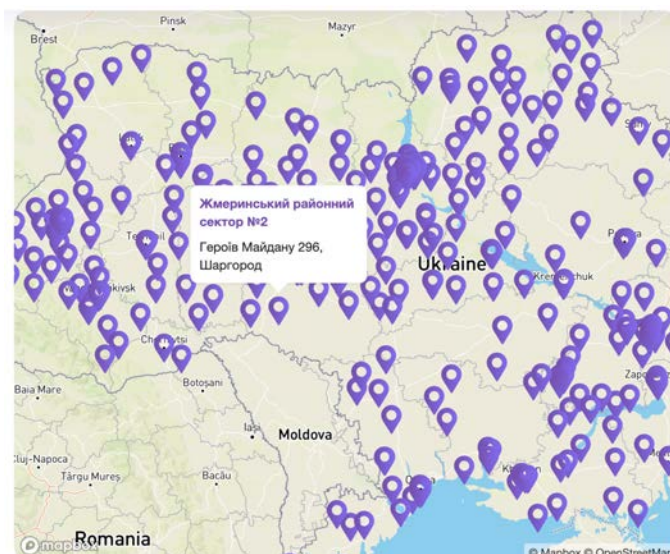


Рисунок 1.3 – Перегляд можливостей на мапі

Загалом, платформа не вирізняється якоюсь винятковою функціональністю чи зручністю користування, а головною її перевагою можна назвати те, що вона є найбільшою з існуючих в Україні і нараховує значну кількість пропозицій про надання допомоги, а також згаданих волонтерських організацій в своїй системі.

1.2.2 Огляд онлайн-платформи “UA Helpers”

Онлайн-платформа “UA Helpers” є найбільш схожою за своєю суттю до системи “Волонтерських координатор”. Вона так само агрегує в собі запити (або “прохання”, як вони називаються в системі) і пропозиції на надання допомоги, розміщують їх у себе на сайті після проведеної модерації та дають користувачам можливість переглядати увесь список, самостійно контактуючи з волонтерами і особами, що потребують допомоги [4]. Зовнішній вигляд головної сторінки вебсайту наведено на рис. 1.4.

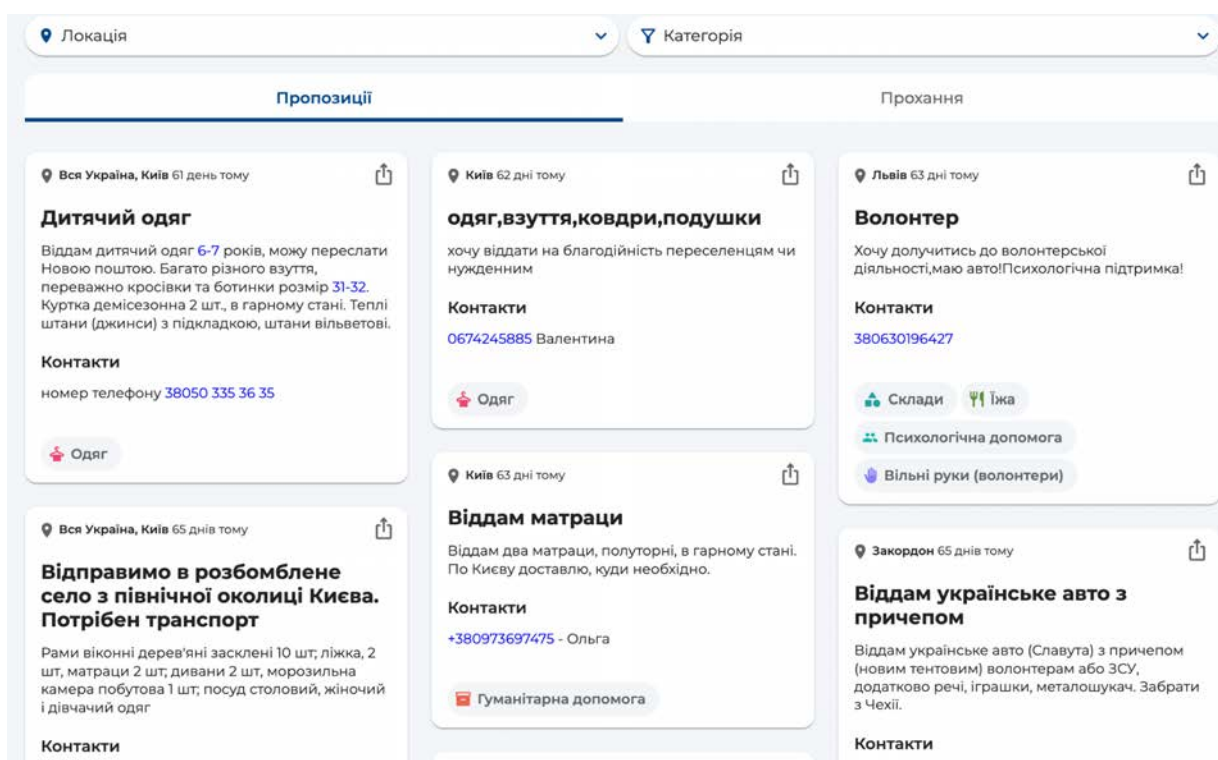


Рисунок 1.4 – Список пропозицій на головній сторінці сайту “UA Helpers”

Для подачі своєї пропозиції чи прохання необхідно надати всі її подробиці за посиланням на “Google Forms”, розміщеним у шапці сайту. Після подачі заявки, її мануально обробляють координатори платформи та розміщують у списку на головній сторінці.

Як видно з рис. 1.4, користувач має можливість фільтрувати за місцезнаходженням та категорією пропозиції чи прохання. Оскільки на платформі відсутня реєстрація, ніякої рекомендаційної системи не передбачено. Відсутня можливість повнотекстового пошуку за деталями заявки.

1.2.3 Огляд онлайн-платформи “Волонтер.орг”

Так само як і “volunteer.country”, онлайн-платформа “Волонтер.орг” допомагає волонтерам знаходити можливості для реалізації власного бажання допомогти і долучитися до вже існуючих проєктів, а волонтерським організаціям – знайти для себе нові ресурси [5].

Прямо на головній сторінці сайту (рис. 1.5) знаходиться повний перелік усіх активних проєктів, активностей, оголошень тощо. Так само як і в інших системах, є можливість фільтрації результатів за категорією і місцезнаходженням.

Не зважаючи на реалізовану можливість реєстрації, видача результатів в списку доступних активностей ніяк не буде персоналізована, а пропозиції про підходящі для вас проєкти роздаються мануально координаторами платформи.

Крім усього вищезазначеного, платформа не дозволяє користувачам розміщувати свої запити на отримання допомоги. При наявному бажанні допомогти, координатори платформи самостійно можуть направити вас на осіб, що потребують допомоги, що, звісно, займає час і, ймовірно, потребує автоматизації.

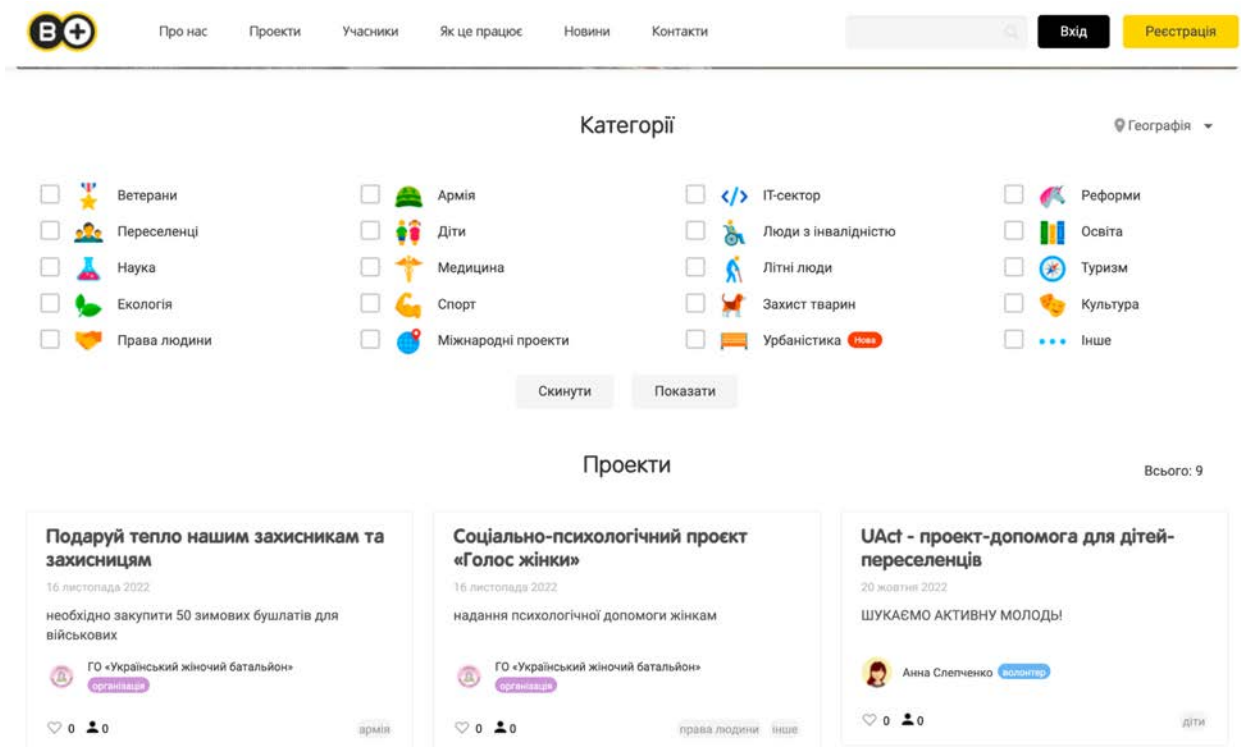


Рисунок 1.5 – Головна сторінка “Волонтер.орг” з переліком усіх доступних активностей, проєктів і можливостей для волонтерів

1.2.4 Аналіз огляду систем-аналогів

Підбиваючи підсумки з проведеного огляду систем-аналогів, треба зазначити, що жодна з розглянутих систем не мала вбудованої рекомендаційної системи на сайті. Більшість процесів, зокрема з пошуку співпадінь між пропозиціями та проханнями користувачів виконуються власноруч адміністраторами систем, а власне, можливості мануального пошуку для користувачів зазвичай є доволі обмеженими. Порівняння усіх трьох розглянутих систем наведена у табл. 1.3.

Загалом, роблячи висновки з огляду систем-аналогів можна зазначити, що рекомендації майбутньої системи повинні в першу чергу ґрунтуватись на близькості “наповнення” пропозиції чи запиту на надання допомоги, що зазвичай є простим текстовим описом, а також їх місцезнаходженні і категорії

допомоги, а також, ймовірно, на відповідності атрибутів запиту і пропозиції, специфічних для кожної категорії допомоги.

Таблиця 1.3 – Результати порівняння систем-аналогів

Критерій порівняння	Назва системи		
	volunteer.country	UA Helpers	Волонтер .org
Можливість пошуку пропозицій	✗	✓	✗
Можливість пошуку запитів	✓	✓	✓
Можливість повнотекстового пошуку за описом	✓	✗	✗
Можливість пошуку за місцезнаходженням	✓	✓	✓
Можливість пошуку за категорією допомоги	✓	✓	✓
Наявність системи авторизації для збереження користувацької інформації і історії активності	✗	✗	✓
Наявність рекомендацій при пошуку пропозицій чи запитів	✗	✗	✗

Підсумовуючи, система рекомендацій має повторювати поведінку користувачів під час власноручного пошуку пропозицій і запитів, що відбувається за рахунок використання наявних можливостей пошуку і фільтрації результатів.

Персоналізовані рекомендації для користувача, після його авторизації, можуть надаватись у вигляді підходящих пропозицій на залишені ним запити на надання допомоги, і навпаки – запитів за залишеними пропозиціями. Під час генерації персоналізованих рекомендацій повинна враховуватись історія взаємодії як самого конкретного користувача, так і всіх користувачів системи загалом.

1.3 Висновки за розділом

За результатами проведеного аналізу предметної області, розгляду категорій користувачів системи “Волонтерський координатор” та їх поведінки всередині системи, а також спираючись на рекомендації, сформовані в результаті огляду систем-аналогів, перелічені у п. 1.2.4 було складено документ – технічне завдання до проєкту побудови рекомендаційної системи для системи “Волонтерський координатор”, що наведено у додатку А.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

В другому розділі описано процес проектування рекомендаційної системи, відповідно до визначених у технічному завданні критеріїв. Проводиться аналіз наявних інструментів для розробки програмного забезпечення, їх вибір згідно до поставленої задачі, формується структурна схема програми, методологія обробки вхідних даних, а також зберігання вихідних, проектується дизайн програмного забезпечення, що розробляється.

2.1 Структурна схема програми

Перед проектуванням окремих модулів і компонентів системи, насамперед, треба визначитись з її базовою архітектурою. Вдалий вибір архітектури може вберегти розробника від прийняття помилкових рішень або необхідності витратити занадто багато часу на організацію роботи у системі, а не на реалізацію бізнес-логіки, тобто логіки роботи програми.

Після детального аналізу технічного завдання, було прийнято рішення про реалізацію клієнт-серверної архітектури, оскільки система, що розробляється має бути модульною і легко інтегрованою з іншими компонентами системи “Волонтерський координатор”, які вже реалізують дану архітектуру. Це також дозволить розробляти і тестувати рекомендаційну систему незалежно від інших компонентів системи.

Графічне представлення клієнт-серверної архітектури наведено на рисунку 2.1 [6].

Сама система має бути представлена у вигляді вебзастосунку з відкритим API (англ. “Application Programming Interface”), аби розробники інших компонентів системи “Волонтерський координатор” могли знаходити вірний застосування рекомендаційній системі на власний розсуд, інтегрувавши її з системним компонентом авторизації і аутентифікації користувачів [7].

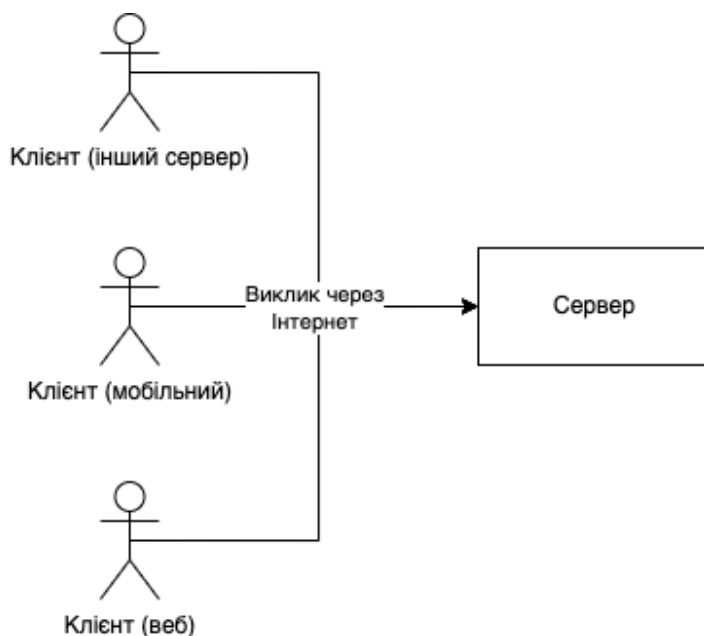


Рисунок 2.1 – Графічне представлення клієнт-серверної архітектури

Принцип взаємодії з іншими компонентами має бути побудований за підходом REST (англ. “Representational State Transfer”), оскільки він стандартизує підхід до взаємодії, є індустріальним стандартом в сфері реалізації клієнт-серверних API, легко реалізується на практиці та не потребує від клієнта встановлення жодних залежностей для взаємодії за принципами REST [8].

REST-підхід заснований на побудові взаємодії між клієнтом і сервером у вигляді надання сервером клієнту можливості проведення певного списку операцій, таких як перегляд, створення, редагування та видалення, над деякими “ресурсами” в системі. У випадку з системою, що проектується, головним ресурсом виступають персоналізовані рекомендації користувачів. Надсилаючи запит, клієнт повинен мати можливість переглянути список створених спеціально для нього рекомендацій.

Нижче наведені основні правила (або архітектурні обмеження) створення RESTful застосунків:

- використання клієнт-серверної архітектури (рішення про використання якої вже було прийнято на етапі проектування);

- відсутність стану, тобто кожен запит містить всю необхідну інформацію для його обробки, і не покладається на те, що сервер знає щось з попереднього запиту;
- кешування, тобто дані, які передаються сервером, повинні містити інформацію про те, чи можна їх кешувати, і якщо можна, то як довго;
- однорідний інтерфейс, який зменшує зв'язність між компонентами і сервісами які вони надають і дозволяє нескладно змінювати компоненти за потреби;
- шари абстракції: кожен компонент потрапляє в якийсь шар, і спілкується лише з компонентами в шарі під ним або в шарі над ним. Обмеження знання системи одним шаром зменшує складність компонентів;
- запитування коду (необов'язкове): клієнти повинні дозволяти розширювати свою функціональність дозволяючи завантаження додаткового коду в формі аплетів чи скриптів.

В якості транспорту для комунікації між різними компонентами системи має використовуватись протокол HTTPS (англ. “HyperText Transfer Protocol Secure”), що є поліпшеною версією протоколу HTTP (англ. “HyperText Transfer Protocol”), що підтримує шифрування даних для їх безпечної передачі по мережі. Він є стандартом при побудові клієнт-серверних застосунків та забезпечує гарантоване передавання даних між клієнтом і сервером. Крім того, протокол HTTPS може використовуватись як протокол транспортного рівня і дарує гнучкість проєктувальникам і розробникам у виборі протоколу (формату) передачі даних між компонентами.

В якості формату передачі даних рекомендується обрати формат JSON (англ. “JavaScript Object Notation”). Він легко сприймається людиною, є дуже популярним серед розробників і є компактнішим за аналогічні текстові формати типу XML (англ. “Extensible Markup Language”), YAML (англ. “Yet Another Markup Language”), TOML (англ. “Tom's Obvious, Minimal

Language”), тобто для передачі одних і тих самих даних використовується менша кількість байт інформації, аніж у всіх вищезазначених.

Остаточний варіант запропонованої архітектури взаємодії між клієнтом і сервером в системі, що усі вищезазначені підходи, протоколи та формати передачі даних подано на рис. 2.2.

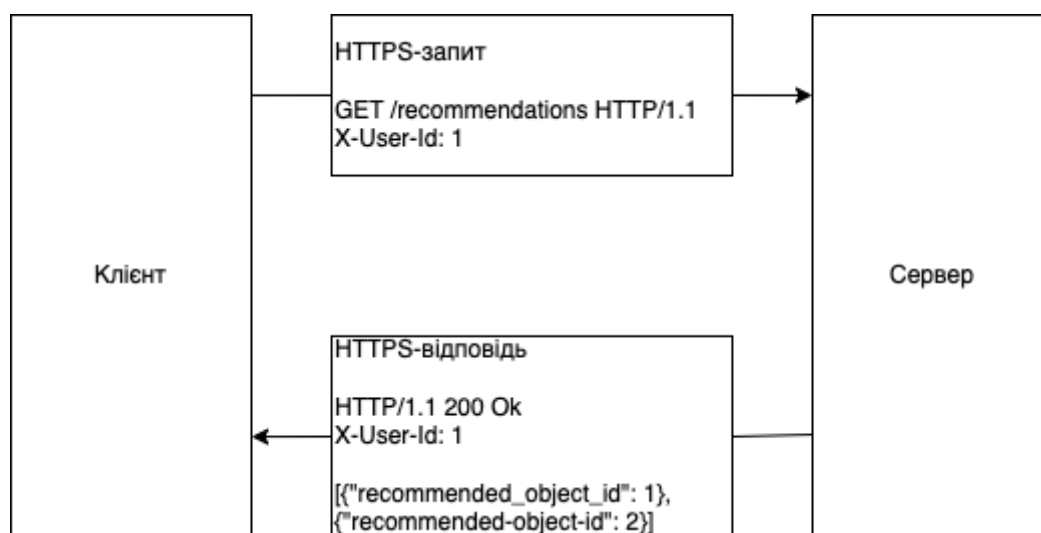


Рисунок 2.2 – Запропонований варіант архітектури взаємодії між клієнтом і сервером

Складена архітектура взаємодії не диктує розробнику ніяких конкретних способів її реалізації, бо може бути побудована використовуючи засоби будь-якої мови програмування загального призначення, а також за допомогою великої кількості бібліотек та фреймворків, як для клієнтської так і для серверної частини. Крім того, клієнтська і серверна частина можуть бути побудовані використовуючи різні інструменти розробки, що також є безумовним плюсом і додає гнучкості в розробці.

2.2 Опис компонентів розроблюваної системи

Розділення системи на компоненти значно пришвидшує процес розробки, оскільки дає змогу розробнику зосередитись на виборі

оптимального інструменту для закриття кожної окремої, конкретної потреби. Модульність системи дозволяє легко замінювати одні компоненти на інші в майбутньому, не змінюючи при цьому високорівневу архітектуру системи.

Перш ніж переходити до визначення компонентів системи, слід повернутися трохи назад до функціональних вимог. ПЗ, що проєктується, мусить надавати можливість користувачам отримувати список рекомендованих пропозицій відповідно до опублікованого ними запиту на надання допомоги, і навпаки – запитів до опублікованої пропозиції. Виходячи з цього сценарію, рекомендаційна система повинна мати наступні компоненти:

- вебзастосунок з відкритим програмним інтерфейсом, як вже було описано в п. 2.1, що він буде надавати можливість клієнту отримати список рекомендацій. Це буде основний компонент системи, що звертатиметься до всіх інших;
- рекомендаційний субмодуль, що буде частиною вебзастосунку і що доступ до нього клієнт буде отримувати через відкритий інтерфейс вебзастосунку;
- сховище аналітичних даних, що вони будуть отримані в результаті проведених системою розрахунків з знаходження ступеню збіжності між пропозиціями і запитами користувачів;
- виконувач фонових задач для обрахунку ступеню збіжності при додаванні пропозицій і запитів в систему.

