Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Програмна система персоналізованих рекомендацій на основі аналізу користувацьких даних
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ. 2101095.01.21.ПЗ

Виконала студентка IV курсу, група ІПЗ-21-1 _____ Софія ЧЕРНУШИНА
Підпис Ім'я, ПРІЗВИЩЕ

Керівник старший викладач _____ Ганна БЕДРАТЮК
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. пед. наук, доцент _____ Наталія ПРАВОРСЬКА
Посада Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

2 червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
_____ Л. П. Бедратюк
7.02. 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Чернушина Софія Олександрівна

Прізвище, ім'я, по батькові студента

1. Тема роботи Програмна система персоналізованих рекомендацій на основі аналізу користувацьких даних

Керівник роботи Бедратюк Ганна Іванівна, старший викладач

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2025 р. №8
2. Строк подання студентом роботи на кафедру 01.06.2025 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області, проєктування програмного забезпечення, програмна реалізація
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди, 16 шт.)
Графічні матеріали (5 шт.)

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Н.І. Праворська, канд. пед. наук, доцент	8.05.25	8.05.25
Антиплагіат	Форкун Ю. В., канд. техн. наук, доцент	12.05.2025	02.06.2025

7. Дата видачі завдання « 02 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1. Ознайомлення з тематикою дипломного проектування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12– 31.12.2024	
2. Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використан- тання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02 2025	
3. Проектування програмного забезпечення	21.02 – 20.03 2025	
4. Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2025	
5. Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2025	
6. Попередній захист	Травень 2025	
7. Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05 2025	
8. Здача КвР на кафедру: підготовка КвР для розміщення у репозитарії ІТУ; підготовка до захисту та захист КвР	з 1.06 2025	

Студент _____ С.О. Чернушина
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Г.І. Бедратюк
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Програмна система персоналізованих рекомендацій на основі аналізу користувацьких даних».

Автор проєкту: Чернушина Софія Олександрівна.

Керівник проєкту: Бедратюк Ганна Іванівна.

Пояснювальна записка: 90 с., 22 рис., 3 табл., 3 дод., 41 джерел.

Графічна частина: 4 креслення формату А4.

АНАЛІЗ ДАНИХ, ПЕРСОНАЛІЗОВАНІ РЕКОМЕНДАЦІЇ, ПРОГРАМНА СИСТЕМА, ШТУЧНИЙ ІНТЕЛЕКТ.

Мета кваліфікаційної роботи: розроблення програмної системи персоналізованих рекомендацій в динамічному режимі на основі аналізу наборів користувацьких даних.

У кваліфікаційній роботі проаналізовано предметну область, визначено вимоги до системи, досліджено методи обробки даних і побудови рекомендацій, розроблено архітектуру та реалізовано ШІ-алгоритм аналізу даних.

Для реалізації програми використовувались такі мови програмування: Python з використанням FastAPI. FAISS, SentenceTransformers. Contextual Bandits для обробки та нормалізації даних.

Систему можна використовувати в електронній комерції, онлайн-освіті та медіасервісах для індивідуалізації контенту й підвищення залученості користувачів.

Результатом проєкту є готове програмне рішення з персоналізованими рекомендаціями за допомогою штучного інтелекту на основі аналізу даних користувачів.

3.05.2025
Дата

Підпис

ПЕРЕЛІК СКОРОЧЕНЬ

BFF	–	Backend for Frontend
CNES	–	Centre National d'Études Spatiales
CPU	–	Central Processing Unit
CRUD-операції	–	Create, Read, Update, Delete
CSS	–	Cascading Style Sheets
DFD	–	Data Flow Diagram
E2E	–	End-to-End testing
FAISS	–	Facebook AI Similarity Search
IDE	–	Integrated Development Environment
ЛР-моделі	–	Логіко-рівнева модель
ORM	–	Object-Relational Mapping
REST API	–	Representational State Transfer
SQL	–	Structured Query Language
СУБД	–	Система управління базами даних
UI/UX	–	User Interface/User Experience
VS	–	Visual Studio
VW	–	Vowpal Wabbit

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ. 2101095.01.21.ПЗ	Пояснювальна записка	90		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
			Графічні документи			
4	А4	КвРІПЗ. 2101095.01.21.ПЗ	DFD діаграма	2		
5	А4	КвРІПЗ. 2101095.01.21.ПЗ	Архітектура системи	1		
6	А4	КвРІПЗ. 2101095.01.21.ПЗ	Діаграма станів	1		
7	А4	КвРІПЗ. 2101095.01.21.ПЗ	ER-діаграма	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ. 2101095.01.21.ПЗ		
Виконала		Чернушина С.О.		30.05.25	Програмна система персоналізованих рекомендацій на основі аналізу користувацьких даних	Літ.	Арк. 6 / Акрушів 90
Керівник		Бедратюк Г.І.		30.05.25			
Рецензент		Павлєва О.О.		10.05.25			
Н. Контр.		Праворська Н. І.		30.05.25		ХНУ. ІПЗ-21-1	
Зав. каф.		Бедратюк Л.П.		30.05.25			

ЗМІСТ

ВСТУП

З кожним днем попит на штучний інтелект зростає, і на разі він вже існує на кожному кроці, таким чином якщо раніше користувачі шукали інформацію в Google, тоді на даний час однакову саму інформацію, яку ви шукаєте можливо запитати у будь-якого ШІ, і він одразу надасть відповідь на питання. Дана тема зводиться до того що, користувачі завжди знаходять ресурс який є легким в управлінні, ефективним та швидким.

Це призводить до використання певних алгоритмів в повсякденному житті, адже користувачі обирають найшвидший та найкращий шлях для вирішення своїх проблем. Точність та швидкість алгоритмів є ключовими факторами, що визначають ефективність та якість отриманих рішень.

Прикладом якісного, але в певному сенсі погано впливаючого алгоритму є платформа TikTok. Раніше це були прості відео під які користувачі знімали короткі ролики, але допрацювавши даний алгоритм, система TikTok стала набагато складнішою. TikTok заохочує користувачів залишатись надовго, саме через свій підхід та новітній алгоритм, який підвищує рівень дофаміну під час того як користувач переглядає ролики, оскільки мозок отримує задоволення від перегляду коротких відео та відпочинку, дану практику називають також «TikTok brain», що означає звикання мозку до швидких дофамінових викидів і нестача нормальної обробки інформації.

Проблема інформаційного перевантаження та потреба в ефективних механізмах фільтрації контенту стали особливо важливими. Саме тут на допомогу приходять програмні системи персоналізованих рекомендацій. Їхнє основне призначення – аналізувати великі масиви даних про користувацьку поведінку та на основі виявлених патернів пропонувати індивідуальні рекомендації, що з високою ймовірністю зацікавлять конкретного користувача.

Однак, створення по-справжньому ефективної рекомендаційної системи є складним завданням. Багато існуючих рішень або надають статичні рекомендації, що не враховують динаміку змін інтересів користувача, що викликає питання щодо

об'єктивності та релевантності пропозицій. Актуальність даної кваліфікаційної роботи полягає саме у дослідженні та розробці такої програмної системи персоналізованих рекомендацій, яка б функціонувала в динамічному режимі, використовуючи сучасні методи аналізу користувацьких даних та технології штучного інтелекту для забезпечення високої точності та адаптивності.

Метою кваліфікаційної роботи є створення новітньої системи «Skunk», яка адаптовується під користувачів та може аналізувати їх поведінку відносно наборів даних тощо. Також, варто зауважити, що враховуючи інтереси користувачів, система у реальному часі забезпечує їх персоналізованими рекомендаціями.

Для досягнення поставленої мети в роботі вирішуються наступні завдання:

– провести комплексний аналіз предметної області систем персоналізованих рекомендацій;

– дослідити методи аналізу користувацьких даних для ефективного виявлення патернів поведінки та індивідуальних вподобань;

– обґрунтувати вибір архітектури програмної системи та технологічного стеку, що забезпечить її гнучкість, масштабованість та ефективність;

– спроєктувати архітектуру системи, визначивши ключові компоненти: збору даних, їх обробки, моделі генерації рекомендацій та інтерфейсу взаємодії з користувачем;

– реалізувати алгоритми, здатні аналізувати користувацькі дані та формувати релевантні персоналізовані рекомендації в динамічному режимі;

– розробити інтерактивний та інтуїтивно зрозумілий користувацький інтерфейс для взаємодії з системою та надання зворотного зв'язку;

– здійснити всебічне тестування розробленої програмної системи для оцінки її функціональності, точності рекомендацій та продуктивності.