Діаграма залежності компонентів зображена на рис. 2.3.



Рисунок 2.3 – Діаграма залежності компонентів

2.2.1 Проектування вебзастосунку

Основний компонент рекомендаційної системи, представлений у вигляді вебзастосунку, обробляти запити, що надсилаються клієнтською частиною, опрацьовувати дані, що приходять на сервер, організовувати їх для збереження у системі, реалізовувати роботу зі спільною базою (сховищем) даних (БД).

Відповідно до прийнятих на поточний момент практик розробки вебзастосунків, даний компонент системи має використовувати MVC-архітектуру (“Модель - Представлення - Контролер”, від англ. “Model - View - Controller”) [9]. Графічне представлення цієї архітектури наведено на рисунку 2.4.

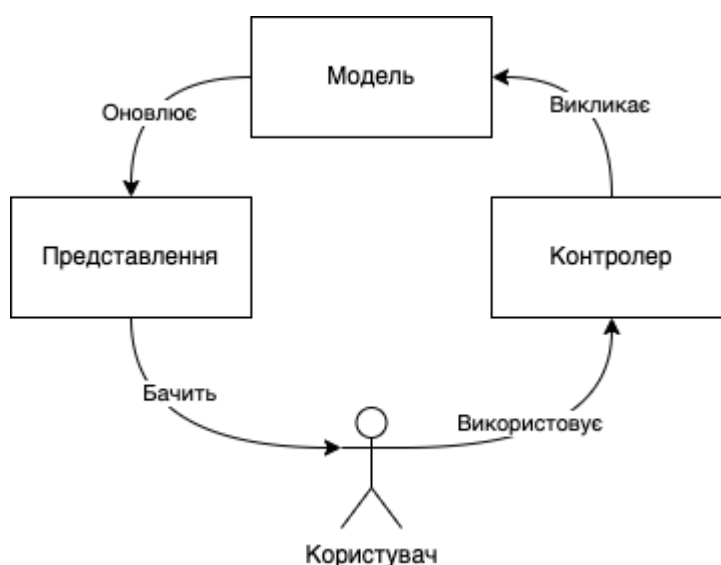


Рисунок 2.4 – Схематичне зображення MVC-архітектури

Використання подібного шаблону проектування дозволить подрібнити серверну частину вебзастосунку на ще дрібніші модулі, розробка і модульне тестування яких може йти майже незалежно один від одного, а, отже, ймовірна модифікація системи не призводить до значних проблем.

Нижче наводиться коротка характеристика виділених модулів:

- модель (англ. “Model”) - частина програми, що відповідає за реалізацію усієї бізнес-логіки, тобто логіки роботи з даними, їх обробки, форматування, зберігання у базі даних тощо;
- представлення (англ. “View”) - частина програми, яку видно клієнту (користувачу), тобто будь-які текстові дані, в нашому випадку, у форматі JSON;
- контролер (англ. “Controller”) - частина програми, яка зв’язує між собою модель і представлення: використовуючи інтерфейс моделі оновлює представлення для користувача.

2.2.2 Проєктування бази даних

Важливою частиною проєктування будь-якої системи є рішення про організацію даних в ній. Для системи, що розробляється, було створено ERD (англ. “Entity Relationship Diagram”) [10], яка зображена на рис. 2.5.

ERD описує концептуальну схему зв’язків між виділеними сутностями предметної області. Вона дозволяє уявити структуру даних в середині застосунку, не прив’язуючись до конкретної системи керування базами даних (СКБД).

Нижче наведено перелік виділених сутностей предметної області:

- заявка про допомогу – складена користувачем заявка, що являє собою запит або пропозицію на надання допомоги;
- задоволення заявки – історично обрана користувачем заявка, що задовольняє складену ним заявку, що формують пари запит-пропозиція;
- користувач – волонтер чи отримувач допомоги, що розміщує свої оголошення (запити, пропозиції) в системі);
- рекомендація – розрахована системою збіжність між парою заявка-пропозиція, що являє собою вірогідність (у відсотках), з якою дану заявку можна порекомендувати іншій, відповідній заявці.

Кожна сутність наділена штучним унікальним ідентифікатором (УІ). В деяких випадках це обумовлено відсутністю реального УІ, як то з запитами і пропозиціями про надання допомоги. В деяких випадках це зумовлено тим, що атрибути, що можуть ідентифікувати сутність можуть змінюватись в часі. Наприклад, це стосується адреси електронної пошти і номеру телефона користувача, що є унікальними за своєю природою, але користувач мусить мати можливість змінювати їх в налаштуваннях свого профілю.

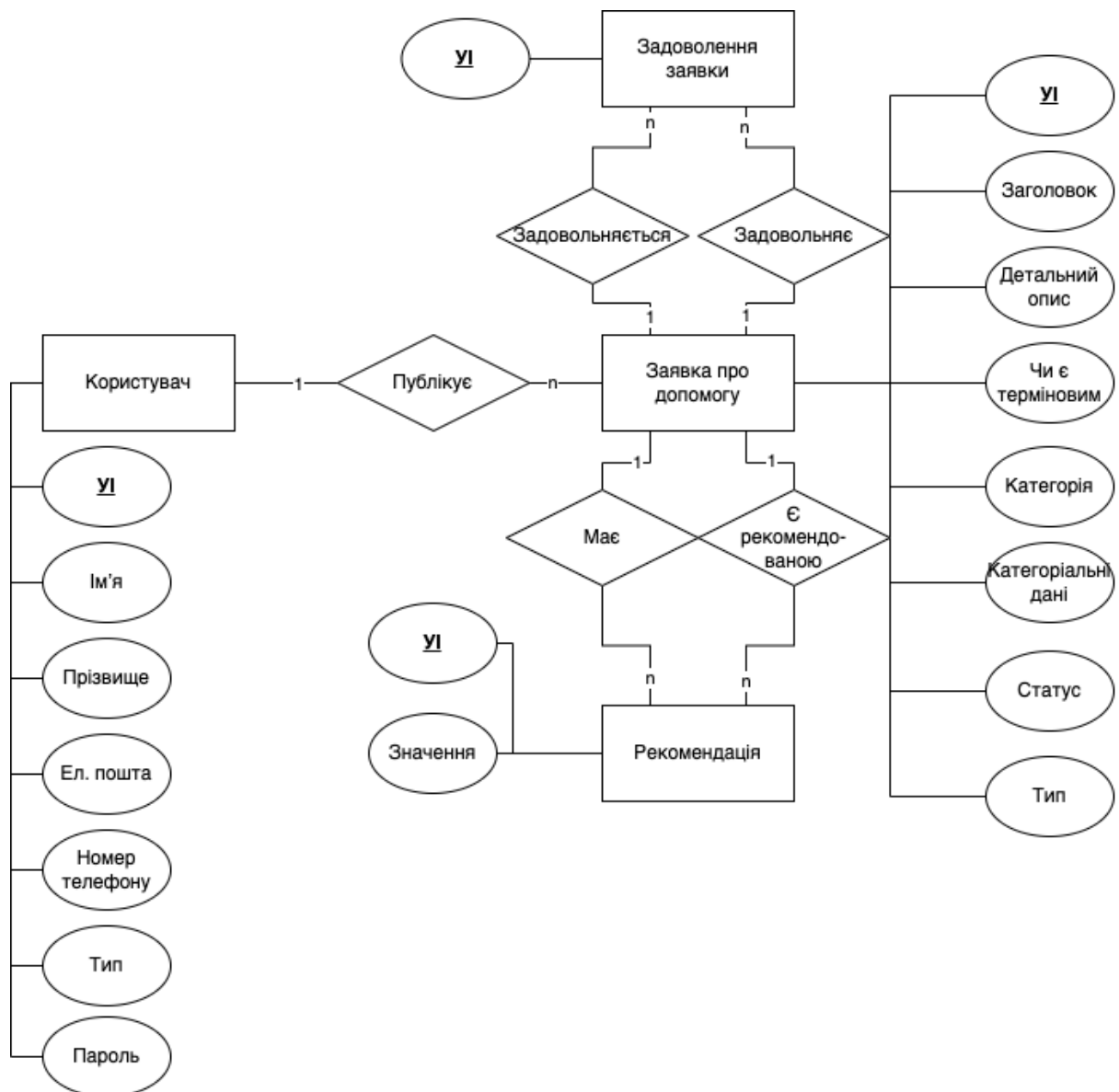


Рисунок 2.5 – ERD предметної області

2.2.3 Проектування дизайнів клієнтської частини

Хоча розробка клієнтської частини програми не є темою роботи, однак розробка дизайнів для неї може допомогти при проектуванні методів програмного інтерфейсу. Знаючи які саме дані повинні бути представлені в графічному інтерфейсі користувача (GUI, англ. “Graphical User Interface”), можна оптимізувати відповіді сервера, щоби вони не включали в себе зайвих атрибутів, і завантаження сторінки в користувача займало менше часу і трафіку.

Згідно з функціональними вимогами до ПЗ, викладеними в технічному завданні до проєкту, користувач мусить мати можливість переглядати короткий список рекомендованих пропозицій на надання допомоги відповідно до залишеного ним запиту, і навпаки – список рекомендованих запитів до залишеної пропозиції.

Відповідно до визначених потреб було складено вайрфрейми майбутніх сторінок у користувацькому вебзастосунку (рис. 2.6). Вайрфрейм (англ. “Wireframe”) - це образ дизайну низької точності, не сильно деталізований рисунок дизайну в чорно-білому вигляді з упором не на візуальну складову, а на структуру і зміст [11]. Основна мета створення вайрфреймів полягає в тому, щоб зобразити загальну функціональність самого продукту. Вони відображають такі речі, як навігація, основні групи змісту, кнопки, керуючі елементи і загальний макет. Це спосіб візуалізації того, як елементи інтерфейсу будуть відображатися для користувачів на найважливіших екранах.

З розроблених вайрфреймів видно, що для сторінок перегляду запиту чи пропозиції на надання допомоги необхідно розробити по одному публічному методу програмного інтерфейсу, що буде повертати список рекомендацій для розглядаємого об’єкту. Структура відповіді від серверу повинна включати в себе заголовки рекомендованих запитів чи пропозицій разом зі ступенем їх схожості з розглядаємим об’єктом.

Заголовок

Детальний опис запиту чи пропозиції на надання допомоги, що включає в себе повний опис ресурсів, що потребуються або можуть бути надані.

Опис може складатись з декількох абзаців.

Рекомендації

Рекомендація #1	81%
Рекомендація #2	79%
Рекомендація #3	73%
Рекомендація #4	67%
Рекомендація #5	63%

Місто	Терміново	Опубліковано	Атрибут N
Запоріжжя	Так	01/11/2022	Значення N

Редагувати

Приховати

Рисунок 2.6 – Вайрфрейм сторінки детального перегляду запиту чи пропозиції на надання допомогу зі списком рекомендацій

2.2.4 Проектування рекомендаційного субмодулю

Проектування сервісу з надання персоналізованих рекомендацій, що має стати субмодулем проєктованого вебзастосунку, потребує окремого розгляду.

Рекомендаційні системи, в загальному, бувають двох типів: засновані на змісті, або ж контенті, (англ. “Content-Based”) і засновані на схожості вибору (англ. “collaborative filtering”). Розглянемо обидві на прикладі побудови рекомендаційної системи для онлайн кінотеатру.

Перші, ті що засновані на змісті, будуть враховувати історію активності користувача і рекомендувати для нього об’єкти, що схожі на ті, які він вже обрав тим чи іншим чином в минулому. Вертаючись до приклада з кінотеатром, якщо користувач подивився трилогію “Iron Man”, то система порекомендує йому фільми про супергероїв або з Робертом Дауні-молодшим в ролях, оскільки вони будуть схожі на його історію перегляду.

Другі, що засновані на схожості вибору різних користувачів, або на колаборативній фільтрації, шукають схожість між різними користувачами, і

якщо користувач з якоїсь групи обрав для себе певний об'єкт, то система буде рекомендувати його іншим користувачам з тої самої групи. Наприклад, якщо є декілька користувачів, котрим подобається жанр фентезі, і один з них подивився фільм в жанрі детективу, то система порекомендує цей фільм іншим користувачам, що до цього надавали перевагу жанру фентезі.

Під час розробки рекомендаційної системи для волонтерського координатору буде надано перевагу першому типу систем, а саме рекомендаційним системам заснованим на змісті. Це зумовлено кількома факторами.

По-перше, користувачі системи, особливо отримувачі допомоги, ймовірно, не будуть залишати багато запитів на надання допомоги. Через це, буде дуже складно скласти їх в групи, оскільки під час надання рекомендацій даних про користувача буде занадто мало для генерації релевантних пропозицій.

По-друге, заявки і пропозиції про надання допомоги мають однакову структуру, а тому знайти їх близькість за змістом, а саме за тими їх ознаками, що були визначені на етапі аналізу предметної області і що мають вирішальне значення при мануальному пошуку запитів і пропозицій про надання допомоги користувачами, буде легшою задачею, аніж знаходити схожість користувачів і формувати з них "групи" за спільними потребами.

Навчання "моделі" рекомендаційної системи повинно відбуватись, враховуючи вибір всіх користувачів платформи в минулому. Всі створені під час користування платформою пари запит-пропозиція повинні зберігатись в базу даних, для того щоб в майбутньому використовуватись для розрахунку вкладів кожної ознаки запиту чи пропозицію на кінцевий результат "схожості" двох заявок між собою. Модель буде вважатися вдалою, якщо вона зможе повторити вибори користувачів в минулому під час тестування.

2.3 Вибір інструментарію розробки програмного забезпечення

Як і вибір правильної архітектури, вибір відповідних інструментів може вберегти розробника від випадкових помилок, значно пришвидшити розробку системи і привести її у потрібний напрямок. До таких інструментів належать мова програмування, фреймворки, програмні платформи, сторонні бібліотеки, сервіси та програми, середовища розробки і таке інше.

Відповідно до визначеної у підрозділі 2.1 структури програми, а також рішення про роздільну розробку окремих компонентів системи, необхідно обрати інструменти для:

- розробки і тестування серверного вебзастосунку;
- роботи з сховищем (базою даних);
- автоматизованого розгортання системи.

Окрім вищезазначених пунктів також необхідно визначитись з використанням допоміжних інструментів для розробки системи в цілому, формування документації тощо.

Підходячи до вибору треба враховувати наступні фактори:

- популярність того чи іншого інструменту;
- наявність готових рішень поставленої проблеми;
- можливість інтеграції конкретного інструменту з іншими, що використовуються під час розробки і роботи системи;
- досвід використання інструменту розробником.

Використовуючи вищезазначені критерії було обрано інструменти для розробки і підтримки рекомендаційної системи для волонтерського координатора. Результати порівняльного аналізу і обґрунтування кожного вибору наведено нижче.

2.3.1 Вибір інструментів для розробки і тестування серверного вебзастосунку

На сьогоднішній день існує багато варіантів для написання вебзастосунків, представлених як спеціалізованими рішеннями, як, наприклад, мови програмування PHP, Perl і подібні, так і різноманітні бібліотеки, фреймворки та програмні платформи, написані на мовах програмування загального застосування, таких як C++, C#, Java, JavaScript, Python, Ruby та інші.

Результати порівняння мов програмування наведено у табл. 2.1. Для порівняння були обрані мови, що найчастіше використовуються у розробці рекомендаційних систем, що зумовлює наявність багатьох готових рішень [12].

Таблиця 2.1 - Результати порівняння різних мов програмування

Мова програмування	Java	JavaScript	Python	R
Особливість				
Наявність вебфреймворків для написання серверної частини	+	+	+	-
Наявність готових бібліотек для написання рекомендаційних систем	+	+	+	-
Наявність повномасштабної стандартної бібліотеки, що надає інструменти для вирішення більшої частини типових задач	+/-	-	+	-
Простота синтаксису	+/-	+/-	+	+/-
Швидкість роботи	+	+	+/-	+
Швидкість розробки	-	+	+	+/-

З-поміж великого загалу мов програмування і їх діалектів, що надають готові рішення поставленої проблеми, було обрано мову програмування Python. Нижче наведені основні переваги цього вибору:

- простий синтаксис Python, що, при правильному проектуванні програм, читається подібно до англійської мови (також, під простим синтаксисом треба розуміти відсутність великої кількості спеціальних символів і ключових слів, що наявні в інших мовах програмування);
- високий рівень абстракції мови програмування, який дозволяє розробнику приділяти більше уваги логіці роботи програми, а не її написанню;
- наявність великої спільноти: десятки, чи навіть сотні тисяч розробників по всьому світу, що впливають на життя Python, підтримуючи саму мову, її бібліотеки, фреймворки та інші програмні засоби, а також документацію та керівництва у актуальному стані, відповідно до сучасних стандартів написання програм [13].

Python має велику кількість створених спільнотою вебфреймворків, найпопулярнішим з яких є Django. Під час написання цього проекту буде використано його надбудову – Django REST Framework (DRF), що призначений для реалізації RESTful застосунків. Застосування цих двох інструментів, зокрема, дозволить відмовитись від:

- написання свого інтерфейсу роботи з базою даних: Django має власну ORM (англ. “Object-relational mapping”), яка інтегрована з більшістю СКБД (таких, як, наприклад, SQLite, MS SQL Server, MySQL, PostgreSQL, MongoDB та інші);
- написання власної панелі адміністратора: Django автоматично генерує панель адміністратора, яку можна легко налаштувати;
- пошуку стороннього інструменту для тестування: Django та DRF мають власні підсистеми для тестування написаного коду;

- реалізації вручну MVC-архітектури: структура Django-застосунку не дає можливості відхилитися від написання коду з використанням цієї архітектури;
- серіалізації та десериалізації вхідних та вихідних даних: DRF автоматично приводить дані до формату JSON [14].

Крім того, для DRF написані бібліотеки для автоматичного документування API за стандартами Swagger. Приклад Swagger документації наведено на рис 2.7.

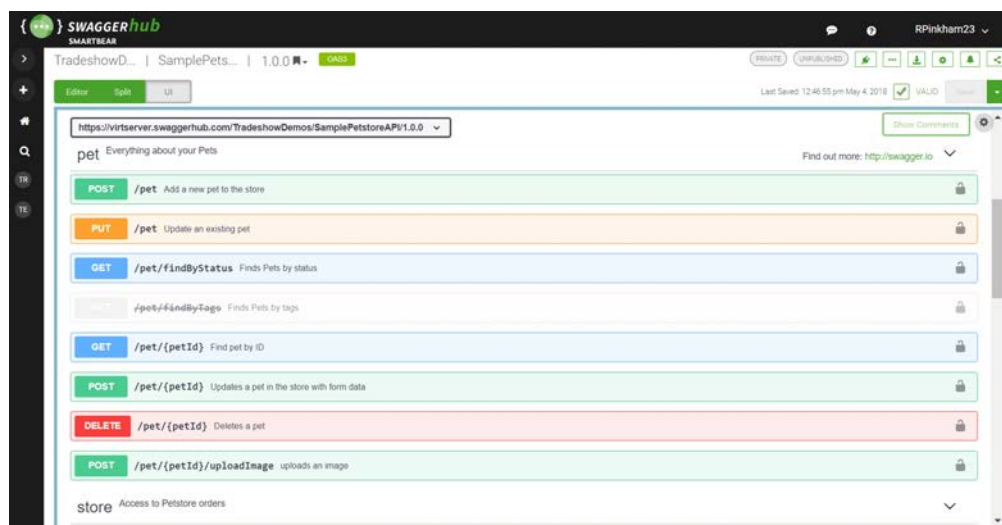


Рисунок 2.7 – Приклад документації API у форматі Swagger

Окрім вищезазначених переваг, Django REST Framework пропонує автоматичну валідацію даних, а також інструментів виконання CRUD (англ. “Create Read Update Delete”) операцій для визначених у застосунку сутностей відповідно до REST-архітектури.

Python також має багато готових рішень для написання рекомендаційних систем, імплементації машинного навчання і штучного інтелекту, як, наприклад, TensorFlow, PyTorch, SciKit, Theano, Pandas, NumPy, Keras [15].

Для вирішення проблеми контролю залежностей у Python-проекті буде використовуватись інструмент Poetry. Він дозволить слідкувати за

оновленнями встановлених сторонніх бібліотек та фреймворки, уникати їх конфліктів та переносити програму на інші пристрої, виконуючи всі дії у спеціальному віртуальному середовищі.

2.3.2 Вибір інструментів для виконувача задач

Як вже було зазначено раніше, одним з компонентів системи, що проєктується буде виконувач періодичних і фонових задач, що вони обраховують ступені збіжності між запитами і пропозиціями на надання допомоги.

Враховуючи обрані раніше інструменти для написання вебзастосунку, серед усіх наявних варіантів було віддано перевагу фреймворку Celery на мові програмування Python.

По-перше, це ультимативне рішення, що дозволяє виконувати як періодичні, так і фонові задачі, а також підтримує багато різних сховищ для зберігання результатів виконання задач і організації їх в черги.

По-друге, це рішення може бути легко інтегровано з іншими компонентами і може розділяти з ними спільну кодову базу, оскільки може бути написано, використовуючи ту саму мову програмування Python.

Крім того, Celery має інтеграцію з обраним раніше фреймворком для написання вебзастосунку – Django. Вона може використовувати наявні в Django засоби ORM, конфігурувати через налаштування вебфреймворку, а також використовувати базу даних, що керується за допомогою Django, як сховище для періодичних задач.