Предметом дослідження є програмні засоби, алгоритмічні моделі та методи, що використовуються для побудови динамічної системи персоналізованих рекомендацій, зокрема методи збору, обробки даних, машинного навчання для генерації рекомендацій та механізми адаптації на основі зворотного зв'язку.

Об’єктом дослідження виступає процес формування та надання персоналізованих рекомендацій користувачам на основі інтелектуального аналізу їхніх даних.

Практичне значення розробленої програмної системи полягає у можливості її застосування для покращення рівня персоналізації у широкому спектрі онлайн-сервісів, таких як платформи електронної комерції, освітні ресурси та інше. Впровадження такої системи дозволить покращити користувацький досвід, збільшити залученість аудиторії та ефективність взаємодії з запропонованим контентом чи продуктами. Розроблений підхід також може слугувати основою для подальших досліджень та створення більш досконалих рекомендаційних сервісів.

Таким чином, дана кваліфікаційна робота спрямована на вирішення актуальної проблеми створення ефективної системи «Skunk» персоналізованих рекомендацій, що відповідає сучасним вимогам до динамічності, точності та адаптивності. Результатом роботи є програмний продукт, що демонструє практичну реалізацію запропонованих підходів та готовий до потенційного впровадження.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області та її особливостей

Часто користувачі діляться особистою інформацією на просторах інтернету або ж у стрічці з'являється персоналізована реклама, це в більшій мірі залежить від алгоритмів, а саме алгоритмів рекомендацій. У соціальних мережах чи інформаційних технологіях такі алгоритми називають – рекомендаційними системами. Рекомендаційні системи – це алгоритми які проводять аналіз щодо того, що подобається користувачам, а саме аналізують вподобання, коментарі, реакції користувачів на певний контент, щоб далі мати змогу обробляти цю інформацію і показувати відповідно вже з наданої та обробленої інформації – персоналізований контент. Тобто, саме рекомендаційні системи вирішують, що ж саме буде з'являтись у стрічці користувача.

Незалежно від того, чи користувачі користуються Spotify, Instagram, Amazon або Netflix, вони кожного дня стикаються з продуктами, які є рекомендованими. У 2017 році Netflix заявив, що його користувачі відкривають для себе 80% серіалів за допомогою алгоритмічних рекомендацій. До прикладу, коли користувач переглядає різні веб-сайти, щоб замовити персоналізований сайт для себе чи свого бізнесу ймовірність, що надалі йому будуть попадатись у стрічці агенції чи конструктори сайтів дуже висока.

Щоб алгоритми працювали належним чином, їм потрібно багато користувацьких даних. Тому не менш значущими є неявні дані, що збираються шляхом пасивного спостереження за поведінкою користувача:

- історія переглядів сторінок;
- здійснені покупки;
- час, проведений за вивченням опису товару чи переглядом відео;
- послідовність кліків, використані пошукові запити.

Одним з найкращим методів рекомендацій є фільтрація на основі вмісту. Вміст тут стосується атрибутів продуктів, які подобаються користувачеві. Ідея

фільтрації на основі вмісту полягає в тому, щоб позначати продукти за допомогою певних ключових слів, розуміти, що подобається користувачеві, шукати ці ключові слова в базі даних і рекомендувати різні продукти з однаковими атрибутами.

Після обробки користувацьких даних, створюється профіль користувача, який потім використовується для пропонування користувачеві, і чим більше користувач надає інформації або виконує більше дій за рекомендацією, тим точнішими стають рекомендації.

У профілі користувача створюються вектори, які описують вподобання користувача. При створенні профілю користувача використовується матриця корисності, яка описує зв'язок між користувачем і продуктом.

Матриця корисності відображає перевагу, яку користувач віддає певним елементам. У даних, отриманих від користувача, варто знайти певний зв'язок між елементами, які подобаються користувачеві, і тими, які йому не подобаються (таблиця 1.1).

Таблиця 1.1 – Матриця корисності

Користувачі	Фільм 1	Фільм 2	Фільм 3
Користувач23	5	1	5
Користувач24	2	4	

Деякі стовпчики в матриці порожні, тому що ми не отримуємо повну інформацію від користувача щоразу, а мета рекомендаційної системи не в тому, щоб заповнити всі стовпчики, а в тому, щоб рекомендувати користувачеві фільм, якому він надасть перевагу.

Оскільки розробка алгоритмів рекомендацій пов’язана зі штучним інтелектом, варто також розглянути, яка саме працює штучний інтелект у сфері динамічних рекомендацій.

Кожний продукт має певний набір характеристик як у таблиці 1.2.

Таблиця 1.2 - Характеристики фільмів.

Фільми	Жанр	Режисер
Interstellar	Фантастика	Космос, Подорож у часі, Чорні Діри
The Irishman	Кримінальна драма	Мафія, Гангстери, Роберт Де Ніро

Системи рекомендацій на основі вмісту створюють регресійну модель на основі користувацьких даних та атрибутів продуктів, щоб рекомендувати товари конкретному користувачеві.

Усі ці продукти та атрибути використовуються під час машинного навчання для моделі штучного інтелекту, яка буде прогнозувати, що на основі вхідних даних надалі пропонувати користувачу.

Як і в кожної моделі системи рекомендацій на основі вмісту мають свої переваги та недоліки. Перевагою є те, що фільтрування на основі вмісту забезпечує більшу ступінь прозорості шляхом надання інтерпретованих функцій, що пояснюють рекомендації. Наприклад, система рекомендацій фільмів може пояснити, чому рекомендується певний фільм, наприклад, жанр чи актор перетинаються з раніше переглянутими фільмами. Тому користувач може прийняти більш обґрунтоване рішення щодо перегляду рекомендованого фільму.

Одним з головних недоліків фільтрації на основі вмісту є обмеження функцій. Рекомендації на основі вмісту випливають виключно з функцій, що використовуються для опису елементів. Функції елементів системи можуть не в змозі зафіксувати те, що подобається користувачеві естетично. Наприклад, користувачеві подобається фільм Interstellar за графікою, дизайном, світловими елементами, але система не в змозі цього розпізнати, тому що неможливо створити такі функції, поки що, які би задовільняли естетичний погляд користувача як у фільмі Interstellar.

Отже, з вищеподаної інформації можливо зробити висновок, що для побудови ефективних рекомендацій важливим джерелом є зворотний зв'язок, коли

користувачі свідомо та цілеспрямовано надають системі оцінки наприклад, рейтинг фільму за п'ятибальною шкалою, залишають текстові відгуки, позначають контент як вподобаний або додають товари до списку бажань. Такі дані є надзвичайно цінними, оскільки безпосередньо відображають суб'єктивне сприйняття.

Важливу роль відіграють і атрибути об'єктів рекомендації – детальні характеристики товарів, фільмів чи будь-якого іншого контенту жанр, ціна, технічні параметри, ключові слова, що є критично важливим для контентно-орієнтованих методів. Нарешті, контекстуальні дані, що описують умови взаємодії час доби, місцезнаходження, тип пристрою, дозволяють надавати значно більш релевантні та своєчасні пропозиції.

Однією з фундаментальних проблем, є так звана проблема «холодного старту». Вона проявляється у двох основних аспектах:

– по-перше, для нових користувачів система ще не накопичила достатньо даних про їхні вподобання, щоб формувати якісні та персоналізовані рекомендації;

– по-друге, для нових об'єктів, щойно доданих до системи, ще немає історії взаємодій та відгуків, що ускладнює визначення цільової аудиторії для їх рекомендації.

У контексті даної кваліфікаційної роботи, де розробляється універсальна програмна система, здатною працювати з різними, завантажуваними користувачем, наборами даних, особливого значення набуває її гнучкість та адаптивність.

Система повинна володіти механізмами, що дозволяють їй не просто аналізувати структуру та зміст наданого датасету для виявлення ключових характеристик об'єктів, але й інтерактивно взаємодіяти з кінцевим користувачем для уточнення його поточних преференцій. Це передбачає динамічну генерацію запитань, що базуються на специфіці конкретного набору даних, та використання отриманих відповідей для побудови максимально точного профілю користувача.

Здатність системи обробляти зворотний зв'язок на запропоновані рекомендації та на основі цього коригувати свої моделі в режимі реального часу є ключовою вимогою для створення справді інтелектуальних та ефективних рекомендаційних сервісів. Таким чином, дослідження та розробка подібної

універсальної та адаптивної платформи становить актуальне та складне інженерне завдання, вирішення якого відкриває нові можливості для персоналізації взаємодії в цифровому середовищі.

1.2 Аналіз наявного програмо-технічного забезпечення

Система рекомендацій - це інструмент, призначений для надання персоналізованих пропозицій користувачам на основі їх уподобань, поведінки та взаємодії з платформою. Ці системи аналізують такі дані, як історія придбання, історія перегляду, демографічна інформація користувачів та контекстна інформація для доставки відповідного вмісту. Нижче наведені найпоширеніші типи систем рекомендацій:

Методи спільної фільтрації:

- засновані на основах взаємодії між користувачами та об'єктами;
- включають спільну фільтрацію на основі користувача і спільну фільтрацію на основі елементів;
- найкраще підходить для електронної комерції.

Методи фільтрації на основі вмісту:

- зосереджуються на атрибутах товарів і минулих взаємодіях користувача;
- визначення конкретних інтересів цільового користувача.

Гібридні рекомендаційні системи:

- поєднують методи спільної фільтрації та контент-орієнтовані методи;
- икористовують моделі машинного навчання, такі як глибокі нейронні мережі та рекурентні нейронні мережі для покращення прогнозів.

Проаналізувавши алгоритми рекомендацій, варто перейти до підходів, реалізовані такими гігантами індустрії, як Netflix та Spotify, а також інноваційною платформою для пошуку роботи Welcome to the Jungle (раніше відомою як Otta). Такий аналіз дозволить виявити загальні тенденції та специфічні для різних доменів особливості, що є важливим для обґрунтування архітектурних та

алгоритмічних рішень при розробці системи «Skunk», яка позиціонується як універсальний інструмент для аналізу довільних наборів даних.

Spotify пропонує рекомендації на основі алгоритмів, які є релевантними, унікальними та специфічними для кожного користувача. Їхній флагманський підхід BaRT є прикладом ефективної комбінації технологій. Тобто даний підхід відбирають і впорядковують контент кожного слухача Spotify, в тому числі в «Пошуку», «Домашньому» та «Персональних» плейлистах.

Щоб мати такий розширений аналіз рекомендацій, Spotify використовує безліч алгоритмів у поєднанні, але найголовніші серед них:

- voyager;
- глибокі нейронні мережі;
- матрична факторизація;
- аналіз звуку;
- обробка природної мови;
- спільна фільтрація;
- фільтрація на основі вмісту.

Ключову роль відіграє обробка природної мови NLP для аналізу текстових даних, пов'язаних з музикою, що дозволяє системі розуміти семантику та настрій композицій. Унікальною особливістю є глибокий аналіз безпосередньо аудіосигналів для виділення таких характеристик, як темп чи тональність, що дає змогу знаходити музично схожі треки.

Для ефективного пошуку схожості використовуються моделі на основі глибокого навчання та векторних представлень. Voyager - це бібліотека для проведення швидких приблизних пошукових запитів найближчих сусідів у колекції векторів, що знаходяться в пам'яті. Він використовує алгоритм HNSW, «Ієрархічні навігаційні малі світи» на основі, з численними функціями, доданими для зручності та швидкості (рисунок 1.1). Spotify алгоритми використовують різні типи даних для надання цих рекомендацій. Важливість цих типів даних може змінюватися з часом, залежно від того, як користувач використовує Spotify.

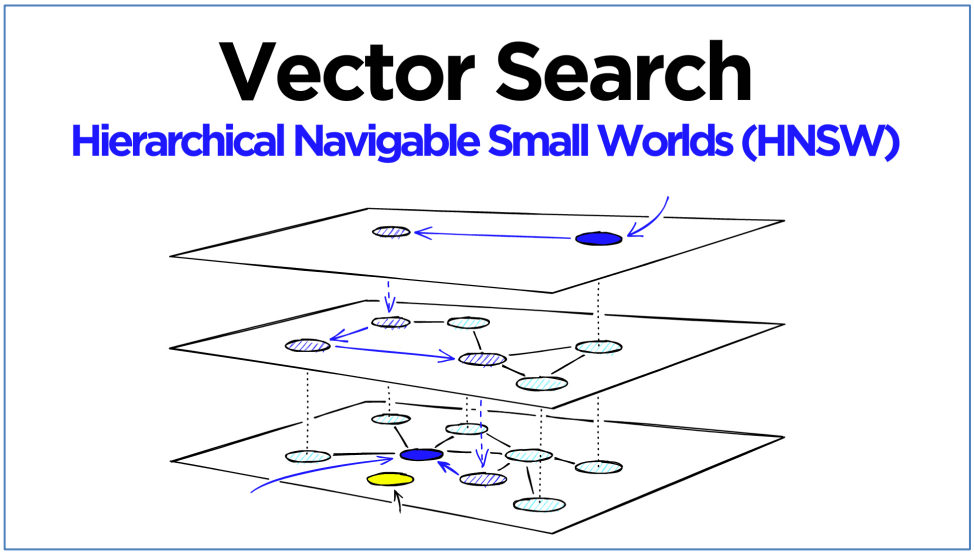

Рисунок 1.1 – алгоритм HNSW

Смаковий профіль користувача є найважливішим типом даних для забезпечення найкращого користувацького досвіду. Профіль смаку це те, що користувач шукає, слухає і пропускає і що зберігає у своїй медіатеці, впливає на інтерпретацію Spotify смаку користувача. Spotify називає це користувацьким смаковим профілем (рисунок 1.2). Смаковий профіль дає алгоритмам уявлення про те, що цікавить людину і що, людина полюбляє слухати (рисунок 1.3).