2.3.3 Вибір інструментів для роботи з базою даних

Під “інструментами для роботи з базою даних” слід розуміти систему керування базами даних, яка буде використовуватись для організації збереження інформації у системи.

Для використання у даному програмному продукті була обрана СКБД PostgreSQL, для роботи з котрою і була розроблена ORM фреймворку Django, який є основним засобом для написання серверної частини вебзастосунку, що відповідає за роботу за базою даних.

PostgreSQL - об'єктно-реляційна система керування базами даних. Вона є альтернативою як комерційним СКБД (наприклад, Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (як MySQL, Firebird, SQLite). Порівняно з іншими проектами з відкритим кодом, PostgreSQL не контролюється якоюсь однією компанією, її розробка можлива завдяки співпраці багатьох людей та компаній, які хочуть використовувати цю СКБД та впроваджувати у неї найновіші досягнення, а, отже, завжди реалізує найновіші і найактуальніші підходи для роботи з базами даних.

2.3.4 Вибір інструментів для автоматизованого розгортання системи

Розгортання системи - не менш важливий процес ніж написання і тестування програмного коду. Можливість переносити, масштабувати і доповнювати систему є дуже важливою у процесі підтримки програмного забезпечення.

З метою вирішення цієї проблеми було прийняте рішення про використання інструменту віртуалізації Docker і його надбудови Docker Compose. Кожен окремий компонент системи може бути винесений в окремий контейнер, що дозволяє не прив'язуватись до конкретної операційної системи, її параметрів і налаштувань.

На відміну від віртуальних машин, Docker використовує значно менше пам'яті комп'ютеру, оскільки створює не повне середовище системи, а лише необхідну її частину для функціонування окремого компоненту.

Програми, які упаковані в Docker-контейнери, можуть бути виконані у будь-якій системі, яка підтримується програмним забезпеченням Docker.

2.3.5 Вибір допоміжних інструментів

Велику роль під час створення програмного забезпечення, крім інструментів для написання коду і розгортання системи, грають засоби для автоматизації процесу розробки. До таких відносяться середовища розробки, аналізатори коду, системи контролю версій тощо.

Для вирішення задачі контролю версій під час розробки проекту було обрано Git - централізовану систему, що дозволяє зберігати стан програмного забезпечення на різних етапах його розробки. Також, робота з Git надає можливість використання одного з віддалених сховищ Git-репозиторіїв, таких як, наприклад, Github, Bitbucket, Gitlab та інші. У якості віддаленого сховища для Git-репозиторіїв було обрано сервіс Gitlab, через великий досвід використання, а також можливість використання систем безперервної доставки та розгортання Gitlab CI під час розробки.

Система безперервної інтеграції та доставки дозволяє автоматизувати процес тестування коду після його завантаження до Git-репозиторію, пакування проекту до Docker-контейнерів та їх завантаження до віддаленого сховища Docker Hub.

У якості головного інструменту для розробки було обрано PyCharm Professional – інтегроване середовище розробки від компанії JetBrains, що пропонує комплексне рішення для написання коду мовою програмування Python. Завдяки можливості використання розширень, що покращують наявний функціонал, або пропонують другий, PyCharm може стати потужним інструментом для розробки проекту в цілому. За замовчуванням він інтегрований для роботи з системою контролю версій Git, має вбудоване розширення для менеджменту Docker-контейнерів. Крім того, дане середовище розробки має інструменти для статичного аналізу коду, його автодоповнення, підсвічування, а також контроль за дотриманням стилю написання програмного коду.

3 ОПИС ПРОГРАМИ

В третьому розділі наводиться опис алгоритму функціонування програми, а також обґрунтування прийнятих в ході розробки рішень щодо реалізації компонентів системи, визначених на етапі проектування.

3.1 Алгоритм функціонування програми

3.1.1 Алгоритм функціонування рекомендаційної підсистеми

Рекомендаційний субмодуль, що він є частиною вебзастосунку і головним серцем розроблюваного програмного забезпечення, має свої власні особливості імплементації і функціонування.

Як вже було визначено на етапі проектування, розроблювана рекомендаційна модель повинна надавати користувацькі рекомендації, засновані на зміст [16]. Це значить, що для надання рекомендацій для довільної пари запит-пропозиція система має розрахувати ступінь їх збіжності. Для простоти сприйняття звичайними користувачами, ступінь збіжності двох заявок буде подано у відсотковому значенні, де 100% значить, що дві заявки є ідентичними.

Схожість, чи то близькість, двох заявок повинна бути розрахована спираючись на відповідність їх ознак, визначених на етапі аналізу програмного забезпечення, а саме:

- близькості детального текстового опису однієї заявки до іншої;
- близькості текстів заголовку однієї заявки до іншої;
- близькості місцезнаходження однієї заявки до іншої;
- збіжності статусу терміновості однієї заявки з іншою;
- близькості дат відтермінування (втрати актуальності) обох заявок;
- збіжності значень, специфічних для кожної категорії заявок.

За кожним з перелічених пунктів буде вираховано відсоток схожості. У разі, якщо значення обох ознак двох заявок є ідентичними, то відсоток схожості складає 100%. Для ознак, в яких розраховується близькість значень, відсоток схожості може приймати значення від 0 до 100 включно. Для ознак, в яких розраховується збіжність значень, відсоток схожості може приймати значення 0 або 100.

Після розрахунку відсотку схожості кожної окремої ознаки заявок, вони будуть помножені на відповідний коефіцієнт вкладу та просумовані, як показано у формулі (3.1):

$$s = \sum_{i=1}^N a_i c_i, \quad (3.1)$$

де s – це ступінь схожості, що приймає значення у діапазоні $[0;1]$;

a_i – i -та ознака (атрибут) заявки;

c_i – коефіцієнт вкладу i -ї ознаки заявки;

N – кількість ознак.

Сума коефіцієнтів вкладу кожної ознаки є 1. Конкретне значення кожного окремого коефіцієнта буде розраховане таким чином, щоб отриманий результат давав найвищий результат при оцінці якості моделі. Методика оцінювання якості моделі буде розглянута пізніше в даному розділі.

Крім того, нульову збіжність мусять мати заявки, якщо вони є одного і того самого типу (запит, пропозиція), або мають різну категорію допомоги (транспорт, гуманітарна, ліки, тощо). Це зумовлено тим, що вказані заявки з системних і логічних обмежень не можуть бути складені в пару запит-пропозиція.

3.1.1.1 Знаходження близькості текстів

Для розрахунку ступеню збіжності текстових ознак, таких як то детальний текстовий опис заявки чи її короткий заголовок, необхідно програмно імплементувати алгоритм пошуку близькості двох довільних текстів українською мовою.

Проблемами аналізу довільних текстів піклується обробка природної мови (від англ. “Natural Language Processing”, ОПМ) – галузь штучного інтелекту, що фокусується на вирішенні проблеми розуміння машиною людської мови. ОПМ поєднує комп’ютерну лінгвістику – моделювання людської мови на основі правил, використовуючи статистичні моделями, моделі машинного та глибокого навчання. Разом ці технології дозволяють комп’ютерам обробляти людську мову у формі тексту або голосових даних і «розуміти» її повне значення разом із наміром і почуттям мовця чи письменника [17].

Існує декілька основних методів визначення близькості двох довільних текстів. Перший, та найпростіший носить назву “Індекс Жаккара” (ІЖ), що наведено у формулі (3.2):

$$\text{ІЖ} = \frac{|A \cap B|}{|A \cup B|}, \quad (3.2)$$

де А і В – множини слів порівнюваних текстів відповідно (з повтореннями).

Принцип цього методу полягає в тому, щоб знайти кількість слів, що є спільними для обох текстів і розрахувати відсоткове значення кількості спільних слів від загальної кількості слів в обох текстах (формула 3.2). Наприклад, для речень “Пляшка води є порожньою” та “Води немає в пляшці” спільними є слова “пляшка” та “вода”. Графічне представлення ІЖ на даному прикладі зображено на рис. 3.1.



Рисунок 3.1 – Схематичне зображення індексу Жаккара

Слід зауважити, що для більш ефективного використання індексу Жаккара, як і інших методів визначення близькості текстів, вхідний текст має пройти попередню обробку, а саме розбиття його на окремі лексеми (слова). Цей процес називається токенизацією. Процес токенизації також може супроводжуватись виключенням з тексту усіх токенів, що не мають власного значення. До таких відносяться, наприклад, розділові знаки, а також так звані стоп-слова, найчастіше допоміжні частини мови, зокрема, сполучники (такі як “а”, “також”, “і”, “й” тощо) [18].

Для посилення ефекту токенизації, отримані лексеми також проходять етап трансформації: стеммінгу (англ. “stemming”), тобто знаходження основи в слова, або лемматизації (англ. “lemmatization”), тобто приведення слова в початкову форму [19]. Порівняння слів після стеммінгу та лемматизації наведено в табл. 3.1.

Таблиця 3.1 – Порівняння слів після стеммінгу і лемматизації

Слово	Результат стеммінгу	Результат лемматизації
безпритульними	безпритул	безпритульні
письменник	письм	письменник
письмо	письм	письмо
шукач	шука	шукач
шукатимуть	шука	шукати

Як видно з результатів порівняння в табл. 3.1, лемматизація краще здатна передавати сенс слова і зберігає його семантику, оскільки слова після стеммінгу, представлені своїми основами, не передають інформації про те, зокрема, до якої частини мови відноситься слово. Після стеммінгу, слова “письменник” та “письмо” будуть представляти собою однакові токени.

Але, навіть після попередньої обробки тексту, а саме його токенизації і лемматизації, значення індексу Жаккара для розглянутого прикладу буде складати всього $2/7 = 0.28$, що є дуже низьким результатом, хоча тексти є дуже близькими за значенням. Це пов'язано з тим, що ІЖ відображає синтаксичну близькість текстів. Проблему пошуку близькості текстів за значенням займаються алгоритми семантичного пошуку близькості текстів, що будуть розглянуті пізніше.

Алгоритми, що знаходять семантичну близькість текстів, потребують ще одного етапу обробки тексту. Він носить назву векторизація, чи то вкладання слів (англ. “Word embedding”). Це є загальною назвою низки методик мовного моделювання та навчання ознак в ОПМ, в яких слова або фрази зі словника відображають у вектори дійсних чисел. Концептуально воно дає математичне вкладення з простору з багатьма вимірами, по одному на слово, до неперервного векторного простору набагато нижчої розмірності [20]. Оскільки перетворення слів на дійсні числа задача нетривіальна, для векторизації так само існує безліч методів, серед яких було обрано розроблений спеціалістами з Google алгоритм Word2Vec – група пов'язаних моделей, які використовують для вкладання слів. Ці моделі є пласкими двошаровими нейронними мережами, тренуваними відтворювати лінгвістичний контекст слів. Word2vec бере як вхід великий корпус тексту, й виробляє векторний простір, зазвичай з кількома сотнями вимірів, де кожному унікальному слову з цього корпусу призначено відповідний вектор у цьому просторі. Векторні представлення слів розташовуються в цьому векторному просторі таким чином, що слова, які поділяють спільний контекст у корпусі, розташовуються близько одне до одного в цьому просторі [21].

Графічне зображення повного процесу обробки тексту для його підготовки до використання алгоритмами знаходження семантичної близькості текстів зображено на рис. 3.2.

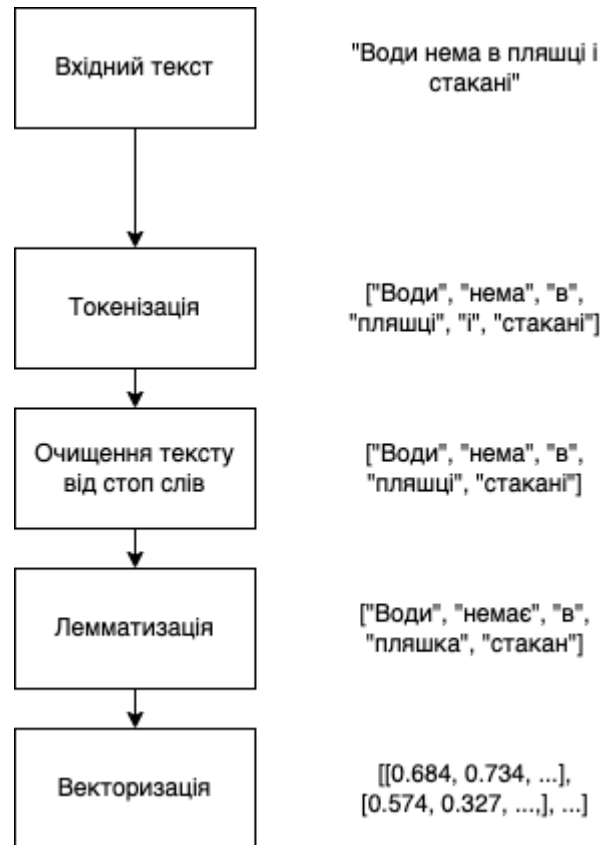


Рисунок 3.2 – Графічне представлення підготовки тексту для знаходження його семантичної близькості з іншими текстами

Після проведеної етапу вкладання слів, між отриманими векторами слів можна розраховувати відстань. Найпростіший спосіб це зробити – розрахувати Евклідову відстань, що подано на формулі (3.3):

$$ED = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}, \quad (3.3)$$

де ED – Евклідова відстань між двома векторами $p = [p_1, p_2, \dots, p_n]$ та $q = [q_1, q_2, \dots, q_n]$.

Проблемою при розрахунку Евклідової відстані є той факт, що отримане значення відстані не є нормалізованим і потребує знаходження додаткового методу нормалізації.

Цю проблему покриває собою ще один спосіб знаходження семантичної близькості текстів – розрахунок косинусу подібності між векторами текстів за формулою (3.4):

$$S_c(A, B) = \cos(\Theta) = \frac{AB}{|A||B|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}, \quad (3.4)$$

де $S_c(A, B)$ – косинус подібності між векторами A і B;

Θ – кут між векторами A і B;

A_i – i -тий компонент вектору A;

B_i – i -тий компонент вектору B.

Отриманий результат приймає значення $[-1; 1]$, а тому є легким для порівняння і представлення його у відсотках (для репрезентації може бути взяте значення по модулю). Саме через свою ефективність та простоту пояснення результатів, метод знаходження семантичної близькості текстів за допомогою розрахунку косинусу подібності між їх векторами був використаний під час розробки ПЗ.

3.1.1.2 Знаходження близькості двох міст

Для розрахунку ступеню схожості двох заявок також необхідно буде визначити, чи знаходяться вони географічно досить поруч, щоби автори цих заявок змогли сконтактувати між собою.

Користувачам буде надано можливість вибрати своє місцезнаходження з переліку всіх міст і територіальних громад України. Список всіх доступних варіантів місцезнаходження буде надаватись системою. Поруч з повною назвою міста будуть також зберігатись дані про його положення на карті: його довгота і широта.

Дані про список всіх міст і територіальних громад України, разом з точним географічним положенням їх центрів, можна отримати в багатьох відкритих джерелах, зокрема на ресурсі “simplemaps” [22].

Знаючи довготу і широту обраного місця, можливо розрахувати його відстань до інших точок на карті, використовуючи вже згадану раніше Евклідову відстань (3.3).

Евклідова відстань, як вже було зазначено у п 3.1.1.1, є не нормованою величиною, а тому, аби перевести її до формату значень близькості інших атрибутів, що знаходиться у діапазоні $[0;1]$ було прийнято рішення, в разі, якщо розрахована відстань між двома містами є нульовою (тобто автори двох заявок вказали одне і те саме місто), то результуюче значення близькості ознак двох заявок буде складати 1. В разі, якщо розрахована відстань буде перебільшувати порогове значення близькості в 50 кілометрів – результуюче значення набуває значення 0. Будь яка відстань від 0 до 50 км між містами буде давати значення в діапазоні $(0;1)$ і буде розраховане як значення в відсотках від 50 км.

3.1.1.3 Методика оцінювання моделі

Оскільки задача знаходження найбільш схожих об'єктів до якогось заданого об'єкту є по суті задачею ранжування, тобто впорядкування об'єктів за значенням від меншого до більшого. В контексті ПЗ, що розробляється, таким значенням буде ступінь схожості двох заявок.

Як вже зазначалось раніше на етапі проєктування, система буде зберігати дані про історичні вибори користувачів для закриття потреб власного запита чи пропозиції. Сформовані користувачами вручну пари запит-пропозиція повинні стати орієнтиром для моделі. Модель завжди повинна знаходити обрану користувачем пару як найбільш релевантну до даної заявки, і обрана методика оцінювання рекомендаційної моделі має враховувати це.

Після детального аналізу, для оцінювання моделі була обрана метрика nDCG (англ. “Normalized Discounted Cumulative Gain”), що є показником якості ранжування. Вона часто використовується для вимірювання якості алгоритмів вебпошуку чи пошуці релевантних документів. Використовуючи ступеневу шкалу релевантності в наборі результатів пошукової системи, DCG вимірює корисність, схожість або релевантність документу на основі його позиції в списку результатів [23]. Нижче наведено метод розрахунку метрики nDCG за формулою (3.5):

$$nDCG_p = \frac{DCG_p}{IDCG_p}, \quad (3.5)$$

де $nDCG_p$ – нормалізована метрика DCG для рангу p ;

DCG_p – метрика DCG для рангу p за формулою (3.6);

$IDCG_p$ – ідеальна метрика DCG для рангу p за формулою (3.7).

$$DCG_p = \sum_{i=1}^p \frac{rel_i}{\log_2(i+1)}, \quad (3.6)$$

де rel_i – i -та градуйована релевантність розглядаємого документу.

$$IDCG_p = \sum_{i=1}^{|REL_i|} \frac{rel_i}{\log_2(i+1)}, \quad (3.7)$$

де REL_i – список ідеально градуйованих документів в порядку спадання значення їх релевантності.

3.1.2 Алгоритм функціонування вебзастосунку

Вебзастосунок, як компонент програми, повинен забезпечувати взаємодію розроблюваного програмного забезпечення з іншими компонентами системи “Волонтерський координатор”. Він, як вже було зазначено на етапі проєктування, має бути реалізований у вигляді клієнт-серверного застосунку з відкритим програмним інтерфейсом, реалізованим за стандартом REST.

Для імплементації серверного вебзастосунку було обрано вебфреймворк Django, що написаний мовою програмування Python. Для всіх вебзастосунків, що реалізуються цією мовою програмування є єдиний стандарт клієнт-серверної комунікації, що носить назву WSGI (Web Server Gateway Interface). Він диктує методи, що вони мають бути реалізовані Python-програмою, аби бути WSGI-сумісними. Далі, WSGI-сумісні програми можуть виконуватись багатьма реалізаціями WSGI-серверу, що надають можливість оптимально оброблювати безліч клієнтських запитів, навіть під час високого навантаження. Принцип комунікації між клієнтом і сервером за протоколом WSGI зображено на рис. 3.3.

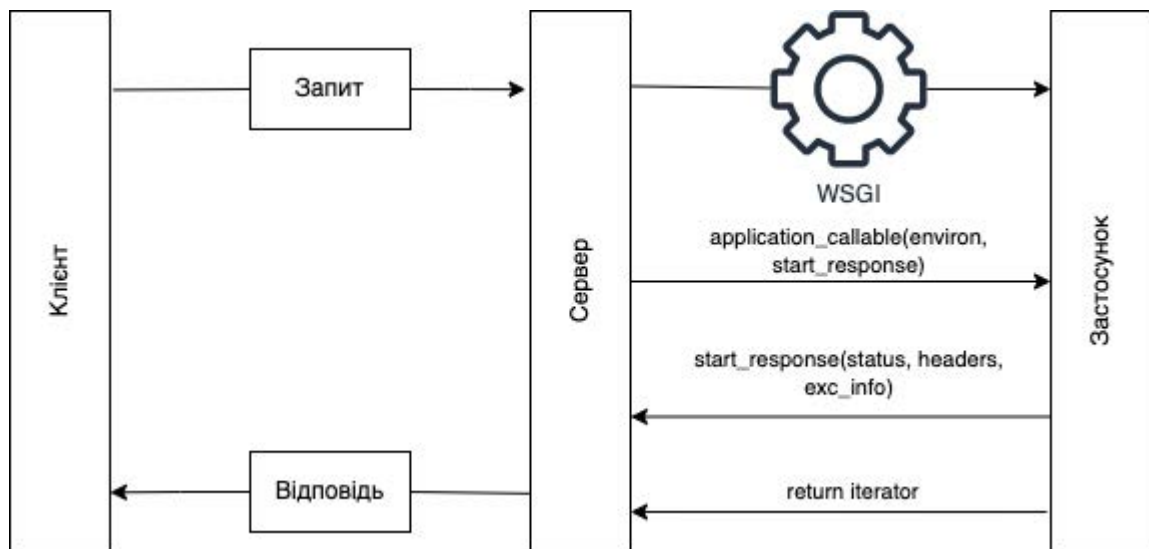


Рисунок 3.3 – Принцип роботи WSGI-застосунку

Потрапивши до Django-застосунку запит спочатку обробляється проміжними шаром (англ. “Middleware”), що видозмінює запити перед їх потраплянням до відповідного контролера і відповіді, після його роботи. Необхідний контролер (чи в термінології Django - “view”) обирається в залежності від того, до якої адреси (англ. “endpoint”) був направлений запит. Контролер, використовуючи інтерфейс моделі для роботи з даними і реалізації бізнес-логіки, тобто правил обробки інформації, відправляє відповідь клієнту, в нашому випадку, в текстовому форматі JSON з вказанням коду статусу HTTP (наприклад, 200 у випадку вдалого запиту, 400 - у випадку некоректно наданих даних, 403 через заборону звернення і так далі). Алгоритм взаємодії окремих модулів серверної частини (представлення, контролерів, моделі) було зображено на рис. 2.3.