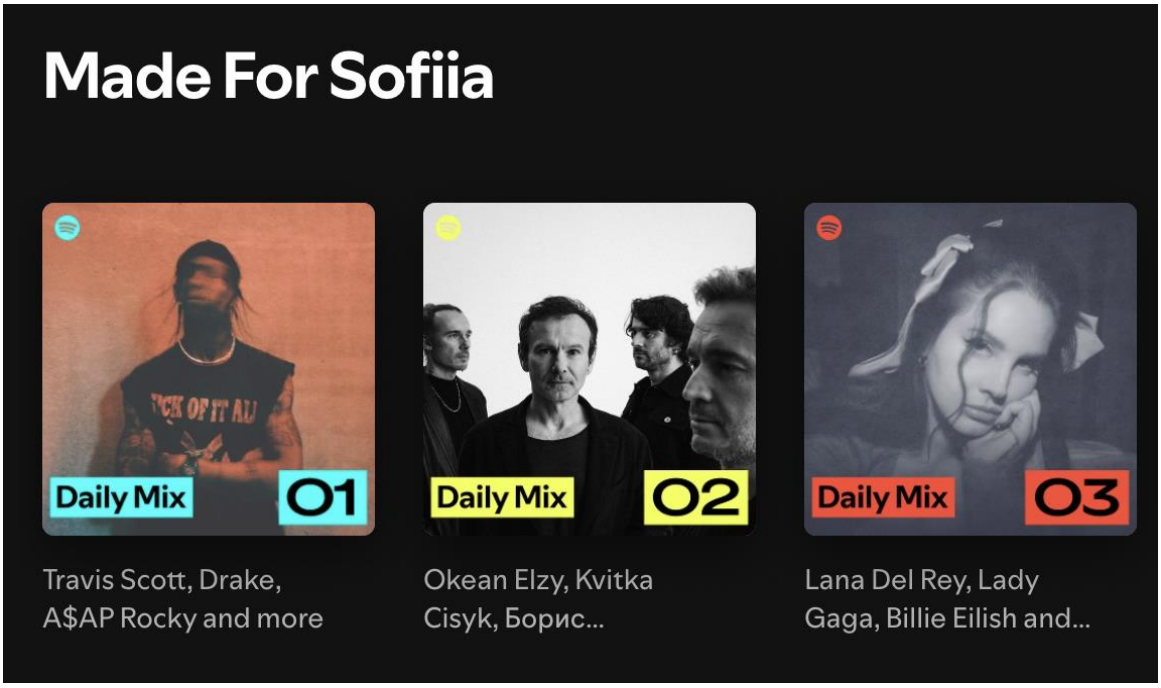

Рисунок 1.2 – Рекомендовані вибірки за смаковим профілем користувача

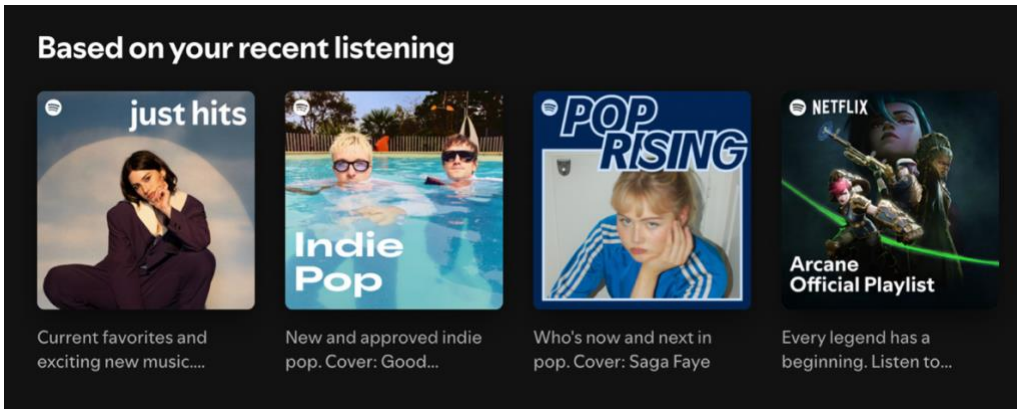

Рисунок 1.3 – Рекомендовані пісні на основі минулих прослуховувань

Отже, можливо виявити значні переваги щодо таких алгоритмів:

– персоналізація;

– масштабованість;

– відкриття нової для користувача музики;

– аналіз контексту.

Серед усіх переваг також є певні недоліки:

– користувачі не розуміють чому іноді їм рекомендують ту чи іншу музику;

– алгоритми можуть лімітувати музику, яка може сподобатись користувачам, тому що вони не настільки розвинуті;

– проблема «холодного старту» для абсолютно нових, ще ніким не прослуханих треків та виконавців.

Наступним у аналізі алгоритмів буде розглядатись веб-сайт Welcome to the Jungle. Welcome to the Jungle це платформа для знаходження роботи, яка вирізняється своїм алгоритмом рекомендацій.

При першому вході на платформу система негайно запускає інтерактивний опитувальник, створений для максимально точного аналізу професійних уподобань користувача. Цей механізм ретельно збирає та аналізує широкий спектр даних, формуючи деталізований цифровий профіль. Welcome to the Jungle враховує ключові аспекти кар'єрних очікувань, включаючи бажану позицію (рисунок 1.4), рівень відповідальності, очікуваний дохід (рисунок 1.5), оптимальний графік роботи та переваги щодо форми зайнятості і місця роботи (рисунок 1.6). Особливу

увагу приділяє аналізу корпоративної культури - система збирає дані про бажану атмосферу в колективі, рівень бюрократії, можливості для професійного зростання та інші важливі для користувача організаційні фактори. Даний аналіз компанії Welcome to the Jungle дозволяє знаходити ідеальні збіги між очікуваннями користувача та реальними пропозиціями на ринку праці, забезпечуючи персоналізований підхід до пошуку роботи.

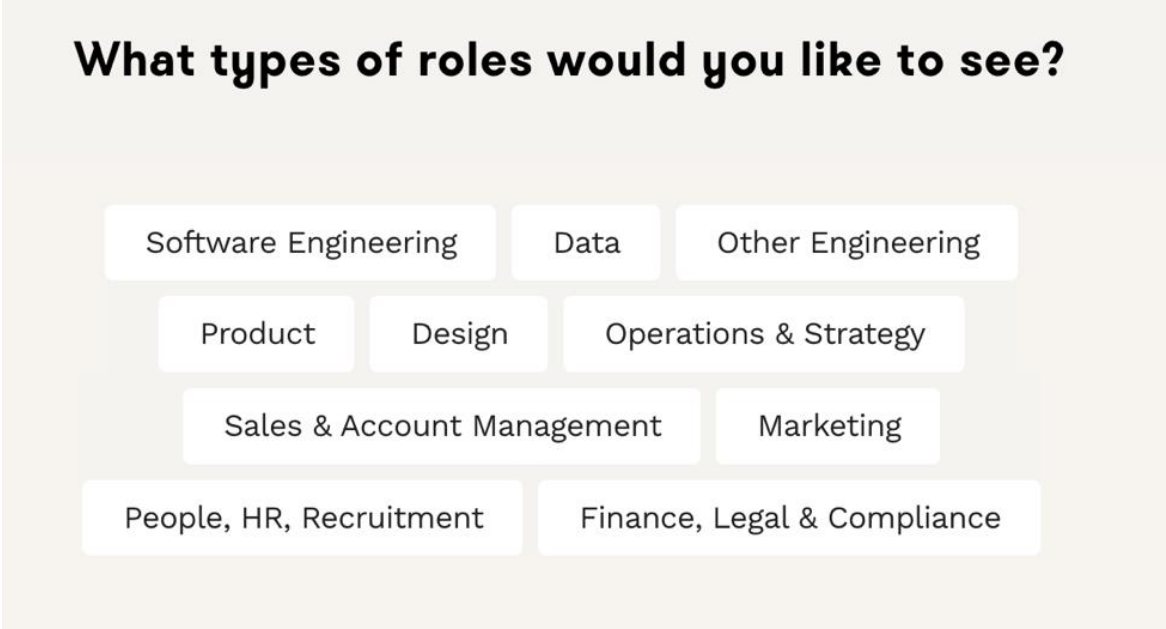

Рисунок 1.4 – Опитування щодо бажаної позиції

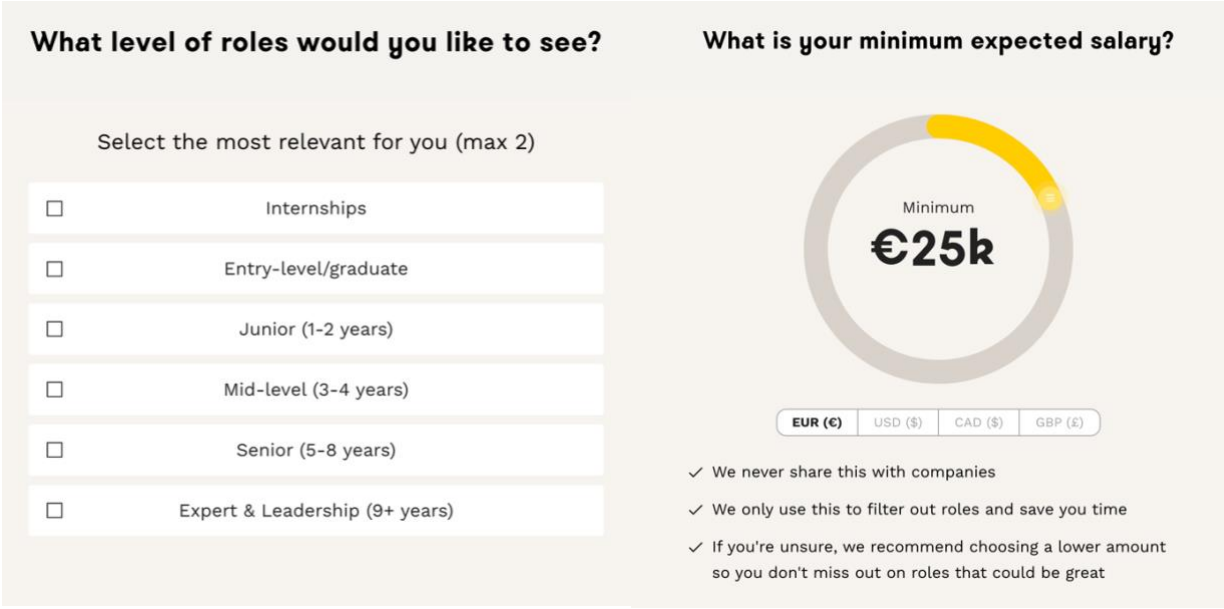

Рисунок 1.5 – Запитання щодо очікуваного доходу

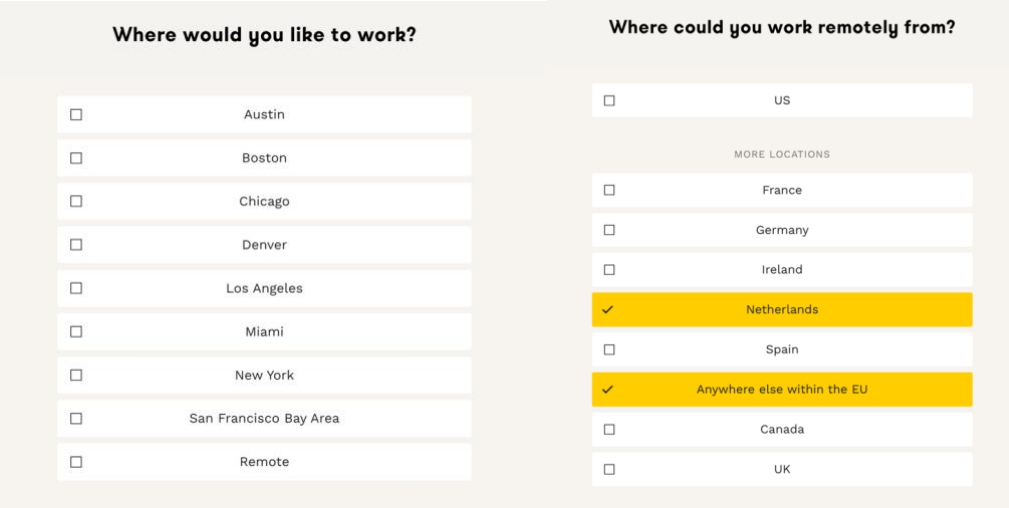

Рисунок 1.6 – Запитання щодо бажаного робочого місця.

Отже, даний алгоритм працює таким чином, що після збору та аналізу інформації з питань, користувачу буде представлено список робіт, відповідно до параметрів та фільтрів, які раніше обрав користувач. Звісно, що потім ці налаштування можливо змінити або ж через певний час, система може запитати знову користувача, чи не змінились його вподобання або вибір щодо працевлаштування, щоб рекомендації працювали коректно.

Перевагами такої системи є:

– динамічність;

– персоналізовані рекомендації;

– прозорість.

Недоліків у даних алгоритмах небагато, а саме:

– обмеженість рекомендацій;

– ігнорування популярних вакансій серед інших кандидатів.

Останнім додатком щодо якого проводиться аналіз є Netflix. Платформа Netflix є одним із найяскравіших прикладів успішного впровадження та монетизації рекомендаційних технологій. Архітектурно виправданим для Netflix стало використання складного гібридного підходу, що чудово поєднує різноманітні методи для досягнення максимальної релевантності та залучення користувачів. Система аналізує величезні масиви даних про поведінку мільйонів передплатників,

виявляючи схожі патерни переглядів та оцінок для реалізації колаборативної фільтрації. Одночасно, для контентно-орієнтованої фільтрації, Netflix детально розбирає метадані кожного фільму та серіалу, включаючи жанр, акторський склад, режисера та ключові слова.

Кожного разу, коли користувач отримує доступ до сервісу Netflix, система рекомендацій шукає з легкістю шоу, фільм або гру, які сподобаються користувачу. Алгоритми оцінюють ймовірність того, що сподобався той чи інший фільм у каталозі, на основі низки факторів, серед яких:

– взаємодія з сервісом;
– інші користувачі зі схожими смаками та вподобаннями на нашому сервісі;
– інформація про фільми.

Системи рекомендацій починає працювати з моменту реєстрації користувача, коли користувач створює свій обліковий запис Netflix або додаєте новий профіль, Netflix пропонує вибрати декілька фільмів, які подобаються. Для подальшої інформації використовуються ці фільми для старту рекомендацій.

Надалі, по мірі того, як користувач продовжуватимете користуватися Netflix, фільми, які він перегляне останнім часом, будуть переважати над тими, які були переглянуті раніше, для того щоб зберігати динамічність системи та підлаштовуватись під неї.

Важливу роль відіграють алгоритми на основі глибокого навчання, зокрема нейронні мережі, які дозволяють моделювати складні нелінійні залежності та послідовності переглядів для передбачення майбутніх інтересів (рисунок 1.7).

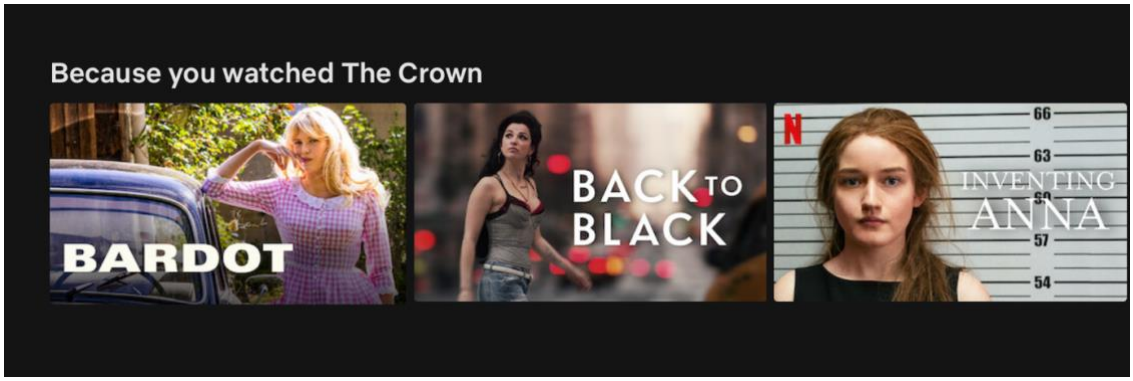

Рисунок 1.7 – Рекомендації Netflix на основі переглянутих фільмів

Аналізуючи розглянуті програмно-технічні рішення (таблиця 1.3), можна констатувати, що сучасні рекомендаційні системи є складними, багатокомпонентними продуктами. Вони все частіше рухаються в напрямку гібридизації підходів, поєднуючи сильні сторони різних алгоритмів.

Таблиця 1.3 Спеціалізовані алгоритми системи

Характеристика	Системи	
	Spotify	Netflix
Основні методи	Гібридний метод (колаборативна фільтрація, NLP, аналіз аудіо, векторні представлення HNSW)	Гібридний метод (колаборативна фільтрація, глибоке навчання)
Ключові дані	Історія прослуховувань, плейлисти, текстовий контент про музику, аудіо характеристики	Історія переглядів, оцінки, метадані фільмів, поведінкові патерни
Холодний старт	Залишається на високому рівні і часто є проблемою	Пом’якшується початковим вибором
UI/UX	На високому рівні інтуїтивно зрозумілий та сучасний	На високому рівні інтуїтивно зрозумілий

Критично важливим є використання різноманітних джерел даних та здатність систем до динамічної адаптації до змін у вподобаннях користувачів. Зростає також значення прозорості та пояснюваності рекомендацій, оскільки користувачі прагнуть розуміти логіку пропозицій. Проблеми «холодного старту» та розрідженості даних залишаються актуальними дослідницькими викликами.

Проведений аналіз показує, що хоча існують потужні та ефективні рекомендаційні системи, спеціалізовані для конкретних доменів, залишається простір для розробки більш універсальних рішень. Система «Skunk», що розробляється в рамках даної кваліфікаційної роботи, має на меті запропонувати саме такий унікальний підхід. Її ключова відмінність полягає в універсальності та здатності адаптуватися до довільних наборів даних, наданих користувачем.