Для приведення отриманих даних до Python-об’єктів, а також структурування відповідей до формату JSON у Django REST Framework використовуються серіалізатори (англ. “serializers”). Вони також відповідають за початкову валідацію даних, їх відповідність висунутим вимогам.

3.2 Основні рішення щодо реалізації компонентів системи

В даному підрозділі наводяться і обґрунтовуються деталі імплементації основних рішень, прийнятих на етапі проектування або зазначених в описі алгоритму функціонування програми.

3.2.1 Основні рішення щодо реалізації рекомендаційної підсистеми

Під час імплементації рекомендаційної підсистеми розроблюваного вебзастосунку постало декілька проблем, зокрема:

- проблема вибору інструментів для знаходження близькості двох довільних текстів;
- проблема реалізації методу оцінювання моделі;
- проблема підбору коефіцієнтів вкладу кожної окремої ознаки у загальний ступінь схожості між двома заявками;
- проблема навчання і інкапсуляції моделі.

3.2.1.1 Вибір інструментів для реалізації знаходження близькості двох довільних текстів

Для реалізації знаходження близькості двох довільних текстів існує чимало готових рішень, написаних мовою програмування Python, зокрема, такі бібліотеки як nltk, Gensim, а також бібліотека spaCy. Порівняння усіх зазначених рішень наведено в табл. 3.2.

Як видно з результатів порівняння, бібліотека spaCy має одну суттєву перевагу над усіма іншими рішеннями з ОПМ мовою програмування Python – готову навчену модель для токенізації, лемматизації і векторизації тексту українською мовою, причому представлені три додаткові бібліотеки даних різних розмірів, що натреновані на різних корпусах текстів, в основному, з новин і наукових статей. Крім того, бібліотека spaCy має більший допустимий

ліміт кількості символів у тексті, що опрацьовується – більше ніж 50000 символів, в той час як nltk і Gensim спроможні опрацьовувати тексти не більше ніж як 10000 символів довжиною.

Таблиця 3.2 – Порівняння ОПМ-бібліотек в Python

Критерій порівняння	Бібліотека		
	nltk	Gensim	spaCy
Наявність засобів токенизації тексту	✓	✓	✓
Наявність засобів лемматизації тексту	✓	✓	✓
Наявність засобів векторизації тексту	✓	✓	✓
Наявність засобів знаходження близькості текстових векторів	✓	✓	✓
Наявність готової моделі для роботи з українською мовою	✗	✗	✓

3.2.1.2 Основні рішення з реалізації методики оцінювання моделі

Для проведення оцінювання отриманої рекомендаційної моделі було підготовлено тестовий датасет, який включає в себе вже закриті користувацькі заявки, що мають для них пару, визначену користувачем.

Далі, оскільки кожна заявка може мати лише одну відповідну їй пару, обрану користувачів, для кожної заявки з тестового датасету, у співпраці з експертами предметної області, було вручну підібрано 4 інших заявки, дві з яких є так само схожими, інші дві – зовсім не схожими. Далі, потенційні пари для заявки, що розглядається, були відсортовані в порядку схожості, де на першому місці знаходиться найбільш релевантна заявка, а саме та, що була власноруч обрана користувачем в минулому, далі – менш релевантні.

Отриманий тестовий датасет і був використаний для оцінювання моделі за метрикою NDCG, алгоритм якої був описаний у п. 3.1.1.3.

3.2.1.3 Основні рішення щодо підбору коефіцієнтів вкладу кожної ознаки у загальний ступінь схожості між заявками

За допомогою експертів предметної області, було визначено початкові значення коефіцієнтів вкладу кожної ознаки у загальну ступінь схожості між заявками, де найбільше вклад має детальний текстовий опис заявки (0.5), а найменший – збіжність дат втрати заявками актуальності (0.02). Потім, використовуючи простий метод підбору, значення кожного коефіцієнту вкладу збільшувалось на 0.01, а інших – зменшувалось, при цьому проводячи повторну оцінку якості моделі, поки оцінка моделі не досягне свого найбільш можливого значення.

За результатами проведеного експерименту, було встановлено наступні коефіцієнти вкладу ознак у загальний ступінь схожості між заявками:

- близькість детального текстового опису однієї заявки до іншої – 0,34;
- близькість текстів заголовку однієї заявки до іншої – 0,09;
- близькості місцезнаходження однієї заявки до іншої – 0,35;
- збіжності статусу терміновості однієї заявки з іншою – 0,04;
- близькості дат відтермінування (втрати актуальності) обох заявок – 0,01;
- збіжності значень, специфічних для кожної категорії заявок – 0,17.

3.2.1.4 Основні рішення щодо навчання і інкапсуляції моделі

Для того, аби інкапсулювати логіку роботи рекомендаційної моделі, підвищити її ефективність в плані часу роботи, а також, щоб модель мала змогу працювати з записами, які до цього були їй не відомі, побудова моделі була виконана у вигляді побудови звичайної лінійної регресії.

Після підбору оптимальних коефіцієнтів вкладу, для навчального датасету було виконано функцію знаходження схожості між заявками. Схожість між двома заявками і є прогнозуєма величина в лінійній регресії. Вхідними даними, в цьому випадку, є ступені близькості текстів, місцезнаходження заявок, збіжність їх атрибутів і т. д.

Принцип роботи моделі зображено на рис. 3.4.

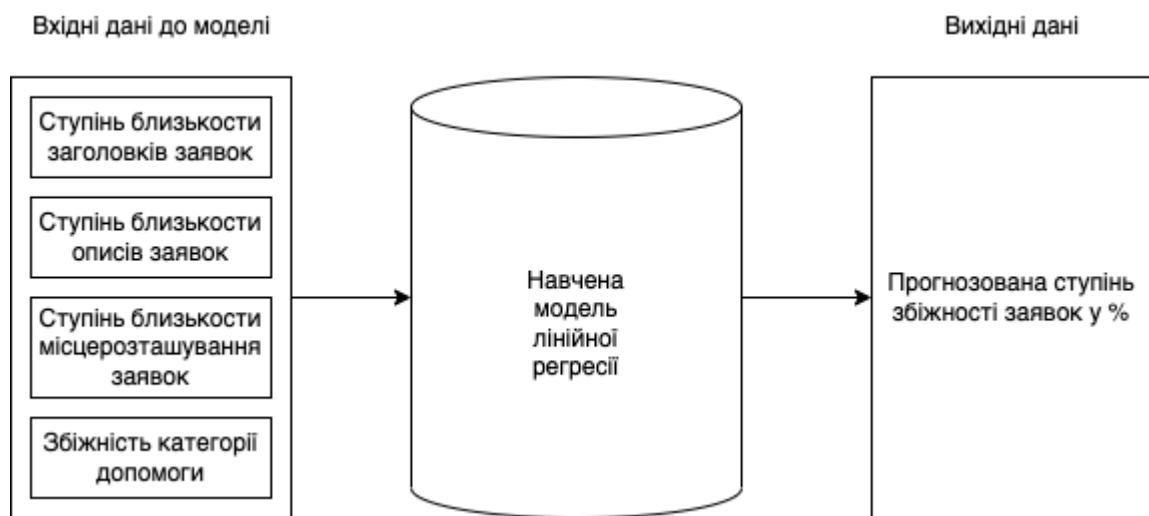


Рисунок 3.4 – Принцип роботи інкапсульованої моделі

Після навчання моделі лінійної регресії, задля того аби мати можливість її заміни в майбутньому без переписування коду вебзастосунку, було прийнято рішення про збереження навченої моделі в файл. Для цієї мети було використано піклінг (англ. “pickling”) – вбудований інструмент мови програмування Python, що дозволяє серіалізувати і десереалізувати об’єкти в байтовий формат, придатний для збереження в пам’яті або у файл. Отриманий файл “model.pickle” зберігається поруч з виконуваним вебзастосунком і зчитується кожен раз при запуску програми.

3.2.2 Основні рішення щодо реалізації вебзастосунку

Фреймворк Django надає можливість розбиття застосунків на окремі структурні частини, що мають назву Django Applications та реалізують у собі архітектуру MVC. Нижче наведено перелік виділених частин:

- `users` – призначений для роботи з системою аутентифікації користувачів у застосунку;
- `coordinating` – призначений для управління заявками (запитами, пропозиціями) користувачів;
- `recommender` – відповідає за генерацію і отримання користувацьких рекомендацій;
- `geo` – відповідає за роботу з локаціями (збереження списку міст і знаходження відстаней між містами).

Діаграма залежностей зазначених вище модулів системи наведена на рис. 3.5.

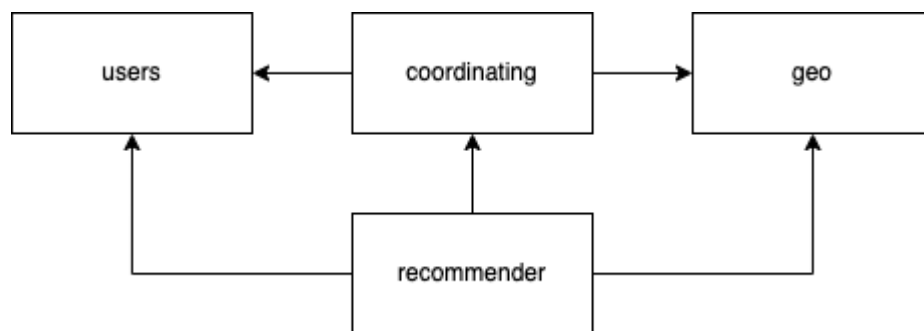


Рисунок 3.5 – Діаграма залежностей модулів вебзастосунку

Як бачимо, імплементований поділ на модулі є вдалим, оскільки в застосунку відсутні циклічні залежності між модулями, що спрощує тестування і робить можливою їх більш незалежну, чи то ізольовану, розробку.

Для автоматичної генерації документації з роботи з серверною частиною застосунку у форматі Swagger використовується бібліотека DRF-YASG (Django REST Framework - Yet Another Swagger Generator).

З метою автоматизації процесу тестування було використано бібліотеку Factory Boy для написання фабрик об'єктів Django-моделей (представлень даних з таблиць в БД). Це дозволило уникнути постійного повторення складної логіки створення тестових об'єктів під час перевірки роботи програми.

Оскільки фреймворк Django не надає інструментарію для роботи з номерами мобільних телефонів, під час реалізації серверної частини було використано бібліотеку Django PhoneNumber Field, що додає можливості валідації і зберігання номерів телефону у базі даних в зручному вигляді.

Для налаштування прав доступу до окремих компонентів системи було використано бібліотеку Django Role Permissions, що дозволяє налаштовувати структуру ролей з різними правами доступу для користувачів програми.

Реалізація стандартних CRUD-операцій була виконана з використанням шаблонних (через використання патерну програмування “Шаблон”) класів Django і Django REST Framework, а саме класів Model для створення, зберігання і редагування таблиць у базі даних, ModelSerializer для серіалізації об'єктів моделі у текстовий формат JSON, GenericAPIView для обробки запитів на виконання CRUD-операцій до серверу.

3.2.3 Основні рішення щодо реалізації виконувача періодичних і фонових задач

Як вже зазначалось на етапі проєктування, виконувач періодичних і фонових задач було реалізовано з використанням фреймворку мовою програмування Python – Celery. Такий вибір було зроблено через популярність даного технічного рішення, наявність об'ємної документації по

розгортанню та налаштуванню системи обробки періодичних і фонових задач, а також через наявність великої кількості готової технічних рішень з інтеграції Celery з іншими популярними фреймворка і бібліотеками, а також сторонніми сервісами [24]. Діаграма взаємодії компонентів розробленого виконувача періодичних і фонових задач зображена на рис. 3.6.

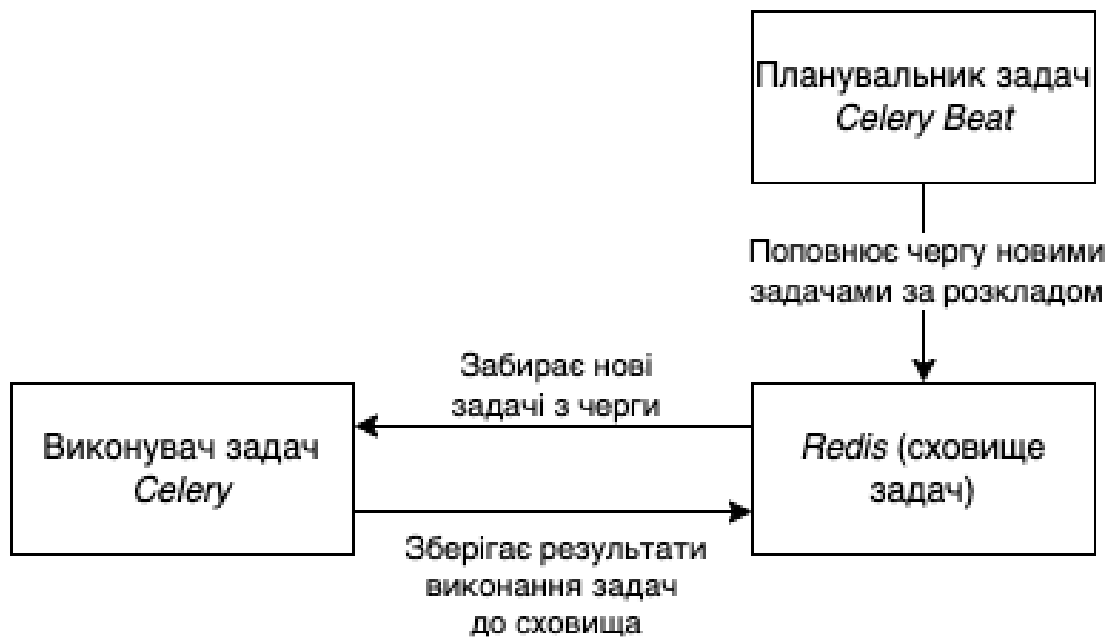


Рисунок 3.6 – Діаграма взаємодії компонентів виконувача задач

В якості інструменту для збереження даних про виконання задач, а також сховища результатів роботи задач в Celery було обрано сховище типу ключ-значення (англ. “key-value”) Redis. Він є NoSQL базою даних, оптимізованою для швидкого вибору даних за заданим ключем. Крім того, Redis підтримує різні типи даних, в тому числі збереження JSON-сумісних об’єктів та масивів даних [25].

Для організації черги задач було також використано вже наявне сховище Redis. Дане технічне рішення було продиктовано тим, що Redis вже використовується для зберігання результатів роботи задач, а отже не потребує додаткового встановлення і довгого налаштування для використання у якості черги.

Для організації розкладу виконання задач було використано модуль фреймворку Celery – Celery Beat. Він дозволяє задати розклад виконання Celery-задач в коді і своєчасно “підкладає” описані задачі в чергу, що вона обробляється виконувачем фонових задач Celery.

Аби зберігати інформацію про те, які задачі були вже покладені в чергу задач модулем Celery Beat, а які ще тільки чекають на то, було використано бібліотеку-інтеграцію з Django ORM – django-celery-beat. Замість сховища у вигляді файлу celerybeat-schedule, що використовується за замовчуванням модулем Celery Beat, django-celery-beat створює додаткову таблицю в базі даних для зберігання того, що є більш оптимальним під час вибірок.

3.3 Підсумки опису програми

В ході виконання етапу реалізації програми було імплементовано визначені раніше компоненти програми. Для спрощення процесу розробки, а також зменшення обсягів тестування, були використані наявні готові рішення у вигляді бібліотек для роботи з фреймворком Django, а також бібліотеки, що надають програмний інтерфейс для використання найпопулярніших ОПМ-алгоритмів. Зазначені технічні рішення були описані в п. 3.2.

По завершенню етапу реалізації отримано програмне забезпечення, що відповідає всім функціональним і не функціональним вимогам, що висуваються до нього згідно зі складеним раніше технічним завданням. Розроблений програмний продукт відповідає наявним сьогодні тенденціям розробки вебзастосунків і може бути легко модифікований в майбутньому.

4 ТЕСТУВАННЯ І РЕЗУЛЬТАТИ

В цьому розділі проводиться опис обраних методологій тестування програмного забезпечення, а також результати проведеного тестування, перевірки програмного забезпечення на відповідність поставленим задачам у технічному завданні до проекту.

4.1 Опис методики проведення тестування

4.1.1 Тестування вебзастосунку

Під час розробки програмного забезпечення було дотримано методики написання коду через тестування (англ. “Test Driven Development”, TDD). Даний підхід передбачає написання тестових методів спершу, а вже потім реалізацію тестуємих методів чи то функцій в коді. Це дозволяє розробнику підбирати більш зручний інтерфейс виклику методів та функцій, що він проєктує, а також писати код, що є простіший для тестування, а отже є менш зв’язним з іншими компонентами системи [26]. Використовуючи TDD-підхід було написано 69 автоматизованих інтеграційних та модульних тестів до проєкту.

Модульні, або ж, юніт-тести ставлять собі на меті перевірку коректності виконання окремих юнітів – найменших функціональних частин програми, наприклад, функцій чи методів класів. Виконання юніт-тестів зазвичай ізольовані від зовнішнього середовища, а всі виклики сторонніх компонентів у них підміняються тестовими об’єктами, що повторюють поведінку реальних. Такі тестові об’єкти називаються моками (від англ. “mock”).

Інтеграційні тести ж ставлять собі на меті перевірку коректності взаємодії різних компонентів системи між собою, правильність обміну даними між ними.

Обраний для написання вебфреймворк Django має власний інструментарій для написання інтеграційних тестів, що імітують запити користувача. Завдяки цьому, можна зробити перевірки повного циклу запит-відповідь. Таки перевірки носять назву наскрізних тестів (англ. “end-to-end testing”).

Крім того, Django надає можливість написання інтеграційних тестів, не підміняючи запити до бази даних. Наявна в Django функціональність створює окрему, тестову, базу даних, виконує в ній всі міграції схеми даних, а також очищає дані під час виконання кожного окремого тестового методу. Це дозволяє проводити тести ізольовано один від одного.

Для полегшення процесу написання тестів було прийнято також рішення про використання популярного тестового фреймворку, написаного мовою програмування Python – `pytest`, що дозволяє писати короткі тестові методи з перевикористовуваними фікстурами – наборами тестових даних, написаних користувачем. Крім того, стандартна функціональність `pytest` може бути доповнена за допомогою використання плагінів, що легко встановлюються. Зокрема, під час розробки даного ПЗ було використано плагіни для написання тестів до фреймворку Django (`pytest-django`), а також плагін для збору метрик покриття коду тестами (`pytest-cov`).

Під час написання тестів було також використано методику написання тестів AAA (англ. “Arrange, Act, Assert”). За таким підходом, кожен тестовий метод умовно розділяється на три частини, першу з яких становить підготовка необхідних вхідних даних для модулю, що тестується, далі безпосередній виклик тестуємої функції або методу, після чого відбувається перевірка коректності повертаємого результату. У випадку з написанням юніт-тестів, в кінці також відбувається перевірка коректних викликів об’єктів, що були замінені моками.

4.1.2 Оцінювання рекомендаційної моделі

Тестування, тобто оцінка якості роботи рекомендаційної моделі, проводиться відповідно до методики, описаної в п. 3.1.1.3, а також з урахуванням деталей і окремих технічних рішень, описаних в п. 3.2.1.2.

Для автоматизації процесу оцінювання моделі, було написано Python-скрипт, що автоматично розраховує метрику оцінювання якості моделі і виводить її в потік стандартного виводу (stdout).

4.2 Результати тестування

4.2.1 Результати тестування вебзастосунку

Використовуючи наявні в Django, а також в Django REST Framework тестові модулі, а також інструментарій pytest було реалізовано 69 автоматизованих тестів для перевірки коректності роботи програми.

Метрики покриття коду тестами були налаштовані таким чином, аби в підсумкову статистику включались лише власноруч написані модулі, виключаючи всі бібліотеки-залежності, або конфігураційні файли, згенеровані вебфреймворком Django.

Результат запуску автоматизованих тестів показано на рис. 4.1. Як видно з консольного виводу, під час запуску всіх автоматизованих тестів не було виявлено жодного дефекту виконання програми. Крім того, відсоток покриття коду тестами складає 92%.

Написані автоматизовані тести покривають всю основну функціональність вебзастосунку, а також повторюють всі можливі сценарії використання відкритого програмного інтерфейсу клієнтським застосунком.

```

user_proposals/views.py      40      2    95%
user_requests/admin.py       20      3    85%
user_requests/serializers.py 42      5    88%
user_requests/services.py    38     10    74%
user_requests/views.py       40      2    95%
users/managers.py            19     13    32%
users/models.py              20      1    95%
users/serializers.py          38      1    97%
users/services/email_verify.py 37     20    46%
users/services/user.py        17      9    47%
users/views.py               46      6    87%
-----
TOTAL                        1148     89    92%

56 files skipped due to complete coverage.

===== 69 passed, 248 warnings in 1.93s

```

Рисунок 4.1 – Результати виконання автоматизованих тестів разом з результатами покриття коду тестами

4.2.2 Результати виконання вебзастосунку

Для демонстрації результатів роботи вебзастосунку, було створено тестову заявку типу “запит” і згенеровано для неї 5 рекомендованих пропозицій. Значення атрибутів заявки наведено у табл. 4.1.

Таблиця 4.1 – Значення атрибутів тестової заявки

Назва атрибуту	Значення
Заголовок	Необхідні ліки (Варфарин Нікомед)
Опис	Терміново необхідні пігулки для серця (Варфарин Нікомед 2,5мг №100 або аналогічні), від 3-х упаковок
Чи є терміновою?	Так
Місто	Запоріжжя
Категорія	Гуманітарна допомога

Далі, користуючись відкритим програмним інтерфейсом програми, а також HTTP-клієнтом Postman було виконано тестовий HTTP-запит для отримання списку рекомендованих позицій (рис. 4.2).