Замість того, щоб бути обмеженою попередньо визначеним типом контенту, «Skunk» використовує інтелектуальні механізми для аналізу структури та змісту

завантаженого датасету. На основі цього аналізу система динамічно формулює запитання до користувача, спрямовані на виявлення його індивідуальних преференцій стосовно характеристик, властивих саме цьому конкретному набору даних. Такий інтерактивний підхід дозволяє:

– застосовувати систему до широкого кола предметних областей без необхідності її модифікації чи перенавчання для кожного нового типу даних;

– ефективно адресувати проблему «холодного старту» для нових, невідомих системі датасетів;

– Забезпечити високий ступінь залученості користувача, який стає активним учасником процесу формування критеріїв для майбутніх рекомендацій.

Отримавши відповіді, програмна система використовує їх для побудови профілю користувача та надає персоналізовані пропозиції з завантаженого датасету. Передбачається також, що система буде здатна навчатися на основі зворотного зв'язку, динамічно покращуючи якість своїх рекомендацій з часом. Таким чином, порівняно з розглянутими аналогами, які є переважно високоспеціалізованими, розроблювана система пропонує гнучку та інтерактивну платформу, що відкриває нові можливості для надання персоналізованих рекомендацій у найрізноманітніших контекстах, де користувач сам визначає джерело даних.

1.3 Визначення вимог до програмного забезпечення

На основі проведеного аналізу предметної області, дослідження існуючих програмно-технічних рішень та з урахуванням специфіки проектованої системи персоналізованих рекомендацій «Skunk», яка орієнтована на роботу з довільними наборами даних, завантаженими користувачем, формулюються наступні вимоги до програмного забезпечення. Ці вимоги поділяються на функціональні, що описують конкретні дії, які система повинна виконувати, та нефункціональні, які визначають якісні характеристики та обмеження системи. Функціональні вимоги визначають основні можливості, які програмна система повинна надавати користувачам для

досягнення поставленої мети – генерації персоналізованих рекомендацій на основі аналізу завантажених даних.

Функціональні вимоги:

– Можливість зареєстрованим користувачам завантажувати власні набори даних у визначених форматах через інтуїтивно зрозумілий інтерфейс;

– Проведення первинного аналізу структури та змісту датасету для виявлення потенційних ознак;

– Перегляд списку завантажених датасетів та їх видалення за потреби;

– Автоматична генерація запитань на основі аналізу датасету;

– Різноманітність типів запитань, адаптованих до колонок датасету;

– Зручний інтерфейс для надання відповідей на згенеровані запитання;

– Збереження відповідей у профілі користувача;

– Застосування відповідних алгоритмів для генерації списку рекомендацій на основі відповідей користувача;

– Врахування преференцій і внутрішніх характеристик об'єктів у датасеті;

– Чіткий, зрозумілий формат представлення рекомендацій;

– Надання пояснень або критеріїв для кожної рекомендованої позиції;

– Можливість оцінити запропоновані рекомендації за механізмом «подобається»/»не подобається»/«нейтрально»;

– Використання зібраного зворотного зв'язку для поступового покращення якості майбутніх рекомендацій.

Нефункціональні вимоги:

– Час генерації запитань після аналізу датасету не повинен перевищувати 30-60 секунд для датасетів до 10 МБ;

– Ефективна обробка датасетів до 50-100 тисяч записів;

– Стабільне функціонування з мінімізацією збоїв та помилок;

– Коректна обробка помилок з інформативними повідомленнями;

– Інтуїтивно зрозумілий, логічний інтерфейс для користувачів різного рівня технічної підготовки;

– Проста та послідовна навігація з чіткими інструкціями та підказками;

– Безпечне зберігання даних користувачів;

– Захищені канали для взаємодії з зовнішніми API;

– Добре структурований код для спрощення подальшої підтримки;

– Модульна гібридна архітектура для незалежної розробки та оновлення окремих компонентівв системі;

– Коректне відображення та функціонування клієнтської частини в останніх версіях веб-браузерів Chrome, Brave, Safari, Arc.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Архітектура програмної системи

Вибір архітектури програмної системи є одним із найбільш відповідальних та фундаментальних етапів проєктування, оскільки саме архітектурні рішення закладають основу для майбутньої функціональності, надійності, масштабованості та підтримуваності продукту. Для системи персоналізованих рекомендацій «Skunk», яка має обробляти різноманітні користувацькі дані та забезпечувати динамічну адаптацію, вибір оптимальної архітектури набуває особливої ваги.

При розгляді можливих архітектурних підходів для системи було проаналізовано декілька поширених архітектур. Зазвичай схожі системи з рекомендаціями використовують одну з трьох архітектур:

- монолітна архітектура;
- мікросервісна архітектура;
- гібридна архітектура.

Монолітна архітектура об’єднує всі компоненти системи в одну програму, тобто весь код, необхідний для програми зберігається в одному центральному місці. Наразі, рідко коли використовується монолітна архітектура, тому що дану архітектуру важко інтегрувати в новітні системи. Зазвичай якщо потрібно додавати нові функції або технології, монолітний додаток має бути повністю переобладнаний, щоб прийняти нове доповнення.

Мікросервісна архітектура складається з набору незалежних сервісів (рисунок 2.1). Кожен автономний сервіс повинний виконувати окрему функцію.

Перевагами даної архітектури є:

- масштабованість;
- незалежний розвиток;
- стійкість до помилок.

Однак, найбільший недолік мікросервісної архітектури це великі витрати на підтримку інфраструктури, що є недоречним, оскільки якщо в майбутньому буде

плануватись розвиток системи рекомендацій «Skunk», підтримка такої архітектури буде вимагати значних витрат.

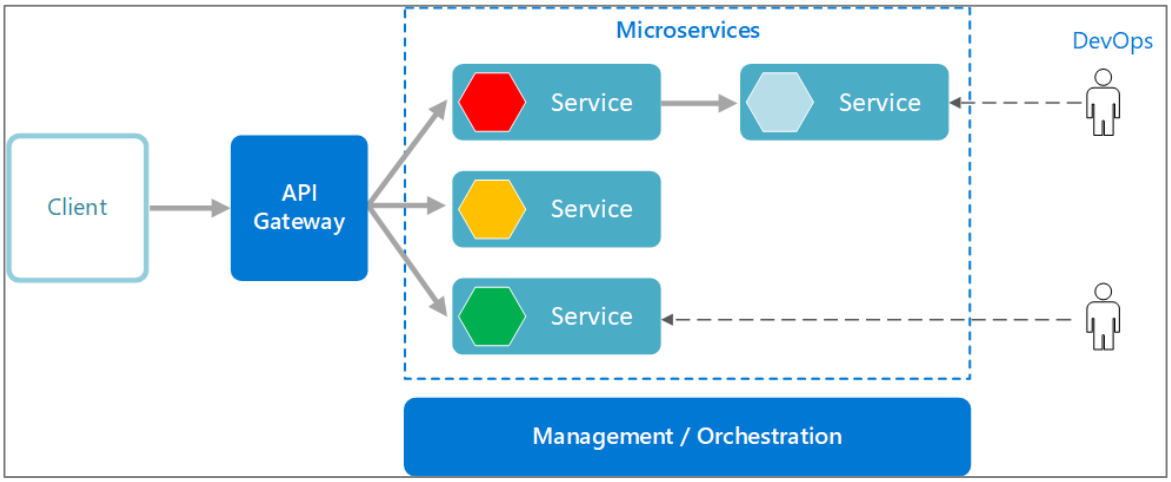

Рисунок 2.1 – Схема мікросервісної архітектури

Гібридна архітектура є поєднанням двох архітектур, а саме мікросервісної та монолітної. Такий підхід забезпечує зручність тим, що основну логіку системи обробляє основний застосунок, а складні процеси винесені в окремі сервіси.

Гібридна архітектура має безліч переваг:

– малі витрати на підтримку інфраструктури;
– гнучкість у використанні технологій програмування;
– можливість делегування складних задач.

Програмна система персоналізованих рекомендацій має мати можливість обробляти велику кількість користувачів одночасно, окремі функції та сервіси повинні бути автономними та ізольованими задля подальшої підтримки та впровадження нових технологій, швидку обробку запитів і забезпечувати захист.

Так, як система рекомендацій повинна бути готовою для масштабування і легкою в управлінні, варто використовувати гібридної багаторівневої архітектури з виокремленням спеціалізованого мікросервісу для ядра рекомендаційної логіки. Такий підхід дозволяє поєднати переваги структурованості та керованості, властиві багаторівневим системам, з гнучкістю та можливістю використання оптимальних

технологій для різних завдань, що характерно для мікросервісного підходу стосовно найбільш обчислювально-складних компонентів.

Обрана архітектура передбачає наступні логічні рівні (рисунок 2.2):

– frontend відповідає за всю взаємодію з кінцевим користувачем, відображення інтерфейсу та збір вхідних даних;

– backend-for-Frontend (BFF) слугує проміжним шаром, який оптимізує взаємодію між фронтендом та різними бекенд-сервісами, агрегує дані та керує автентифікацією. Цей шар також може брати на себе частину логіки, пов'язаної з підготовкою даних для зовнішніх інтелектуальних сервісів;

– основний бекенд реалізований у вигляді спеціалізованого мікросервісу рекомендацій. Він інкапсулює складну логіку аналізу даних, застосування алгоритмів машинного навчання та формування персоналізованих пропозицій;

– рівень даних об'єднує всі сховища, що використовуються системою, включаючи реляційну базу даних для зберігання структурованої інформації та хмарне сховище для файлів датасетів;

– рівень зовнішніх сервісів включає в себе сторонні платформи та API, з якими інтегрується система для виконання специфічних завдань, таких як інтелектуальний аналіз даних чи управління автентифікацією.

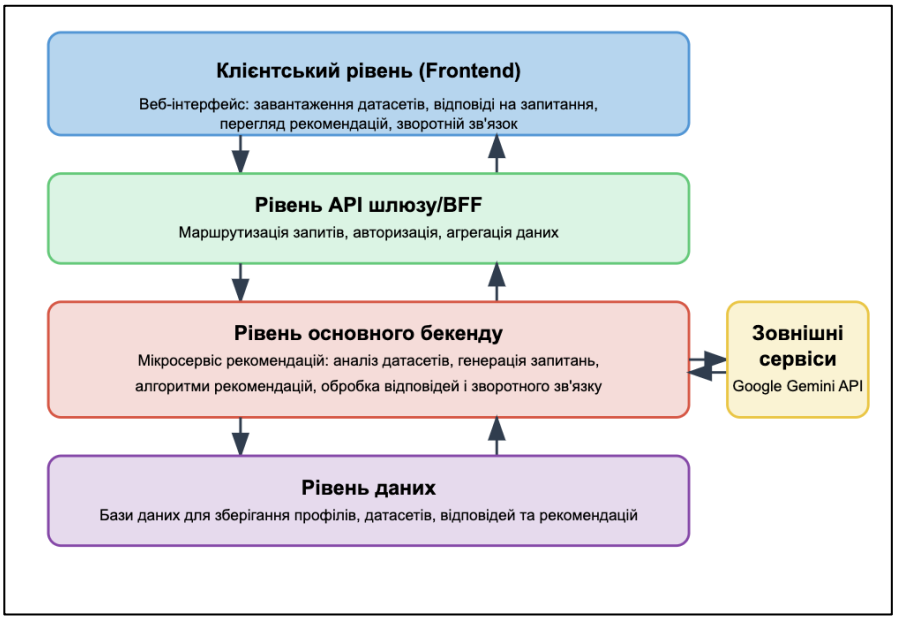

Рисунок 2.2 – Загальна архітектура системи «Skunk»

Переваги обраного гібридного багаторівневого підходу полягають у чіткому розділенні відповідальностей, що в подальшому спрощує розробку, тестування та підтримку системи. Він дозволяє гнучко обирати технології для кожного рівня відповідно до його завдань.

Виокремлення ядра рекомендаційної логіки в мікросервіс забезпечує можливість його незалежного розвитку та масштабування, не впливаючи на інші частини системи. Водночас, наявність BFF-шару дозволяє оптимізувати взаємодію між фронтендом та цим мікросервісом, а також іншими зовнішніми системами.

Таким чином, дана архітектура створює збалансовану основу для реалізації функціональної, надійної та гнучкої системи персоналізованих рекомендацій.

2.2 Декомпозиція системи та її модулів

Після визначення загальної архітектури системи «Skunk», що була детально розглянута у попередньому підрозділі, наступним логічним кроком є проведення її декомпозиції на функціональні модулі. Цей процес є важливим для ефективного управління складністю системи, оскільки дозволяє розбити велику задачу на менші, більш керовані та логічно завершені частини.

Кожен модуль, має чітко визначені межі відповідальності та інтерфейси взаємодії, може розроблятися, тестуватися та модифікуватися відносно незалежно. Важливо зазначити, що вдала декомпозиція є запорукою не лише спрощення процесу розробки, але й підвищення загальної підтримуваності, надійності та гнучкості програмного продукту, що особливо актуально для систем, які, подібно до «Skunk», передбачають інтеграцію різнорідних технологій та потенціал для подальшого розвитку.

Метою декомпозиції системи персоналізованих рекомендацій є виділення ключових функціональних блоків, що відповідають за реалізацію основних етапів обробки даних та взаємодії з користувачем – від моменту завантаження ним датасету до отримання фінальних персоналізованих пропозицій та надання зворотного зв'язку. Такий підхід дозволяє краще зрозуміти внутрішню структуру

системи, визначити основні потоки між її компонентами та оцінити взаємозв'язки між даними компонентами.

Для візуалізації цих аспектів варто використовувати діаграми потоків даних DFD, які наочно демонструють, як «Skunk» взаємодіє зі своїм зовнішнім оточенням та як дані трансформуються всередині неї.

На контекстному рівні DFD, програмна система розглядається як єдиний, цілісний процес (рисунок 2.3). Ця діаграма окреслює межі системи та її фундаментальні інформаційні зв'язки із зовнішніми сутностями. Ключовим зовнішнім актором, безумовно, є «Користувач», який ініціює роботу системи, надаючи їй вхідні дані у вигляді завантажених датасетів та відповідей на уточнюючі запитання, що формують його профіль смаків та отримуючи у відповідь персоналізовані рекомендації та згенеровані запитання.

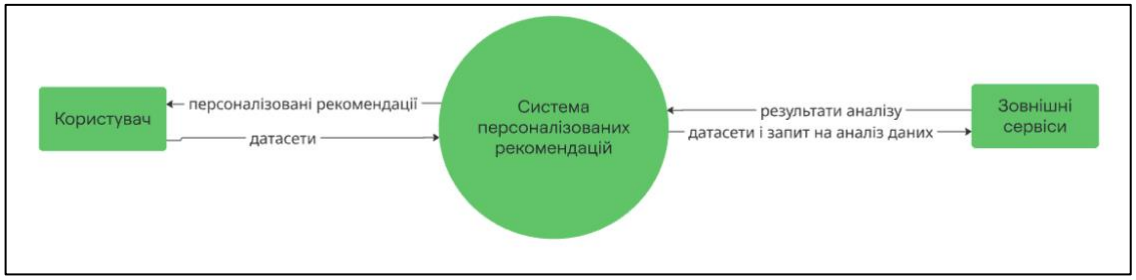

Рисунок 2.3 – DFD нульового рівня

Окремим напрямком взаємодії є передача даних до «Зовнішніх сервісів», що включає показники ефективності, статистику взаємодії користувачів та інше. На цьому рівні абстракції усі зовнішні технологічні компоненти, такі як хмарні сховища, ШІ-моделі та системи аналітики, розглядаються узагальнено як частина зовнішнього середовища системи без деталізації їх внутрішньої архітектури чи специфіки реалізації.

Окрім користувача, система активно взаємодіє з іншими важливими зовнішніми сутностями:

– інтелектуальним сервісом Google Gemini API, який залучається для глибокого аналізу даних та динамічної генерації запитань;

– хмарним сховищем Supabase Storage, призначеним для надійного зберігання файлів датасетів;

– та системою автентифікації Clerk, яка відповідає за управління доступом та ідентифікацію користувачів у системі.