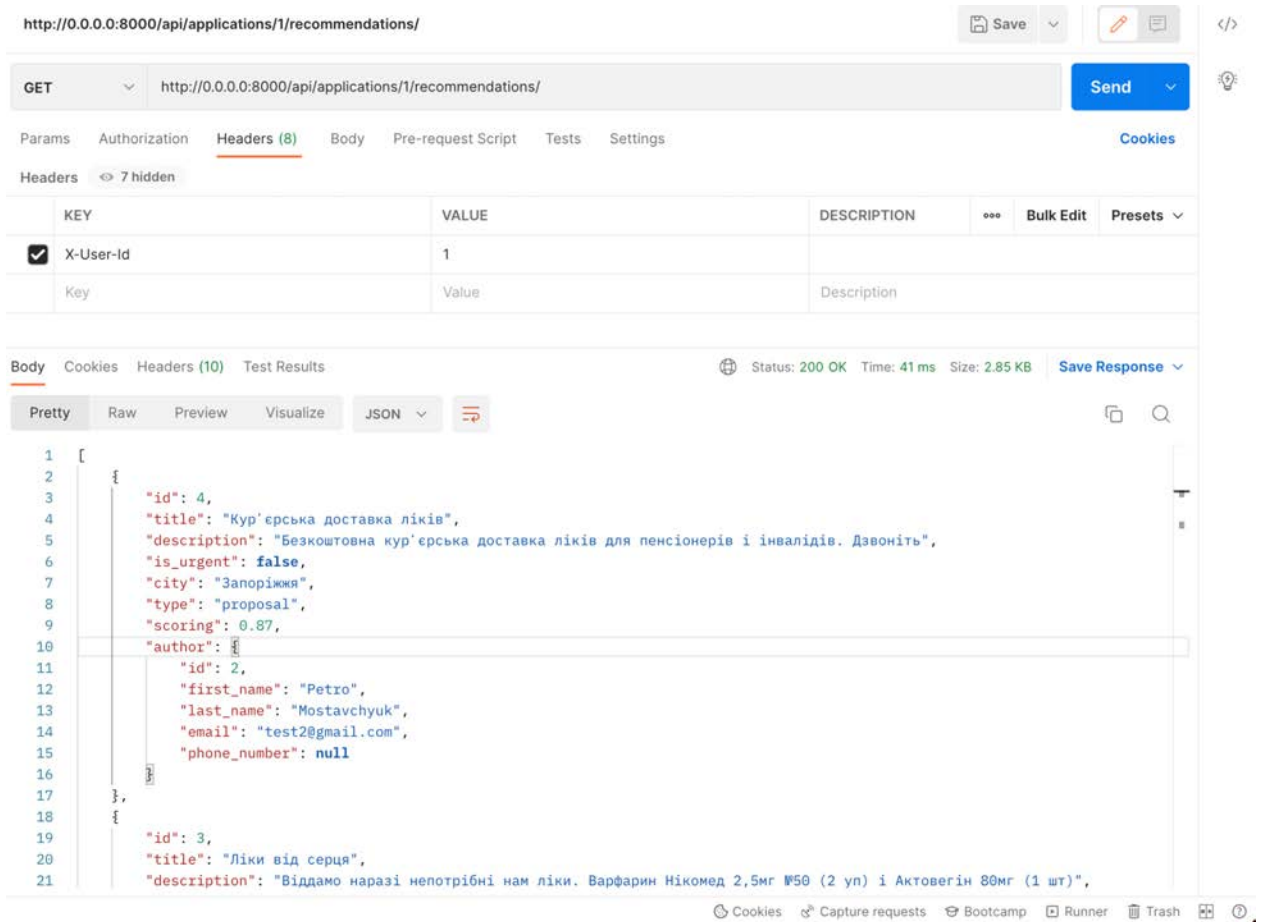


Рисунок 4.2 – Відправлення тестового HTTP-запиту на отримання списку рекомендованих пропозицій

Повний текст тіла HTTP-відповіді сервера наведено нижче. Результати відсортовані за значенням атрибуту `scoring`, що показує розрахований програмою ступінь схожості між рекомендованою і вхідною заявками.

```
[
  {
    "id": 4,
    "title": "Кур'єрська доставка ліків",
```

```

        "description": "Безкоштовна кур'єрська доставка ліків для
пенсіонерів і інвалідів. Дзвоніть",
        "is_urgent": false,
        "city": "Запоріжжя",
        "type": "proposal",
        "scoring": 0.87,
        "author": {
            "id": 2,
            "first_name": "Petro",
            "last_name": "Mostavchyuk",
            "email": "test2@gmail.com",
            "phone_number": null
        }
    },
    {
        "id": 3,
        "title": "Ліки від серця",
        "description": "Віддамо наразі непотрібні нам ліки. Варфарин
Нікомед 2,5мг №50 (2 уп) і Актовегін 80мг (1 шт)",
        "is_urgent": false,
        "city": "Вільнянськ",
        "type": "proposal",
        "scoring": 0.85,
        "author": {
            "id": 2,
            "first_name": "Petro",
            "last_name": "Mostavchyuk",
            "email": "test2@gmail.com",
            "phone_number": null
        }
    },
    {
        "id": 6,
        "title": "ГО \"280 днів\"",
        "description": "Громадська організація \"280 днів\" займається
безкоштовним наданням медичної допомоги, спорядженням необхідних
ліків та медичними консультаціями осіб, що постраждали внаслідок
бойових дій, а також всім категоріям пільговиків. Дзвоніть або
звертайтеся за допомогою по вул. Запорізькій 12!",

```

```

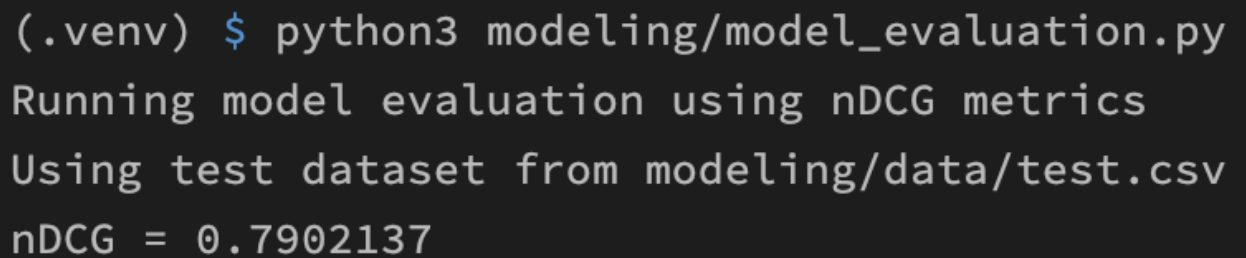
    "is_urgent": false,
    "city": "Запоріжжя",
    "type": "proposal",
    "scoring": 0.83,
    "author": {
      "id": 2,
      "first_name": "Petro",
      "last_name": "Mostavchyuk",
      "email": "test2@gmail.com",
      "phone_number": null
    }
  },
  {
    "id": 2,
    "title": "Ліки від серця",
    "description": "Організація \"HelpMed\" займається безкоштовною
роздачею ліків різним категоріям громадян, що вони їх потребують.
Доступні Варфарин Нікомед, Трипліksam, Панангін, Сіднофарм тощо",
    "is_urgent": false,
    "city": "Запоріжжя",
    "type": "proposal",
    "scoring": 0.79,
    "author": {
      "id": 2,
      "first_name": "Petro",
      "last_name": "Mostavchyuk",
      "email": "test2@gmail.com",
      "phone_number": null
    }
  },
  {
    "id": 5,
    "title": "Отримання ліків",
    "description": "Допомога з отриманням рідкісних лікарських
засобів для серця. Дзвоніть або пишіть у Viber",
    "is_urgent": false,
    "city": "Кушугум",
    "type": "proposal",
    "scoring": 0.69,

```

```
"author": {  
  "id": 2,  
  "first_name": "Petro",  
  "last_name": "Mostavchyuk",  
  "email": "test2@gmail.com",  
  "phone_number": null  
}  
}  
]
```

4.2.3 Оцінювання рекомендаційної моделі

Результати запуску скрипта, розробленого для автоматичного розрахунку метрики якості рекомендаційної моделі наведено на рис. 4.3.



```
(.venv) $ python3 modeling/model_evaluation.py  
Running model evaluation using nDCG metrics  
Using test dataset from modeling/data/test.csv  
nDCG = 0.7902137
```

Рисунок 4.3 – Результати оцінювання якості моделі

5 КЕРІВНИЦТВО ПРОГРАМІСТА

В даному розділі приводиться керівництво до роботи програміста, що має задачу інтеграції з розробленим програмним забезпеченням, або його подальшої модифікації. Тут наводиться опис випадків застосування програми, її характеристик, а також варіантів поліпшення, доповнення чи виправлення програмного коду. Керівництво програміста оформлюється відповідно до ГОСТ 19.504–79.

5.1 Призначення і умови застосування програми

Розроблене програмне забезпечення призначено для використання як компонент загальної системи “Волонтерський координатор” та надає можливість переглядати складені рекомендації до запитів і пропозицій про надання допомоги різним категоріям користувачів.

5.2 Характеристика програми

Програмний продукт складається з декількох компонентів: це, передусім, головний вебзастосунок з розробленою рекомендаційною підсистемою, а також виконувач періодичних і фонових задач, що він забезпечує постійну генерацію нових рекомендацій для користувачів.

Вебзастосунок являє собою WSGI-застосунок, написаний мовою програмування Python з використанням вебфреймворку Django та його розширення Django REST Framework. Розроблене ПЗ надає відкритий програмний інтерфейс, що він може бути легко інтегрований до інших компонентів системи, та надає можливість отримання списку рекомендованих запитів до пропозиції та навпаки.

Рекомендаційна система функціонує за принципом генерації рекомендацій, заснованих на змісті. Для кожної заявки система зберігає до

п'яти заявок іншого типу, що є найбільш схожими до заявки, що розглядається. Список рекомендацій для користувача оновлюється кожного ранку, аби враховувати новостворені заявки, а також зміни статусу актуальності заявок з попередніх рекомендацій.

Виконувач періодичних і фонових задач написаний мовою програмування Python з використанням фреймворку Celery, а також сховищем даних типу ключ-значення Redis для організації черги задач, а також зберігання результатів їх роботи.

5.3 Звертання до програми

Звертання до серверної частини програми виконуються через модуль “manage.py”. Всі команди, що мають бути виконані Django-застосунком, спочатку проходять крізь чей менеджер команд. Крім того, усі команди повинні бути виконані у середовищі системи контролю залежностей Poetry (тобто починатися з “poetry run”), а також у контейнерах Docker.

Для тестування програми необхідно виконати наступні команди:

- `docker-compose build;`
- `docker-compose run app pytest.`

Для розгортання програми необхідно виконати наступну команду:
`docker-compose up -d --build.`

Після розгортання програми почне свою роботу вебсервер. Для взаємодії з REST API програми необхідно відправляти клієнтські HTTP-запити на URL застосунку (`http://0.0.0.0:8000` за замовчуванням).

Для перегляду документації розробленого API, достатньо перейти за відносним посиланням `/api/docs/swagger/` (`https://0.0.0.0:8000/api/docs/swagger` за замовчуванням) та, слідує підказкам документації, або користуючись інтерактивним вебінтерфейсом Swagger (рис. , розробник має можливість виконувати всі доступні методи API).

5.4 Вхідні і вихідні дані програми

Вхідними і вихідними даними програми у вузькому сенсі є HTTP-запити і відповіді на них від вебзастосунку до клієнтської частини, Вхідні і вихідні дані програми надаються виключно у текстовому форматі JSON.

Вхідними даними у широкому сенсі є дані заповнення форм: інформація, що стосується заявок (запитів, пропозицій), їх авторів тощо.

Вихідними даними є згенеровані рекомендаційним субмодулем персоналізовані рекомендації відповідно до кожної відкритої в системі заявці.

5.5 Повідомлення програмісту

Розроблений вебзастосунок під час відправлення HTTP-запитів до нього повертає відповіді (повідомлення) відповідно до стандартів HTTP-протоколу. У тілі відповіді, як правило, знаходяться текстові дані у форматі JSON.

Приклад позитивної відповіді вебзастосунку з HTTP-кодом 200 наведено нижче.

```
[
  {
    "id": 1,
    "title": "Тестовий заголовок запиту на надання допомоги",
    "relevance": 0.96
  },
  {
    "id": 2,
    "title": "Інший тестовий заголовок запиту на надання допомоги",
    "relevance": 0.94
  }
]
```

Приклад негативної відповіді вебзастосунку з HTTP-кодом 400 наведено нижче.

```
{
  "title": [
    "Це поле обов'язкове для заповнення."
  ],
  "non_field_errors": [
    "Ви не можете додавати більше нових заявок."
  ]
}
```

Приклад негативної відповіді вебзастосунку з HTTP-кодом 401 наведено нижче.

```
{
  "detail": "Unauthenticated."
}
```

6 КЕРІВНИЦТВО СИСТЕМНОГО ПРОГРАМІСТА

В даному розділі приводиться керівництво до роботи системного програміста, що має задачу підтримки чи відновлення працездатності програмного забезпечення. Тут наводиться опис випадків застосування програми, її характеристики, а також можливі проблеми та варіанти їх вирішення. Керівництво системного програміста оформлюється відповідно до ГОСТ 19.504-79.

6.1 Загальні відомості про програму

Розроблене програмне забезпечення призначено для використання як компонент загальної системи “Волонтерський координатор” та надає можливість переглядати складені рекомендації до запитів і пропозицій про надання допомоги різним категоріям користувачів.

Розроблене програмне забезпечення виконується на сервері ізольовано від зовнішнього середовища, використовуючи засіб контейнеризації Docker.

6.2 Структура програми

Структурно, програмне забезпечення складається з декількох компонентів, кожен з яких повинен запускатись в своєму окремому процесі, а отже потребує свого окремого запущеного Docker-контейнеру:

- самого WSGI-серверу, що обслуговує написаний вебзастосунок з відкритим програмним інтерфейсом, який обробляє вхідні запити клієнтської частини;
- виконувача періодичних і фонових задач, написаного з використанням вебфреймворку Celery, що відповідає за генерацію нових рекомендацій і видалення вже неактуальних;

- бази даних Postgres, що зберігає записи про залишені користувачами заявки, безпосередньо користувачів, а також згенеровані для них рекомендації;
- сховища даних Redis для зберігання результатів роботи виконувача періодичних і фонових задач, а також організації задач в чергу.

Всі вказані компоненти системи комунікують між собою в приватній Docker-мережі. Лише вебсервер WSGI є публічним і може бути доступний використовуючи HTTP-запити по 8000 порту.

6.3 Налагодження програми

Розроблене програмне забезпечення має власну систему логування, що пише системні повідомлення в системний потік виводу (stdout) в середині Docker контейнеру.

Для перегляду логів будь-якого компоненту системи необхідно виконати команду `docker logs -f <назва сервісу>`, де назва сервісу – це визначена в файлі `docker-compose.yml` назва компоненту системи, одна з “app”, “celery”, “postgres”, “redis”.

Якщо повідомлення є недостатньо інформативними, то для відлагодження програми можна використовувати інтерактивний Python-shell в середині контейнера. Для цього необхідно виконати команду `docker exec -it app python3 manage.py shell -i python`.

6.4 Перевірка програми

Перевірити програму на те, чи знаходиться вона в стані роботи, можна зробити використовуючи засоби Docker. Для цього, треба або переглянути список запущених контейнерів, виконавши команду `docker ps`, або переглянути використані ресурси контейнерів `docker stats`.

6.5 Повідомлення системному програмісту

Повідомлення системному адміністратору подаються в звичному форматі для логів: “<Рівень повідомлення> | <Дата повідомлення> | <Повідомлення>”, де рівень повідомлення – значення одне з “DEBUG”, “INFO”, “WARNING”, “ERROR”, “CRITICAL”, що позначає рівень важливості отриманого повідомлення і передбачає різну реакцію на їх отримання.

Приклад повідомлення з логів наведено нижче.

```
INFO | 2022-11-27 12:53:03 | Successfully generated 5 new  
recommendations for application ID 5
```

ВИСНОВКИ

В ході виконання роботи було проведено аналіз предметної області, що включав в себе вивчення вже наявних відомостей про предметну область з розроблюваної системи “Волонтерський координатор”, огляд систем-аналогів, що ставив собі на меті вивчити можливості покращення системи “Волонтерський координатор”, а також можливі шляхи побудови рекомендаційної системи для нього. За результатами проведеного аналізу було складено документ – технічне завдання до проєкту, що включає функціональні та нефункціональні вимоги до розроблюваного програмного забезпечення, зокрема, вимоги до організації отримання користувачем персоналізованих рекомендацій відповідно до залишених ним в системі заявок і пропозицій про надання допомоги.

Враховуючи перелічені у технічному завданні до проєкту вимоги, було також проведено етап проєктування програмного забезпечення. В ході нього були визначені основні компоненти майбутньої системи, розроблена високорівнева архітектура, а також підібрані інструменти для розробки кожного компоненту системи, а саме, вебзастосунку, бази даних, виконувача періодичних і фонових задач, а також допоміжних інструментів, що вони будуть використані під час розробки.

Спроектоване програмне забезпечення відповідає всім сучасним стандартам проєктування, є легким для інтеграції з іншими частинами системи “Волонтерський координатор”, а також мусить бути простим у розробці і тестуванні завдяки автономності своїх компонентів в середині.

Третім етапом стала безпосередня реалізація проєкту: імплементація визначеної архітектури з заздалегідь обраними інструментами. На цьому етапі були прийняті ще декілька важливих рішень щодо архітектури пакетів нижчих рівнів, що досягло не порушити початкової структури проєкту, а також легко перейти до тестування програми.

Заключними етапами стали тестування і введення проекту до експлуатації. Завдяки використанню TDD підходу вдалося легко досягти майже повного покриття кодової бази модульними тестами, а виділення сценаріїв використання на процесі розробки полегшили процес створення та виконання тест-кейсів. Написана в процесі розробки документація щодо виконання програми, а також використання найновіших інструментів автоматизації процесу розгортання (таких як засіб контейнеризації Docker, інструмент для управління залежностями проекту Poetry та інші), дозволили легко ввести готовий програмний продукт в експлуатацію.

Реалізоване програмне забезпечення відповідає всім вимогам, поставленим в технічному завданні до проекту. Отримана програма є стійкою до змін і передбачає можливості свого подальшого розширення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Міхеєва, О. Актуальність волонтерської діяльності у період війни / О. Міхеєва // Післявоєнний світ : люди, проблеми, цінності : зб. матеріалів міжнародної науково-практичної інтернет-конференції. Київ, 15 квітня 2022 р. / Редколегія: В. П. Андрущенко (голова), І. Г. Ветров (заст. голови, відп. ред.), Г. М. Торбін (заст. голови), О. В. Потильчак (упорядн.). – Київ : Вид-во НПУ імені М. П. Драгоманова, 2022. – С. 91-92.
2. Як ми заснували волонтерку та автоматизували її роботу | DOU [Електрон. ресурс] / Режим доступу: <https://dou.ua/forums/topic/38094/>.
3. Українська Волонтерська Служба [Електрон. ресурс] / Режим доступу: <https://volunteer.country/>.
4. UA Helpers база волонтерської допомоги та запитів [Електрон. ресурс] / Режим доступу: <https://uahelpers.com/>.
5. Волонтерська платформа “Волонтер.орг” [Електрон. ресурс] / Режим доступу: <https://volonter.org/>.
6. Berson, A. Client/Server Architecture / A. Berson – New York, McGraw-Hill; Subsequent edition, 1996. – 569 p.
7. What is API? | Amazon Web Services [Electronic resource]. – Access mode: <https://aws.amazon.com/ru/what-is/api/>.
8. Fielding, R. T. Architectural Styles and the Design of Network-based Software Architectures / R. T. Fielding – Irvine, University of California, 2000. – 180 p.
9. Reenskaug, T. The Model-View-Controller (MVC) Its Past and Present / T. Reenskaug – Oslo, University of Oslo, 2003. – 197 p.
10. Chen, P. The entity-relationship model—toward a unified view of data / P. Chen – New York, Association for Computer Machinery, 1976. – 28p.
11. What Are Wireframes? Balsamiq [Electronic resource]. – Access mode: <https://balsamiq.com/learn/articles/what-are-wireframes/>.

12. What is the best programming language for Machine Learning? [Electronic resource]. – Access mode: <https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7>.

13. Swaroop, C. H. A Byte of Python / C. H. Swaroop – Ebshelf Inc., 2013 – 189 p.

14. Django Advantages and Disadvantages [Electronic resource]. – Access mode: <https://data-flair.training/blogs/django-advantages-and-disadvantages/>.

15. Top 9 Python Libraries for Machine Learning in 2023 [Електронний ресурс] / Режим доступу – <https://www.upgrad.com/blog/top-python-libraries-for-machine-learning/>.

16. Types of Recommender Systems & Their Use Cases – Medium [Electronic resource]. – Access mode: <https://medium.com/mllearning-ai/what-are-the-types-of-recommendation-systems-3487cbafa7c9>.

17. What is Natural Language Processing? – [Electronic resource]. – Access mode: <https://www.ibm.com/cloud/learn/natural-language-processing>.

18. Глибовець, А. М. Алгоритм токенизації та стемінгу для текстів українською мовою / А. М. Глибовець, В.В. Точицький – Київ, Національний університет "Києво-Могилянська академія", 2017. – 3 с.

19. Stemming and lemmatization - Stanford NLP Group [Electronic resource]. – Access mode: <https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html>.

20. Вкладання слів – Вікіпедія [Електрон. ресурс] / Режим доступу: https://uk.wikipedia.org/wiki/%D0%92%D0%BA%D0%BB%D0%B0%D0%B4%D0%B0%D0%BD%D0%BD%D1%8F_%D1%81%D0%BB%D1%96%D0%B2.

21. Word2Vec – Вікіпедія [Електрон ресурс] / Режим доступу: <https://uk.wikipedia.org/wiki/Word2vec#%D0%9F%D1%96%D0%B4%D1%85%D1%96%D0%B4>.

22.Ukraine Cities Database – Simple Maps [Electronic resource]. – Access mode: <https://simplemaps.com/data/ua-cities>.

23.Discounted Cumulative Gain – Wikipedia [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Discounted_cumulative_gain.

24.Celery – Distributed Task Queue [Electronic resource]. – Access mode: <https://docs.celeryq.dev/en/stable/>.

25. Redis: In-memory data store. How it works and why you would need it – Amazon Web Services [Electronic resource]. – Access mode: <https://aws.amazon.com/redis/>.

26.Beck, K. Test Driven Development: By Example / K. Beck – Addison-Wesley Professional; 1st edition, 2002. – 240 p.

ДОДАТОК А
Технічне завдання

A.1 Вступ

Рекомендаційна система для платформи “Волонтерський координатор”, де різні користувачі мають можливість викладати свої запити та пропозиції про допомогу, слугує для підбору персоналізованих рекомендацій, засновуючись на історії активності користувачів на платформі.

A.2 Підстави для розробки

Підставою для розробки документу є дипломна магістерська кваліфікаційна робота за темою “Дослідження та програмна реалізація рекомендаційної системи для волонтерського координатора”, яку було затверджено наказом №363 від 4 листопада 2022 р. Національним Університетом “Запорізька політехніка”.

A.3 Вимоги до програми або програмного продукту

Вимоги до розробки рекомендаційної системи були складені враховуючи наявну модель предметної області на платформі “Волонтерський координатор”, а також поради до її покращення і побудови рекомендаційної системи, викладені в результаті аналізу систем-аналогів.

A.3.1 Вимоги до функціональних характеристик

Головною особливістю, що буде надана користувачам з імплементацією рекомендаційної системи буде можливість перегляду списку відповідних до їх поданою заявки запитів чи пропозицій про надання допомоги.

Остаточні вимоги до реалізації рекомендаційної системи з клієнтської перспективи є складені користувацькі історії (від англ. “user stories”):

- я, як отримувач допомоги, хочу мати можливість переглядати список найбільш релевантних пропозицій про надання допомоги відповідно до поданого мною запиту на надання допомоги;

- я, як волонтер, хочу мати можливість переглядати список найбільш релевантних запитів на надання допомоги відповідно до поданої мною пропозиції на надання допомоги.

Ступінь релевантності запитів до пропозицій і навпаки має бути визначений за наступними критеріями:

- за схожістю детального опису запиту і пропозиції;
- за схожістю короткого заголовку запиту і пропозиції;
- за відповідністю місце розташування (локації) запиту і пропозиції;
- за відповідністю категорії запиту і пропозиції;
- за ступенем терміновості запиту і пропозиції;
- за збіжністю атрибутів, специфічних для кожної конкретної категорії, що присвоєно запиту і пропозиції.

Вплив кожного з критеріїв, а також загальна доцільність їх використання, має бути визначені під час дослідження і імплементації рекомендаційної системи, враховуючи існуючі історичні дані закриття запитів і пропозицій про надання допомоги на платформі “Волонтерський координатор”.

А.3.2 Вимоги до надійності

Розроблене ПЗ має адекватно реагувати на відсутність доступу до сторонніх сервісів (підключення до баз даних, використання сторонніх бібліотек, що використовують системні бібліотеки і інструменти) та у такому разі повідомляти користувача про виниклі помилки.

Програмне забезпечення має швидко відповідати на запити клієнтської частини (до 1.0 с), відповіді серверу мають бути оптимізовані і не містити в собі зайвих даних.

Серед поставлених вимог також є відповідність сучасним стандартам безпеки, захищеність від всіх найчастіших типів атак: міжсайтової підробки запиту (CSRF, від англ. “Cross-Site Request Forgery”), міжсайтового скриптингу (XSS, від англ. “Cross-Site Scripting”), зловмисне впровадження JavaScript чи то SQL (англ. “Structured Query Language”) коду (англ. “Code injection”). Отримане програмне забезпечення також має унеможливлювати несанкціонований доступ до неї, а також отримання персональних даних користувачів.

Програма також повинна проводити валідацію усіх вхідних даних, що потрапляють в систему з клієнтської сторони та в разі невідповідності їх визначеним специфікаціям повідомляти про це користувачу у зрозумілій для нього формі.

А.3.3 Умови експлуатації

Умови експлуатації розроблюваного програмного продукту відповідають вимогам до експлуатації технічних засобів (комп'ютеру, серверу Linux, Web-браузеру, що використовується користувачем тощо).

А.3.4 Вимоги до складу та параметрів технічних засобів

Розроблене ПЗ повинно бути сумісне та мати можливість виконуватись на операційній системі (ОС) Linux або аналогічних ОС, заснованих на Unix. Допускається використання засобів віртуалізації, сумісних з ОС Linux.

A.3.5 Вимоги до інформаційної та програмної сумісності

Клієнтська і серверна частина мають комунікувати між собою за допомогою текстового формату JSON, в якому будуть надані необхідні дані для зчитування.

A.3.6 Вимоги до програмної документації

Разом з розробкою програмного забезпечення повинні бути складені наступні документи:

- керівництво програміста;
- керівництво системного програміста.

Первинний код програми повинен документуватися у процесі розробки з використанням засобів, що надає та чи інша мова програмування.

A.3.7 Стадії та етапи розробки

Стадії і етапи розробки програмного забезпечення відповідають етапам життєвого циклу програмного забезпечення:

- збір і аналіз вимог;
- проєктування;
- реалізація;
- тестування;
- введення в експлуатацію.

A.3.8 Порядок контролю та приймання

Порядок контролю та приймання відповідає затвердженому порядку контролю та приймання магістерських дипломних робіт за спеціальністю “Інженерія програмного забезпечення”.

ДОДАТОК Б
Текст програми

Б.1 Код файла “app/config/settings.py”

```

import os

from pathlib import Path

import environ

from celery.schedules import crontab

# Build paths inside the project like this: BASE_DIR / "subdir".
BASE_DIR = Path(__file__).resolve().parent.parent

env = environ.Env()
env.read_env(os.path.join(BASE_DIR, ".env"))

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = env("DJANGO_SECRET_KEY")

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = env("DJANGO_DEBUG_MODE", cast=bool, default=False)

ALLOWED_HOSTS = ["*"]

# Application definition

INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",

    "rest_framework",
    "drf_yasg",

    "app.coordinating",
    "app.users",

```

```

    "app.recommender",
    "app.geo",
]

MIDDLEWARE = [
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
]

ROOT_URLCONF = "app.config.urls"

TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.DjangoTemplates",
        "DIRS": [],
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",
                "django.contrib.messages.context_processors.messages",
            ],
        },
    },
]

WSGI_APPLICATION = "app.config.wsgi.application"

# Database
# https://docs.djangoproject.com/en/4.1/ref/settings/#databases

DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql",
        "HOST": env("POSTGRES_HOST", default="0.0.0.0"),
        "NAME": env("POSTGRES_DB", default=""),
    }
}

```

```

        "USER": env("POSTGRES_USER", default=""),
        "PORT": env("POSTGRES_PORT", cast=int, default=5432,),
        "PASSWORD": env("POSTGRES_PASSWORD", default=""),
    }
}

# Password validation
# https://docs.djangoproject.com/en/4.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME":
        "django.contrib.auth.password_validation.UserAttributeSimilarityValidator",
    },
    {
        "NAME":
        "django.contrib.auth.password_validation.MinimumLengthValidator",
    },
    {
        "NAME":
        "django.contrib.auth.password_validation.CommonPasswordValidator",
    },
    {
        "NAME":
        "django.contrib.auth.password_validation.NumericPasswordValidator",
    },
]

# Internationalization
# https://docs.djangoproject.com/en/4.1/topics/i18n/

LANGUAGE_CODE = "en-us"

TIME_ZONE = "UTC"

USE_I18N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.1/howto/static-files/

```

```

STATIC_URL = "static/"

# Default primary key field type
# https://docs.djangoproject.com/en/4.1/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"

AUTHENTICATION_BACKENDS = [
    "app.users.authentication_backends.HeaderBasedBackend",
]
AUTH_USER_MODEL = "users.User"


# Custom settings
APPLICATION_EXPIRATION_DAYS = env(
    "APPLICATION_EXPIRATION_DAYS",
    cast=int,
    default=7,
)
AUTHENTICATION_HEADER_NAME = env(
    "AUTHENTICATION_HEADER_NAME",
    default="X-User-Id",
)


# Celery settings
CELERY_BROKER_URL = env("CELERY_BROKER_URL")
CELERY_RESULT_BACKEND = env("CELERY_RESULT_BACKEND")
CELERY_BEAT_SCHEDULER = "django_celery_beat.schedulers:DatabaseScheduler"
CELERY_ACCEPT_CONTENT = ["json", "pickle"]
CELERY_RESULT_ACCEPT_CONTENT = ["json", "pickle"]
CELERY_TASK_SERIALIZER = "json"
CELERY_RESULT_SERIALIZER = "json"
CELERY_EVENT_QUEUE_EXPIRES = 30
CELERY_TIMEZONE = "UTC"
CELERY_BEAT_SCHEDULE = {
    "generate-recommendations": {
        "task": "recommender.generate_recommendations",
        "schedule": crontab(hour=0, minute=10), # every day at 0:10 UTC
    },
    "clean-irrelevant-recommendations": {
        "task": "recommender.clean_irrelevant_recommendations",
        "schedule": crontab(hour=0, minute=0), # every day at 0:00 UTC
    },
}

```

```

    },
}

# Django REST Framework (DRF) settings
REST_FRAMEWORK = {
    "DEFAULT_AUTHENTICATION_CLASSES": [
        "rest_framework.authentication.BasicAuthentication",
        "rest_framework.authentication.SessionAuthentication",
        "app.users.authentication_backends.HeaderBasedAuthentication",
    ],
    "COERCE_DECIMAL_TO_STRING": False,
}

```

Б.2 Код файла “app/config/celery.py”

```

import os

from celery import Celery

# set the default Django settings module for the 'celery' program.
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "app.config.settings")

app = Celery("coordinator")

# Using a string here means the worker doesn't have to serialize
# the configuration object to child processes.
# - namespace='CELERY' means all celery-related configuration keys
#   should have a `CELERY_` prefix.
app.config_from_object("django.conf:settings", namespace="CELERY")

# Load task modules from all registered Django app configs.
app.autodiscover_tasks()

```

Б.3 Код файла “app/config/urls.py”

```

from django.contrib import admin
from django.urls import include, path, re_path
from drf_yasg import openapi
from drf_yasg.views import get_schema_view
from rest_framework import permissions

schema_view = get_schema_view(

```

```

openapi.Info(
    title="Recommender System for Volunteering Coordinator",
    default_version="v1",
    description=(
        "API endpoints for recommender system for volunteering coordinator"
    ),
    terms_of_service="https://www.google.com/policies/terms/",
    contact=openapi.Contact(email="diachkovspecial@gmail.com"),
    license=openapi.License(name="MIT License"),
),
public=True,
permission_classes=[permissions.AllowAny],
)

urlpatterns = [
    path("admin/", admin.site.urls),

    # API endpoints
    path("api/", include("app.recommender.urls")),

    # DRF YASG integration
    re_path(
        r"^api/docs/swagger(?:P<format>\.json|\.yaml)$",
        schema_view.without_ui(cache_timeout=0),
        name="schema-json",
    ),
    path(
        "api/docs/swagger/",
        schema_view.with_ui("swagger", cache_timeout=0),
        name="schema-swagger-ui",
    ),
    path(
        "api/redoc/",
        schema_view.with_ui("redoc", cache_timeout=0),
        name="schema-redoc",
    ),
]

```

Б.4 Код файла “app/coordinating/models.py”

```

from datetime import datetime, timedelta

from django.conf import settings
from django.db import models
from django.utils import timezone
from django.utils.translation import gettext_lazy as _
from model_utils.models import TimeStampedModel


def get_application_expire_time() -> datetime:
    """Application shall expire after configured days amount."""
    return timezone.now() + timedelta(
        days=settings.APPLICATION_EXPIRATION_DAYS,
    )


class ApplicationType(models.TextChoices):
    """User help application type."""

    REQUEST = "request", _("Request")
    PROPOSAL = "proposal", _("Proposal")


class ApplicationStatus(models.TextChoices):
    """User help application status."""

    PENDING = "pending", _("Pending")
    APPROVED = "approved", _("Approved")
    REJECTED = "rejected", _("Rejected")
    CLOSED = "closed", _("Closed")
    COMPLETED = "completed", _("Completed")
    IN_PROGRESS = "in_progress", _("In Progress")
    UNPUBLISHED = "unpublished", _("Unpublished")


class ApplicationCategory(models.TextChoices):
    """Application available categories."""

    TRANSPORT = "transport", _("Transport")
    HOUSING = "housing", _("Housing")

```

```

HUMANITARIAN = "humanitarian", _("Humanitarian help")
GOODS_SERVICES_AND_RESOURCES = "goods", _("Goods, services and resources")
OTHER = "other", _("Other")

```

```

class Application(TimeStampedModel):
    """User help application (request or proposal) model."""

    author = models.ForeignKey(
        "users.User",
        on_delete=models.CASCADE,
        related_name="applications",
        verbose_name=_("Author"),
    )
    title = models.CharField(
        max_length=128,
        verbose_name=_("Title"),
    )
    description = models.CharField(
        max_length=1024,
        verbose_name=_("Description"),
    )
    status = models.CharField(
        max_length=18,
        choices=ApplicationStatus.choices,
        default=ApplicationStatus.PENDING,
        verbose_name=_("Status")
    )
    is_urgent = models.BooleanField(
        default=False,
        verbose_name=_("Is urgent?"),
    )
    type = models.CharField(
        max_length=18,
        verbose_name=_("Application Type"),
    )
    published = models.DateTimeField(
        null=True,
        default=None,
        verbose_name=_("Date and time published"),
    )
    expires = models.DateTimeField(
        default=get_application_expire_time,

```

```

        verbose_name=_("Date and time expires"),
    )
    city = models.ForeignKey(
        "geo.City",
        on_delete=models.CASCADE,
        null=True,
        related_name="applications",
        verbose_name=_("Application target city"),
    )
    category = models.CharField(
        max_length=32,
        choices=ApplicationCategory.choices,
        verbose_name=_("Application category"),
    )

    ACTIVE_STATUSES = [
        ApplicationStatus.APPROVED,
        ApplicationStatus.IN_PROGRESS,
    ]

    class Meta:
        verbose_name = _("Application")
        verbose_name_plural = _("Applications")
        db_table = "applications"

    def __str__(self) -> str:
        return f'<Application "{self.title}" ({self.status})>'

class ApplicationSatisfaction(TimeStampedModel):
    """
    User can select other application as the one satisfying his own.

    That is done to save historical data on user manual choices, so that data
    can be used to train the recommender system model.
    """

    satisfied_application = models.ForeignKey(
        "coordinating.Application",
        on_delete=models.CASCADE,
        related_name="satisfactions",
        verbose_name=_("Satisfied application"),
    )

```

```
satisfying_application = models.ForeignKey(
    "coordinating.Application",
    on_delete=models.CASCADE,
    related_name="satisfying",
    verbose_name=_("Satisfying application"),
)
```

Б.5 Код файла “app/coordinating/selectors.py”

```
from django.db.models import QuerySet

from app.coordinating.models import Application

def get_active_applications() -> QuerySet[Application]:
    """Re-usable selector for getting list of active user applications."""
    return Application.objects.filter(
        status__in=Application.ACTIVE_STATUSES,
    )
```

Б.6 Код файла “app/geo/models.py”

```
import geopy.distance

from django.db import models
from django.utils.translation import gettext_lazy as _

class City(models.Model):
    """Generic model for storing city information."""

    name = models.CharField(max_length=64, verbose_name=_("City name"))
    latitude = models.FloatField()
    longitude = models.FloatField()

    class Meta:
        verbose_name = _("City")
        verbose_name_plural = _("Cities")

    def measure_distance(self, other: "City") -> float:
        """Calculate distance between two cities in km."""
```

```

        coordinates = self.latitude, self.longitude
        other_coordinates = other.latitude, other.longitude
        return geopy.distance.geodesic(coordinates, other_coordinates).km

def __str__(self) -> str:
    return f"<City: {self.name}>"

```

Б.7 Код файла “app/recommender/models.py”

```

from decimal import Decimal

from django.core import validators
from django.db import models
from django.utils.translation import gettext_lazy as _
from model_utils.models import TimeStampedModel

class Recommendation(TimeStampedModel):
    application = models.ForeignKey(
        "coordinating.Application",
        on_delete=models.CASCADE,
        related_name="recommendations",
        verbose_name=_("Original application")
    )
    recommended_application = models.ForeignKey(
        "coordinating.Application",
        on_delete=models.CASCADE,
        related_name="recommended_for",
        verbose_name=_("Recommended application"),
    )
    scoring = models.DecimalField(
        max_digits=10,
        decimal_places=2,
        validators=[
            validators.MinValueValidator(Decimal(0.0)),
            validators.MaxValueValidator(Decimal(1.0)),
        ],
        verbose_name=_("Scoring"),
    )

    class Meta:
        verbose_name = _("Recommendation")

```

```

verbose_name_plural = _("Recommendations")
db_table = "recommendations"
ordering = ["application_id", "-scoring"]

def __str__(self) -> str:
    return (
        f"<Recommended Application ({self.recommended_application_id}) "
        f"for Application ({self.application_id}): "
        f"{self.scoring:.02f}%>"
    )

```

Б.8 Код файла “app/recommender/serializers.py”

```

from rest_framework import serializers

from app.coordinating.models import Application
from app.users.serializers import UserContactInfoSerializer

class RecommendedApplicationSerializer(serializers.ModelSerializer):
    scoring = serializers.DecimalField(max_digits=10, decimal_places=2)
    author = UserContactInfoSerializer()
    city = serializers.StringRelatedField(source="city.name")

class Meta:
    model = Application
    fields = [
        "id",
        "title",
        "description",
        "is_urgent",
        "city",
        "type",
        "scoring",
        "author",
    ]
    read_only_fields = fields

```

Б.9 Код файлу “app/recommender/services.py”

```

from dataclasses import dataclass
from operator import attrgetter

import spacy
import numpy as np

from django.conf import settings
from django.db.models import Q, QuerySet

from app.coordinating.models import Application
from app.geo.models import City
from app.recommender.models import Recommendation

nlp = spacy.load("uk_core_news_lg")
model = pickle.load(open(settings.RECOMMENDER_MODEL_PATH))

@dataclass(frozen=True)
class RelevanceDTO:
    """
    A DTO for saving application and relevance (similarity) pair when
    generating recommendations.
    """

    application: Application
    relevance: float

class RecommenderService:
    STOPWORDS = ["", ".", "!", ";", "a", "i", "й"]
    MAX_DISTANCE = 50.0 # in km

    def generate_recommendations(
        self,
        application: Application,
        active_applications: QuerySet[Application]
    ) -> None:
        relevant_applications = self._get_relevant_applications(
            application=application,
            active_applications=active_applications,

```

```

    )

    top_5 = sorted(
        relevant_applications,
        key=attrgetter("relevance"),
        reverse=True
   )[:5]

    self._save_relevant_applications_to_database(
        application,
        relevant_applications=top_5,
    )

def calculate_relevance(
    self,
    application: Application,
    other_application: Application,
) -> float:
    description_text_similarity = self._get_text_similarity(
        application.description,
        other_application.description
    )
    title_text_similarity = self._get_text_similarity(
        application.title,
        other_application.title,
    )
    city_similarity = self._get_city_similarity(
        application.city,
        other_application.city,
    )
    is_urgent_similarity = self._get_is_urgent_similarity(
        application.is_urgent,
        other_application.is_urgent,
    )
    return.predict(np.array([
        title_text_similarity,
        description_text_similarity,
        city_similarity,
        is_urgent_similarity,
    ]))

def _get_relevant_applications(
    self,

```

```

        application: Application,
        active_applications: QuerySet[Application]
    ) -> list[RelevanceDTO]:
        possible_recommendations = active_applications.exclude(
            Q(type=application.type) | Q(author=application.author),
        )
        return [
            RelevanceDTO(
                application=suggested_application,
                relevance=self.calculate_relevance(
                    application,
                    suggested_application
                ),
            )
            for suggested_application in possible_recommendations
        ]

def _get_text_similarity(self, text: str, other: str) -> float:
    vector = nlp(self._prepare_text(text))
    other_vector = nlp(self._prepare_text(other))
    return vector.similarity(other_vector)

def _get_city_similarity(self, city: City, other_city: City) -> float:
    distance = city.measure_distance(other_city)
    if distance == 0.0:
        return 1.0
    elif distance >= self.MAX_DISTANCE:
        return 0.0
    else:
        return distance / self.MAX_DISTANCE

def _get_is_urgent_similarity(
    self,
    is_urgent: bool,
    other_is_urgent: bool,
) -> float:
    return float(is_urgent == other_is_urgent)

def _prepare_text(self, text: str) -> str:
    return " ".join(filter(
        lambda token: token not in self.STOPWORDS,
        text.split(),
    ))

```

```

def _save_relevant_applications_to_database(
    self,
    application: Application,
    relevant_applications: list[RelevanceDTO]
) -> None:
    if not relevant_applications:
        return

    if not application.recommendations.exists():
        Recommendation.objects.bulk_create([
            Recommendation(
                application=application,
                recommended_application=relevant_application.application,
                scoring=relevant_application.relevance,
            )
            for relevant_application in relevant_applications
        ])
    else:
        to_update = []

        for relevant_application, recommendation in zip(
            relevant_applications,
            application.recommendations.order_by("scoring"),
        ):
            recommendation.recommended_application = (
                relevant_application.application
            )
            recommendation.scoring = relevant_application.relevance
            to_update.append(recommendation)

        Recommendation.objects.bulk_update(
            to_update,
            fields=["recommended_application_id", "scoring"],
        )

```

Б.10 Код файла “app/recommender/tasks.py”

```

from celery import shared_task

from app.coordinating.models import Application
from app.coordinating.selectors import get_active_applications

```

```

from app.recommender.models import Recommendation
from app.recommender.services import RecommenderService

__service = RecommenderService()

@shared_task(name="recommender.generate_recommendations")
def generate_recommendations() -> None:
    """
    Generate recommendations for all active applications.
    """
    active_applications = get_active_applications()

    for application in active_applications:
        __service.generate_recommendations(
            application=application,
            active_applications=active_applications,
        )

@shared_task(name="recommender.generate_recommendations_by_application_id")
def generate_recommendations_by_application_id(application_id: int) -> None:
    """
    Generate recommendations for given application ID.
    """
    application = Application.objects.get(id=application_id)
    active_applications = get_active_applications()

    __service.generate_recommendations(
        application=application,
        active_applications=active_applications,
    )

@shared_task(name="recommender.clean_irrelevant_recommendations")
def clean_irrelevant_recommendations() -> None:
    """
    Clean recommendations for applications that are not active anymore.
    """
    Recommendation.objects.exclude(
        application__status__in=Application.ACTIVE_STATUSES,
    ).delete()

```

Б.11 Код файла “app/recommender/urls.py”

```

from django.urls import path

from app.recommender import views

app_name = "recommender"

urlpatterns = [
    path(
        "applications/<int:application_id>/recommendations/",
        views.RecommendationsView.as_view(),
        name="recommendations-list",
    )
]

```

Б.12 Код файла “app/recommender/views.py”

```

from django.db.models import OuterRef, Subquery
from django.shortcuts import get_object_or_404
from django.utils.decorators import method_decorator
from drf_yasg.utils import swagger_auto_schema
from rest_framework import filters
from rest_framework.generics import ListAPIView
from rest_framework.permissions import IsAuthenticated

from app.coordinating.models import Application
from app.recommender.models import Recommendation
from app.recommender.serializers import RecommendedApplicationSerializer

@method_decorator(
    name="get",
    decorator=swagger_auto_schema(
        operation_summary="Get list of recommendations for given application",
        operation_description=(
            "This endpoint returns 5 topmost similar applications to the one "
            "was given (proposals for given request and vice versa)"
        ),
        tags=["recommendations"],
    ),
)

```

```

    )
)

class RecommendationsView(ListAPIView):
    permission_classes = [IsAuthenticated]
    serializer_class = RecommendedApplicationSerializer

    filter_backends = [filters.OrderingFilter]
    ordering_fields = ["title", "scoring"]
    ordering = ["-scoring"]

    def get_queryset(self):
        author_id = self.request.user.id
        application_id = self.kwargs.get("application_id", 0)

        target_application = get_object_or_404(
            Application,
            id=application_id,
            author_id=author_id,
        )

        return Application.objects.filter(
            id__in=Subquery(
                Recommendation
                .objects
                .filter(application=target_application)
                .values("recommended_application_id")
            )
        ).annotate(
            scoring=Subquery(
                Recommendation
                .objects
                .filter(
                    application_id=target_application.id,
                    recommended_application_id=OuterRef("id"),
                )
                .values("scoring")[:1]
            )
        )

```

Б.13 Код файла “modeling/model_creation.py”

```

import pandas as pd
import pickle
import numpy as np
import tqdm
from settings.credentials import MODELS_PATH
from modeling.base_models import (
    Application,
    TITLE_COEF,
    CONTEXT_COEF,
    IS_URGENT_COEF,
    CITY_COEF
)
from modeling.utils import (
    vector_text,
    get_city_distance_weight,
)
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

pd.set_option('display.max_rows', 999999)
pd.set_option('display.max_columns', 999999)
pd.set_option('display.width', 999999)

recommended_df = pd.read_csv("../data/train.csv", sep=';', header=0)
print(recommended_df.head())

recommended_df = recommended_df.sample(frac=1).reset_index(drop=True)

X = []
Y = []

for i in tqdm.tqdm(range(len(recommended_df))):
    application = Application(
        title=recommended_df.iloc[i]['title'],
        description=recommended_df.iloc[i]['description'],
        city=recommended_df.iloc[i]['city'],
        is_urgent=recommended_df.iloc[i]['is_urgent'],
        category=recommended_df.iloc[i]['category'],

```

```

)
recommended_application = Application(
    title=recommended_df.iloc[i]['recommended_title'],
    description=recommended_df.iloc[i]['recommended_description'],
    city=recommended_df.iloc[i]['recommended_city'],
    is_urgent=recommended_df.iloc[i]['recommended_is_urgent'],
    category=recommended_df.iloc[i]['recommended_category'],
)

title = abs(cosine_similarity(
    np.array(vector_text(application)).reshape(1, -1),
    np.array(vector_text(recommended_application)).reshape(1, -1)
)[0][0])

context = abs(cosine_similarity(
    np.array(vector_text(application)).reshape(1, -1),
    np.array(vector_text(recommended_application)).reshape(1, -1)
)[0][0])

is_urgent = float(application.is_urgent ==
recommended_application.is_urgent)

city = get_city_distance_weight(application.city,
recommended_application.city)

target = title * TITLE_COEF + context * CONTEXT_COEF + is_urgent *
IS_URGENT_COEF + city * CITY_COEF

category = float(application.category == recommended_application.category)

if not category:
    target *= 0.0

X.append([title, context, is_urgent, city, category])
Y.append(target)

X = np.array(X)
Y = np.array(Y)

x_train, x_test, y_train, y_test = train_test_split(X, Y, random_state=42,
test_size=0.1)
print(x_train)

```

```

model = LinearRegression()
model.fit(x_train, y_train)

print("Error: ", np.sqrt(mean_squared_error(y_test, model.predict(x_test))))

model_parameters = ['title', 'context', 'salary', 'skill_set', 'clearance']
for i, j in zip(model_parameters, model.coef_):
    print(i, j)
print('Intercept: ', model.intercept_)

# Error:  0.11401425330702566

# title 0.43008359118760475
# context 0.1481650876793154
# salary 0.005196050198663381
# skill_set 0.09090881608138458
# clearance -0.03073749688890422

# Intercept:  0.07085611721888835

pickle.dump(model, open(f"{MODELS_PATH}/model.pickle", 'wb'))

```

Б.14 Код програми “tests/integration/test_views.py”

```

from datetime import datetime, timezone
from unittest.mock import ANY

import pytest

from django.urls import reverse
from eav.models import Attribute, EnumGroup, EnumValue
from rest_framework import status
from rest_framework.test import APIClient

from geo.models import City
from help_options.models import HelpOption, HelpOptionAttributes
from profiles.models import Profile
from user_requests.constants import UserRequestStatus
from user_requests.factories import UserRequestFactory
from user_requests.models import UserRequest
from users.constants import UserStatuses
from users.factories import UserFactory

```

```

class TestUserRequestView:
    pytestmark = pytest.mark.django_db

    @pytest.mark.parametrize(
        'user_request_status',
        [
            UserRequestStatus.PENDING,
            UserRequestStatus.UNPUBLISHED,
        ]
    )
    @pytest.mark.parametrize(
        'user_request_owner_status',
        [
            UserStatuses.TRUSTED,
            UserStatuses.AUTO_VERIFIED,
        ]
    )
    @pytest.mark.freeze_time('2022-02-23')
    def test_user_request_view_publish(
        self,
        user_request_status: UserRequestStatus,
        user_request_owner_status: UserStatuses,
        api_client: APIClient,
    ):
        # arrange
        user_request = UserRequestFactory.create(
            status=user_request_status,
            owner=UserFactory.create(status=user_request_owner_status),
        )

        api_client.force_authenticate(user=user_request.owner)

        url = reverse(
            viewname='user_requests:user-request-publish',
            kwargs={'pk': user_request.pk},
        )

        # act
        response = api_client.put(url)

        # assert

```

```

assert response.status_code == status.HTTP_200_OK, response.content

user_request.refresh_from_db()
assert user_request.status == UserRequestStatus.APPROVED
assert user_request.publish_date == datetime(
    2022, 2, 23,
    tzinfo=timezone.utc,
)

@pytest.mark.parametrize(
    'user_request_status',
    [
        UserRequestStatus.APPROVED,
        UserRequestStatus.REJECTED,
        UserRequestStatus.IN_PROGRESS,
        UserRequestStatus.CLOSED,
        UserRequestStatus.COMPLETED,
    ]
)
def test_user_request_view_publish_forbidden_invalid_status(
    self,
    user_request_status: UserRequestStatus,
    api_client: APIClient,
):
    # arrange
    user_request = UserRequestFactory.create(status=user_request_status)

    api_client.force_authenticate(user=user_request.owner)

    url = reverse(
        viewname='user_requests:user-request-publish',
        kwargs={'pk': user_request.pk}
    )

    # act
    response = api_client.put(url)

    # assert
    assert response.status_code == status.HTTP_403_FORBIDDEN

@pytest.mark.parametrize(
    'user_request_owner_status',
    [

```

```

        UserStatuses.ADMIN,
        UserStatuses.BANNED,
        UserStatuses.UNVERIFIED,
    ]
)
def test_user_request_view_publish_forbidden_invalid_user_status(
    self,
    user_request_owner_status: UserStatuses,
    api_client: APIClient,
):
    # arrange
    user_request = UserRequestFactory.create(
        status=UserRequestStatus.PENDING,
        owner=UserFactory.create(status=user_request_owner_status),
    )

    api_client.force_authenticate(user=user_request.owner)

    url = reverse(
        viewname='user_requests:user-request-publish',
        kwargs={'pk': user_request.pk}
    )

    # act
    response = api_client.put(url)

    # assert
    assert response.status_code == status.HTTP_403_FORBIDDEN

def test_user_request_view_publish_forbidden_non_owner(
    self,
    api_client: APIClient,
):
    # arrange
    owner = UserFactory.create(status=UserStatuses.TRUSTED)
    non_owner = UserFactory.create(status=UserStatuses.TRUSTED)

    api_client.force_authenticate(user=non_owner)

    user_request = UserRequestFactory.create(
        status=UserRequestStatus.PENDING,
        owner=owner,
    )

```

```

url = reverse(
    viewname='user_requests:user-request-publish',
    kwargs={'pk': user_request.pk}
)

# act
response = api_client.put(url)

# assert
assert response.status_code == status.HTTP_404_NOT_FOUND

# UNPUBLISH

@pytest.mark.parametrize(
    'user_request_owner_status',
    [
        UserStatuses.TRUSTED,
        UserStatuses.AUTO_VERIFIED,
    ]
)
def test_user_request_view_unpublish(
    self,
    user_request_owner_status: UserStatuses,
    api_client: APIClient,
):
    # arrange
    user_request = UserRequestFactory.create(
        status=UserRequestStatus.APPROVED,
        owner=UserFactory.create(status=user_request_owner_status),
    )

    api_client.force_authenticate(user=user_request.owner)

    url = reverse(
        viewname='user_requests:user-request-unpublish',
        kwargs={'pk': user_request.pk},
    )

    # act
    response = api_client.put(url)

    # assert

```

```

assert response.status_code == status.HTTP_200_OK, response.content

user_request.refresh_from_db()
assert user_request.status == UserRequestStatus.UNPUBLISHED

@pytest.mark.parametrize(
    'user_request_status',
    [
        UserRequestStatus.PENDING,
        UserRequestStatus.REJECTED,
        UserRequestStatus.IN_PROGRESS,
        UserRequestStatus.CLOSED,
        UserRequestStatus.COMPLETED,
        UserRequestStatus.UNPUBLISHED,
    ]
)
def test_user_request_view_unpublish_forbidden_invalid_status(
    self,
    user_request_status: UserRequestStatus,
    api_client: APIClient,
):
    # arrange
    user_request = UserRequestFactory.create(status=user_request_status)

    api_client.force_authenticate(user=user_request.owner)

    url = reverse(
        viewname='user_requests:user-request-unpublish',
        kwargs={'pk': user_request.pk}
    )

    # act
    response = api_client.put(url)

    # assert
    assert response.status_code == status.HTTP_403_FORBIDDEN

@pytest.mark.parametrize(
    'user_request_owner_status',
    [
        UserStatuses.ADMIN,
        UserStatuses.BANNED,
        UserStatuses.UNVERIFIED,

```

```

    ]
)
def test_user_request_view_unpublish_forbidden_invalid_user_status(
    self,
    user_request_owner_status: UserStatuses,
    api_client: APIClient,
):
    # arrange
    user_request = UserRequestFactory.create(
        status=UserRequestStatus.APPROVED,
        owner=UserFactory.create(status=user_request_owner_status),
    )

    api_client.force_authenticate(user=user_request.owner)

    url = reverse(
        viewname='user_requests:user-request-unpublish',
        kwargs={'pk': user_request.pk}
    )

    # act
    response = api_client.put(url)

    # assert
    assert response.status_code == status.HTTP_403_FORBIDDEN

def test_user_request_view_unpublish_forbidden_non_owner(
    self,
    api_client: APIClient,
):
    # arrange
    owner = UserFactory.create(status=UserStatuses.TRUSTED)
    non_owner = UserFactory.create(status=UserStatuses.TRUSTED)

    api_client.force_authenticate(user=non_owner)

    user_request = UserRequestFactory.create(
        status=UserRequestStatus.APPROVED,
        owner=owner,
    )

    url = reverse(
        viewname='user_requests:user-request-unpublish',

```

```

        kwargs={'pk': user_request.pk}
    )

    # act
    response = api_client.put(url)

    # assert
    assert response.status_code == status.HTTP_404_NOT_FOUND

class TestUserRequestSearchView:
    pytestmark = pytest.mark.django_db

    def setup_method(self, method):
        self.client = APIClient()
        self.url = reverse('user_requests:search-list')

    @pytest.fixture(autouse=True)
    def cities(self, db) -> list[City]:
        return [
            City.objects.create(name='Zaporizhzhia'),
            City.objects.create(name='Lviv'),
            City.objects.create(name='Kyiv'),
        ]

    @pytest.fixture(autouse=True)
    def help_option_categories(self, db) -> list[HelpOption]:
        transport_option = HelpOption.objects.create(name='Transport')
        other_option = HelpOption.objects.create(name='Other')

        active = EnumValue.objects.create(value='active')
        inactive = EnumValue.objects.create(value='inactive')

        enum_group = EnumGroup.objects.create(name='Status')
        enum_group.values.add(active, inactive)

    # creating attributes by all allowed datatypes
    attributes = [
        Attribute.objects.create(
            name='Description',
            slug='description',
            datatype=Attribute.TYPE_TEXT,
        ),

```

```

Attribute.objects.create(
    name='Count',
    slug='count',
    datatype=Attribute.TYPE_INT,
),
Attribute.objects.create(
    name='Percentage',
    slug='percentage',
    datatype=Attribute.TYPE_FLOAT,
),
Attribute.objects.create(
    name='Destination Date',
    slug='destination_date',
    datatype=Attribute.TYPE_DATE
),
Attribute.objects.create(
    name='Status',
    slug='status',
    datatype=Attribute.TYPE_ENUM,
    enum_group=enum_group,
)
]

for attribute in attributes:
    HelpOptionAttributes.objects.create(
        help_option=other_option,
        attribute=attribute,
    )

return [
    transport_option,
    other_option,
]

@pytest.mark.parametrize(
    'user_status',
    [
        UserStatuses.TRUSTED,
        UserStatuses.AUTO_VERIFIED,
    ]
)

def test_get(self, user_status: UserStatuses) -> None:
    # arrange

```

```

lviv = City.objects.get(name='Lviv')
kyiv = City.objects.get(name='Kyiv')

transport = HelpOption.objects.get(name='Transport')
other = HelpOption.objects.get(name='Other')

phone_number = '+380980000000'
user = UserFactory.create(status=user_status)
Profile.objects.create(user=user, phone_number=phone_number)

user_request_1 = UserRequest.objects.create(
    status=UserRequestStatus.APPROVED,
    city=lviv,
    help_option=other,
    eav__status='active',
    is_urgent=True,
    owner=user,
    publish_date=datetime.now(tz=timezone.utc),
)

# Should be filtered (city)
UserRequest.objects.create(
    status=UserRequestStatus.APPROVED,
    city=kyiv,
    help_option=other,
    eav__status='active',
    is_urgent=True,
    owner=user,
    publish_date=datetime.now(tz=timezone.utc),
)

# Should be filtered (help_option)
UserRequest.objects.create(
    status=UserRequestStatus.APPROVED,
    city=lviv,
    help_option=transport,
    eav__status='active',
    is_urgent=True,
    owner=user,
    publish_date=datetime.now(tz=timezone.utc),
)

# Should be filtered (eav__status)

```

```

UserRequest.objects.create(
    status=UserRequestStatus.APPROVED,
    city=lviv,
    help_option=other,
    eav__status='inactive',
    is_urgent=True,
    owner=user,
    publish_date=datetime.now(tz=timezone.utc),
)

# Should be filtered (is_urgent)
UserRequest.objects.create(
    status=UserRequestStatus.APPROVED,
    city=lviv,
    help_option=other,
    eav__status='active',
    is_urgent=False,
    owner=user,
    publish_date=datetime.now(tz=timezone.utc),
)

# Should not be filtered
user_request_2 = UserRequest.objects.create(
    status=UserRequestStatus.APPROVED,
    city=lviv,
    help_option=other,
    eav__status='active',
    is_urgent=True,
    owner=user,
    publish_date=datetime.now(tz=timezone.utc),
)

# act
self.client.force_authenticate(user)
response = self.client.get(self.url, data={
    'is_urgent': True,
    'city': lviv.name,
    'help_option_category': other.name,
    'status': 'active',
    'ordering': '-publish_date,is_urgent',
})

# assert

```

```

assert response.status_code == status.HTTP_200_OK, response.content
assert response.json() == [
    {
        'id': user_request_2.id,
        'title': '',
        'description': '',
        'expiration_date': ANY,
        'is_urgent': True,
        'help_option': 'Other',
        'city': 'Lviv',
        'publish_date': ANY,
        'owner': {
            'id': ANY,
            'first_name': user.first_name,
            'last_name': user.last_name,
            'phone_number': phone_number,
        },
        'attributes': {'status': 'active'},
    },
    {
        'id': user_request_1.id,
        'title': '',
        'description': '',
        'expiration_date': ANY,
        'is_urgent': True,
        'help_option': 'Other',
        'city': 'Lviv',
        'publish_date': ANY,
        'owner': {
            'id': ANY,
            'first_name': user.first_name,
            'last_name': user.last_name,
            'phone_number': phone_number
        },
        'attributes': {'status': 'active'},
    },
]

```

```
def test_options(self):
```

```
    response = self.client.options(path=self.url)
```

```
    assert response.status_code == status.HTTP_200_OK, response.status_code
```

```
    assert response.json() == {
```

```

'name': 'User Request Search List',
'description': '',
'renders': [
    'application/json',
    'text/html',
],
'parses': [
    'application/json',
    'application/x-www-form-urlencoded',
    'multipart/form-data',
],
'filters': {
    'is_urgent': {
        'type': 'boolean',
        'required': False,
        'read_only': True,
        'label': 'Is Urgent'
    },
    'city': {
        'type': 'choice',
        'required': False,
        'read_only': True,
        'label': 'City',
        'choices': [
            {'value': 'Kyiv', 'display_name': 'Kyiv'},
            {'value': 'Lviv', 'display_name': 'Lviv'},
            {
                'value': 'Zaporizhzhia',
                'display_name': 'Zaporizhzhia'
            }
        ]
    },
    'help_option_category': {
        'type': 'choice',
        'required': False,
        'read_only': True,
        'label': 'Help Option Category',
        'choices': [
            {'value': 'Transport', 'display_name': 'Transport'},
            {'value': 'Other', 'display_name': 'Other'}
        ]
    }
},

```

```

        'ordering_fields': [
            'publish_date',
            'is_urgent',
        ]
    }

def test_options_with_help_category(self):
    response = self.client.options(
        path=self.url,
        data={'help_option_category': 'Other'},
    )

    assert response.status_code == status.HTTP_200_OK, response.content
    assert response.json()['filters'] == {
        'is_urgent': {
            'type': 'boolean',
            'required': False,
            'read_only': True,
            'label': 'Is Urgent',
        },
        'city': {
            'type': 'choice',
            'required': False,
            'read_only': True,
            'label': 'City',
            'choices': [
                {'value': 'Kyiv', 'display_name': 'Kyiv'},
                {'value': 'Lviv', 'display_name': 'Lviv'},
                {
                    'value': 'Zaporizhzhia',
                    'display_name': 'Zaporizhzhia'
                }
            ],
        },
        'help_option_category': {
            'type': 'choice',
            'required': False,
            'read_only': True,
            'label': 'Help Option Category',
            'choices': [
                {'value': 'Transport', 'display_name': 'Transport'},
                {'value': 'Other', 'display_name': 'Other'}
            ],
        }
    }

```

```

},
'description': {
    'type': 'string',
    'required': False,
    'read_only': True,
    'label': 'Description',
},
'count_min': {
    'type': 'integer',
    'required': False,
    'read_only': True,
    'label': 'Count Min',
},
'count_max': {
    'type': 'integer',
    'required': False,
    'read_only': True,
    'label': 'Count Max'
},
'percentage_min': {
    'type': 'float',
    'required': False,
    'read_only': True,
    'label': 'Percentage Min',
},
'percentage_max': {
    'type': 'float',
    'required': False,
    'read_only': True,
    'label': 'Percentage Max',
},
'destination_date_before': {
    'type': 'date',
    'required': False,
    'read_only': True,
    'label': 'Destination Date Before',
},
'destination_date_after': {
    'type': 'date',
    'required': False,
    'read_only': True,
    'label': 'Destination Date After',
},

```

```

        'status': {
            'type': 'choice',
            'required': False,
            'read_only': True,
            'label': 'Status',
            'choices': [
                {'value': 'active', 'display_name': 'active'},
                {'value': 'inactive', 'display_name': 'inactive'}
            ],
        },
    },
}

```

Б.15 Код файла “app/users/authentication_backends.py”

```

from typing import Any, Optional

from django.conf import settings
from django.contrib.auth.backends import ModelBackend
from django.http import HttpRequest
from rest_framework.authentication import BaseAuthentication
from rest_framework.exceptions import AuthenticationFailed

from app.users.models import User

class CustomAuthenticationHeaderMixin:
    """
    Mixin that implements private _authenticate method that gets the user
    by User ID given in special request header.

    The class is implemented as a mixin so that it could be used with Django
    and Django REST Framework authentication backends.
    """

    def _authenticate(self, request: HttpRequest) -> Optional[User]:
        user_id = request.headers.get(settings.AUTHENTICATION_HEADER_NAME)

        if user_id is not None:
            user = User.objects.filter(id=user_id).first()
            return user if user else None

```

```

class HeaderBasedBackend(CustomAuthenticationHeaderMixin, ModelBackend):
    """
    Django's ModelBackend extended to authenticate users via custom
    authentication header that is providing user ID.
    """

    def authenticate(
        self,
        request: HttpRequest,
        **kwargs: Any,
    ) -> Optional[User]:
        return self._authenticate(request)


class HeaderBasedAuthentication(
    CustomAuthenticationHeaderMixin,
    BaseAuthentication
):
    """
    Django REST Framework's BaseAuthentication implementation for
    authenticating users using custom authentication header.
    """

    def authenticate(self, request):
        user = self._authenticate(request)
        if not user:
            raise AuthenticationFailed("No such user")
        return user, None

```

Б.16 Код файла “app/users/managers.py”

```

from django.contrib.auth.base_user import BaseUserManager
from django.db import transaction


class UserManager(BaseUserManager):
    """Custom model manager for User model.

    It provides two extra methods: create_user and create_superuser.
    """

    def _create_user(self, email, password, **extra_fields):

```

```

if not email:
    raise ValueError("The given email must be set")
with transaction.atomic():
    user = self.model(email=email, **extra_fields)
    user.set_password(password)
    user.save(using=self._db)
    return user

def create_user(self, email, password=None, **extra_fields):
    extra_fields.setdefault("is_staff", False)
    extra_fields.setdefault("is_superuser", False)
    return self._create_user(email, password, **extra_fields)

def create_superuser(self, email, password, **extra_fields):
    extra_fields.setdefault("is_staff", True)
    extra_fields.setdefault("is_superuser", True)
    return self._create_user(email, password=password, **extra_fields)

```

Б.17 Код файла “app/users/models.py”

```

from django.contrib.auth.base_user import AbstractBaseUser
from django.contrib.auth.models import PermissionsMixin
from django.db import models
from django.utils.translation import gettext_lazy as _

from app.users.managers import UserManager

class UserType(models.TextChoices):
    """Enumeration for user types."""

    VOLUNTEER = "volunteer", _("Volunteer")
    HELP_RECEIVER = "help_receiver", _("Help receiver")
    ADMIN = "admin", _("Admin")

class User(AbstractBaseUser, PermissionsMixin):
    """Main applications model for storing User profile data."""

    email = models.EmailField(
        unique=True,
        db_index=True,

```

```

        verbose_name=_("User contact email"),
    )
    first_name = models.CharField(max_length=32, verbose_name=_("First name"))
    last_name = models.CharField(max_length=32, verbose_name=_("Last name"))
    phone_number = models.CharField(
        max_length=13,
        null=True,
        default=None,
        verbose_name=_("User contact mobile phone number"),
    )
    type = models.CharField(
        max_length=13,
        choices=UserType.choices,
        verbose_name=_("User type"),
    )

    USERNAME_FIELD = "email"

    objects = UserManager()

    class Meta:
        verbose_name = _("User")
        verbose_name_plural = _("Users")
        db_table = "users"

    def __str__(self) -> str:
        return f"<User: {self.email}>"

```

Б.18 Код файла “app/users/serializers.py”

```

from rest_framework import serializers

from app.users.models import User

class UserContactInfoSerializer(serializers.ModelSerializer):
    """Simple serializer to provide user's contact information."""

    class Meta:
        model = User
        fields = [
            "id",

```

```
    "first_name",  
    "last_name",  
    "email",  
    "phone_number",  
]
```

ДОДАТОК В
Слайди презентації

Слайди презентації зображені на рис. В.1 – В.20.

Національний університет “Запорізька політехніка”

Кафедра програмних засобів

Дипломна кваліфікаційна робота

Дослідження та програмна реалізація рекомендаційної системи для волонтерського координатора

Підготував:
ст. гр. КНТ-121м

Дьячков В. В.

Керівник:
доцент

Федорончак Т. В.

1

Рисунок В.1 – Слайд презентації №1

Об’єкт дослідження – розподілення ресурсів волонтерської організації між наявними запитами на допомогу.

Предметом дослідження є ті чинники, що впливають на рішення, що приймає координатор волонтерської організації чи отримувач допомоги під час визначення, який саме ресурс організації буде направлено на задовільнення того чи іншого запита на допомогу.

Метою роботи стоїть дослідження і програмна реалізація рекомендаційної системи для волонтерського координатора.

Розроблене програмне забезпечення, разом зі створеною рекомендаційною моделлю, стане компонентом майбутньої системи “Волонтерський координатор”, що так само наразі знаходиться на етапі розробки.

2

Рисунок В.2 – Слайд презентації №2

Огляд існуючої моделі предметної області

Системою волонтерський координатор користуються **волонтери, шукачі допомоги і адміністратори**.

Шукачі допомоги створюють запити на отримання допомоги (один з типів заявок).

Волонтери створюють пропозиції про надання допомоги (інший тип заявок).

Адміністратори модерують створені заявки, перевіряючи коректність заповнення даних, а також відмітаючи можливий спам.

3

Рисунок В.3 – Слайд презентації №3

Життєвий цикл заявки

Кожна заявка має власний життєвий цикл – шлях котрий вона проходить з моменту створення до моменту її закриття і видалення з системи.

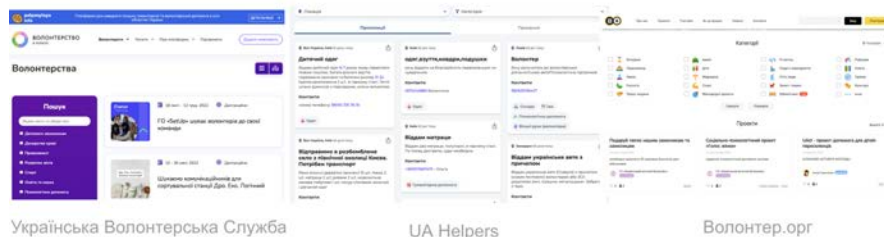
На даному слайді зображена блок-схема, що описує життєвий цикл заявки в системі "Волонтерський координатор".



4

Рисунок В.4 – Слайд презентації №4

Огляд систем-аналогів



5

Рисунок В.5 – Слайд презентації №5

Порівняння систем-аналогів

Критерій порівняння	Назва системи		
	volunteer.country	UA Helpers	Волонтер.org
Можливість пошуку пропозицій	х	✓	х
Можливість пошуку запитів	✓	✓	✓
Можливість повнотекстового пошуку за описом	✓	х	х
Можливість пошуку за місцезнаходженням	✓	✓	✓
Можливість пошуку за категорією допомоги	✓	✓	✓
Наявність системи авторизації для збереження користувацької інформації і історії активності	х	х	✓
Наявність рекомендацій при пошуку пропозицій чи запитів	х	х	х

6

Рисунок В.6 – Слайд презентації №6

Висновки з аналізу предметної області

Під час пошуку релевантних для себе **заявок (пропозицій і запитів)**, користувачі найчастіше звертають увагу на:

- категорію допомоги;
- місцезнаходження допомоги;
- короткий текстовий опис допомоги;
- повний, змістовний опис допомоги.

Відповідно, майбутня рекомендаційна система має враховувати поведінку користувачів при знаходженні релевантних заявок і надавати рекомендації, враховуючи всі вищезазначені критерії пошуку.

7

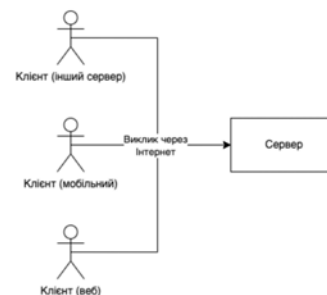
Рисунок В.7 – Слайд презентації №7

Розробка архітектури програми

На самому початку етапу проектування було прийнято рішення, що майбутнє ПЗ має слідувати **клієнт-серверній архітектурі**.

Це дозволить покласти задачу по генерації користувацьких рекомендацій на віддалений сервер, до якого користувачі матимуть змогу звертатись через мережу для отримання списку їх персоналізованих рекомендацій.

Крім того, вибір клієнт-серверної архітектури дозволить легко інтегрувати розроблене ПЗ до системи "Волонтерський координатор", оскільки всі інші її компоненти представлені також у вигляді веб-застосунків.



8

Рисунок В.8 – Слайд презентації №8

Компоненти системи



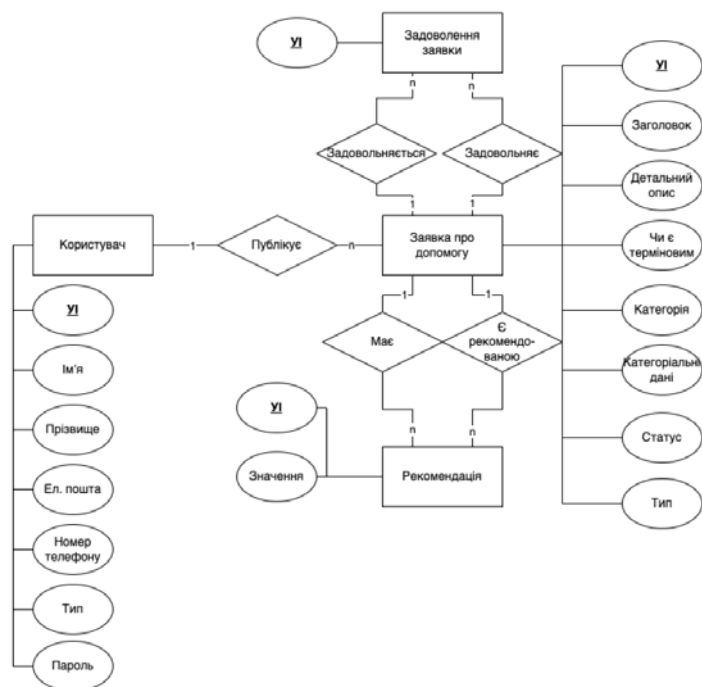
Діаграма залежностей компонентів системи

9

Рисунок В.9 – Слайд презентації №9

Проектування схеми даних

Перед переходом до наступного етапу, було також складено діаграму відношень між сутностями предметної області (Entity Relationship Diagram), що є відображенням схеми даних в системі і стане прототипом майбутньої моделі даних в обраній базі даних



10

Рисунок В.10 – Слайд презентації №10

Вибір мови програмування для реалізації

Мова програмування	Java	JavaScript	Python	R
Особливість				
Наявність веб-фреймворків для написання серверної частини	+	+	+	-
Наявність готових бібліотек для написання рекомендаційних систем	+	+	+	-
Наявність повномасштабної стандартної бібліотеки, що надає інструменти для вирішення більшої частини типових задач	+/-	-	+	-
Простота синтаксису	+/-	+/-	+	+/-
Швидкість роботи	+	+	+/-	+
Швидкість розробки	-	+	+	+/-

11

Рисунок В.11 – Слайд презентації №11

Розробка рекомендаційної моделі

Для генерації користувацьких рекомендацій необхідно знайти найбільш релевантні, тобто схожі заявки.

Знаходження ступеню схожості заявок відбувається за формулою, зображеною на цьому слайді.

Ступінь схожості заявок визначається за ознаками (атрибутами), що були визначені в результаті аналізу предметної області і перелічені на слайді № 11.

$$s = \sum_{i=1}^N a_i c_i$$

Тут, s – це ступінь схожості, що приймає значення у діапазоні $[0;1]$, a_i – i -та ознака (атрибут) заявки, c_i – коефіцієнт вкладу i -ї ознаки заявки, N – кількість ознак.

12

Рисунок В.12 – Слайд презентації №12

Знаходження близькості текстів

Знаходження близькості текстів було реалізовано з використанням найкращих підходів в області NLP (Natural Language Processing).

Для визначення семантичної збіжності двох текстів було обрано **метод знаходження косинуса подібності** між векторизованими представленнями двох довільних текстів.

Векторизація текстів відбувається з використанням методу *Word2Vec*. Перед їх векторизацією, тексти також проходять етапи **токенізації** і **лемматизації**, для покращення кінцевого результату порівняння.

Для виконання всіх вищезазначених операцій над текстом було використано готове рішення на мові програмування Python – бібліотеку **spracy**

$$S_c(A, B) = \cos(\theta) = \frac{AB}{|A||B|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Тут $S_c(A, B)$ – косинус подібності між векторами A і B ,
– кут між векторами A і B , A_i – i -тий компонент вектору A , B_i – i -тий компонент вектору B .

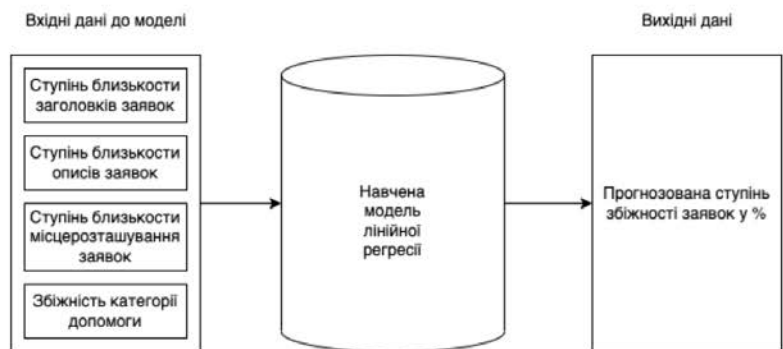
13

Рисунок В.13 – Слайд презентації №13

Навчання і інкапсуляція моделі

Після визначення ступенів вкладу кожної окремої ознаки заявки на кінцевий результат ступеню схожості між заявками, було створено тестовий датасет, де для кожної пари заявка-пропозиція було визначено відповідний ступінь збіжності.

Далі, за отриманими результатами було навчено модель лінійної регресії, де вхідними даними (X) виступали ступені близькості атрибутів заявки, а прогнозованою величиною (Y) – ступінь збіжності заявок.



Схематичне зображення принципу роботи моделі

14

Рисунок В.14 – Слайд презентації №14

Оцінювання якості моделі

В якості метрики оцінювання якості побудованої рекомендаційної моделі було обрано метрику $nDCG$ (Normalized Discounted Cumulative Gain). Вона часто використовується в оцінюванні методів ранжування, в задачах знаходження і вишикування об'єктів за ступенем релевантності.

Тут, на формулах, $nDCG_p$ – нормалізована метрика DCG для рангу p , DCG_p – метрика DCG для рангу p , $IDCG_p$ – ідеальна метрика DCG для рангу p , rel_i – i -та ґрадуована релевантність розглядаємого документу, REL_i – список ідеально ґрадуованих документів в порядку спадання значення їх релевантності.

$$nDCG_p = \frac{DCG_p}{IDCG_p}$$

$$DCG_p = \sum_{i=1}^p \frac{rel_i}{\log_2(i+1)}$$

$$IDCG_p = \sum_{i=1}^{|REL|} \frac{rel_i}{\log_2(i+1)}$$

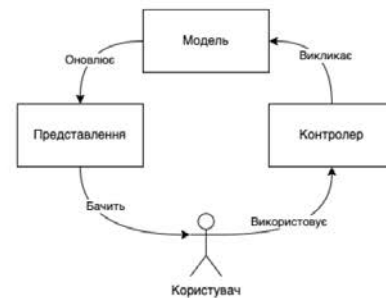
15

Рисунок В.15 – Слайд презентації №15

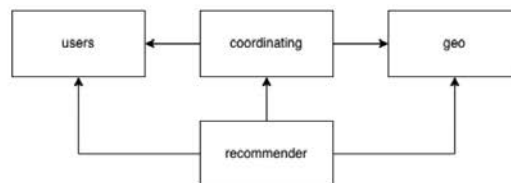
Розробка веб-застосунку

Розробка веб-застосунку велась з використанням веб-фреймворку Django і його розширення Django REST Framework.

Використовуючи DRF було описано декілька методів відкритого програмного інтерфейсу, що були всі задокументовані і викладені в автоматично згенерованій документації до API за форматом Swagger.



MVC-архітектура, використана під час розробки веб-застосунку

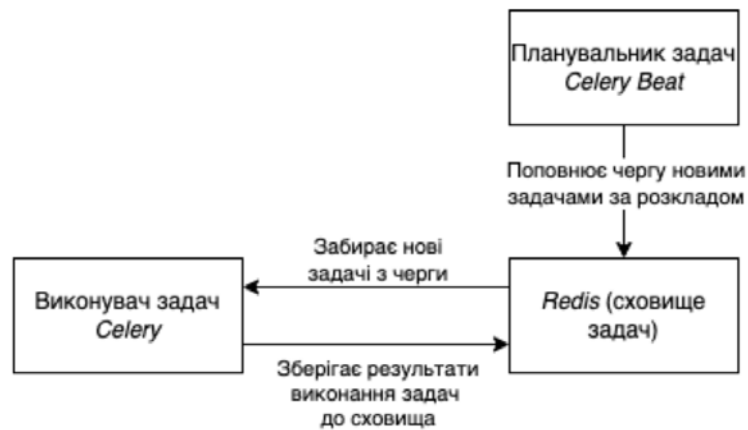


Діаграма залежностей компонентів системи

16

Рисунок В.16 – Слайд презентації №16

Розробка виконувача періодичних задач



Діаграма взаємодії компонентів виконувача періодичних і фонових задач

17

Рисунок В.17 – Слайд презентації №17

Тестування і результати

```

user_proposals/views.py      40    2   95%
user_requests/admin.py       20    3   85%
user_requests/serializers.py  42    5   88%
user_requests/services.py     38   10   74%
user_requests/views.py        40    2   95%
users/managers.py             19   13   32%
users/models.py               20    1   95%
users/serializers.py          38    1   97%
users/services/emails_verify.py 37   20   46%
users/services/user.py        17    9   47%
users/views.py                46    6   87%
-----
TOTAL                        1148   89   92%

56 files skipped due to complete coverage.

===== 69 passed, 248 warnings in 1.91s
  
```

Консольне виведення результат запуску автоматизованих тестів

```

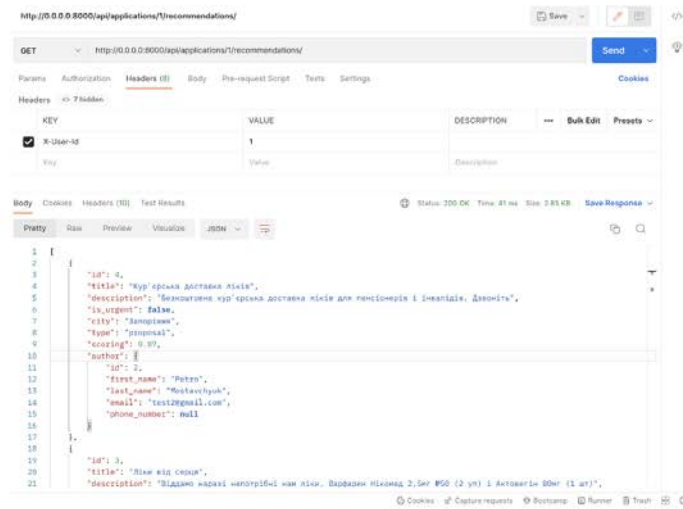
(.venv) $ python3 modeling/model_evaluation.py
Running model evaluation using nDCG metrics
Using test dataset from modeling/data/test.csv
nDCG = 0.7902137
  
```

Запуск скрипта для підбиття метрики
якості моделі

18

Рисунок В.18 – Слайд презентації №18

Приклад виконання програми



Відправлення тестового HTTP-запиту з клієнту
Postman

19

Рисунок В.19 – Слайд презентації №19

Висновки

В результаті виконання дипломної роботи було досліджено процес розподілення волонтерських ресурсів між отримувачами допомоги, а також розроблено рекомендаційну модель з відкритим програмним інтерфейсом, що має можливість підібрати найбільш релевантні запити до пропозиції про надання допомоги, і навпаки – пропозиції до запитів.

Розроблене програмне забезпечення відповідає всім вимогам, поставленим до нього у технічному завданні до проєкту, а також сучасним стандартам проєктування ПЗ і написання програмного коду.

20

Рисунок В.20 – Слайд презентації №20