Тому щоб побачити як саме працює архітектура системи необхідно зробити діаграму DFD першого рівня (рисунок 2.4).

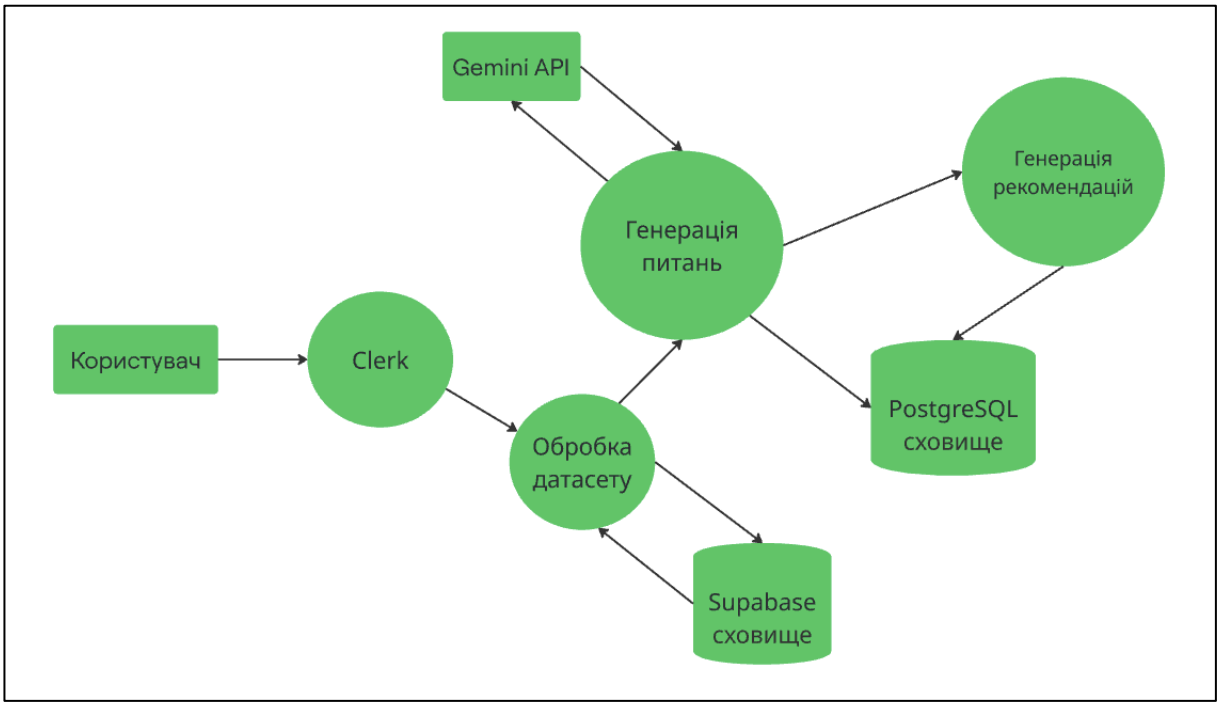

Рисунок 2.4 – DFD першого рівня

Варто зазначити, що програма має гібридну архітектуру, яка поєднує різні підходи та компоненти, що робить повне відображення усіх внутрішніх процесів на одній високорівневій діаграмі надзвичайно складним або неможливим.

Тому діаграма потоків даних першого рівня демонструє архітектуру системи рекомендацій, яка забезпечує послідовну обробку даних від моменту автентифікації користувача до генерації кінцевих рекомендацій.

Взаємодія з системою починається з «Користувача», який виступає зовнішньою сутністю та ініціює процес шляхом надсилання запиту своїх облікових даних. Ці дані надходять до процесу «Clerk», що функціонує як спеціалізована

система автентифікації та управління доступом. «Clerk» виконує верифікацію особи користувача, перевіряючи достовірність наданих облікових даних та визначаючи рівень доступу.

Після успішної автентифікації «Clerk» генерує інформацію про авторизованого користувача, яка містить необхідні метадані та дозволи. Ця інформація передається до наступного процесу обробки датасету. Процес обробки датасету відіграє ключову роль у підготовці вихідних даних для подальшого аналізу. Отримавши інформацію про авторизованого користувача від «Clerk», цей процес встановлює зв'язок із Supabase сховищем, спеціалізованим репозиторієм, що забезпечує надійне зберігання датасетів до 5 ГБ.

Процес виконує витяг необхідних даних із сховища, після чого здійснює комплексну обробку, що може включати трансформацію, очищення, нормалізацію та інші операції, необхідні для підготовки якісного аналітичного матеріалу. В результаті формується підготовлений для аналізу датасет, який передається до наступного етапу обробки.

Центральним інтелектуальним компонентом системи виступає процес генерації питань. Цей процес отримує оброблені дані від попереднього етапу та встановлює взаємодію із зовнішньою сутністю, «Gemini API», що забезпечує доступ до потужних алгоритмів машинного навчання та аналітичних інструментів.

До «Gemini API» надсилається запит на генерацію питань, що містить оброблені дані для аналізу. У відповідь система отримує результати аналізу, які включають сформовані елементи питань. Ці питання генеруються динамічно на основі специфіки наданих даних, забезпечуючи персоналізований та контекстуально релевантний результат.

Згенеровані питання зберігаються у «PostgreSQL» сховищі для подальшого використання та одночасно передаються до наступного процесу в системі. Це забезпечує збереження проміжних результатів аналізу та можливість їх повторного використання у майбутньому.

Завершальним етапом обробки даних є процес генерації рекомендацій. Цей процес приймає сформовані питання від попереднього етапу та використовує їх як

основу для формування комплексних рекомендацій. Рекомендації створюються з урахуванням результатів аналізу даних та відповідей на згенеровані питання.

Сформовані рекомендації зберігаються у «PostgreSQL» сховищі, яке функціонує як центральна база даних системи. Саме це сховище забезпечує постійне зберігання як проміжних результатів так і кінцевих вихідних даних, тобто згенерованих рекомендацій, гарантуючи їх доступність для відображення користувачеві або подальшого використання.

Таким чином «Skunk» демонструє ефективну інтеграцію різноманітних компонентів, забезпечуючи безперервний потік даних, що трансформуються від початкового запиту до кінцевого результату. Взаємодія між процесами побудована за принципом послідовної обробки, де кожен наступний етап використовує результати попереднього.

Представлена діаграма станів відображає життєвий цикл програмного застосунку «Skunk», який надає користувачам рекомендації на основі їхніх вподобань та взаємодій з системою (рисунок 2.5). Діаграма починається з початкового стану чорна точка, який веде до стану «Авторизація». Це вхідна точка в систему, куди користувач потрапляє при запуску застосунку. Від стану «Авторизація» можливі два шляхи розділення станів:

– якщо користувач вже зареєстрований, перехід здійснюється до стану «Існуючий користувач»;

– якщо користувач не зареєстрований, «Skunk» переходить до стану «Новий користувач».

Стан «Новий користувач» призначений для реєстрації нових користувачів у системі. Після успішної реєстрації відбувається перехід до стану «Опитування», де система генерує запитання для визначення вподобань нового користувача. Стан «Існуючий користувач» дозволяє користувачу взаємодіяти з системою через «Swipe картки», де відбувається вибір карток з рекомендаціями. Результати як з «Опитування», так і зі «Swipe карток» зберігаються в стані «Збереження даних», яке являється централізованим сховищем відповідей та вподобань користувачів. Незалежно від того, чи був користувач новим, чи існуючим, після етапу

«Збереження даних» програма переходить до стану «Розрахунок рекомендацій». У цьому стані здійснюється аналіз накопиченої інформації про користувача за допомогою алгоритмів з метою формування персоналізованих рекомендацій.

Отримані рекомендації виводяться користувачу в стані «Показ результатів», який є останнім етапом взаємодії з системою після чого завершується сесія, що позначено кінцевим станом. Завершальним етапом циклу активної взаємодії є стан «Показ результатів», де згенеровані рекомендації надаються користувачеві для перегляду. Після цього стан переходить до кінця, що сигналізує про завершення поточного сеансу взаємодії.

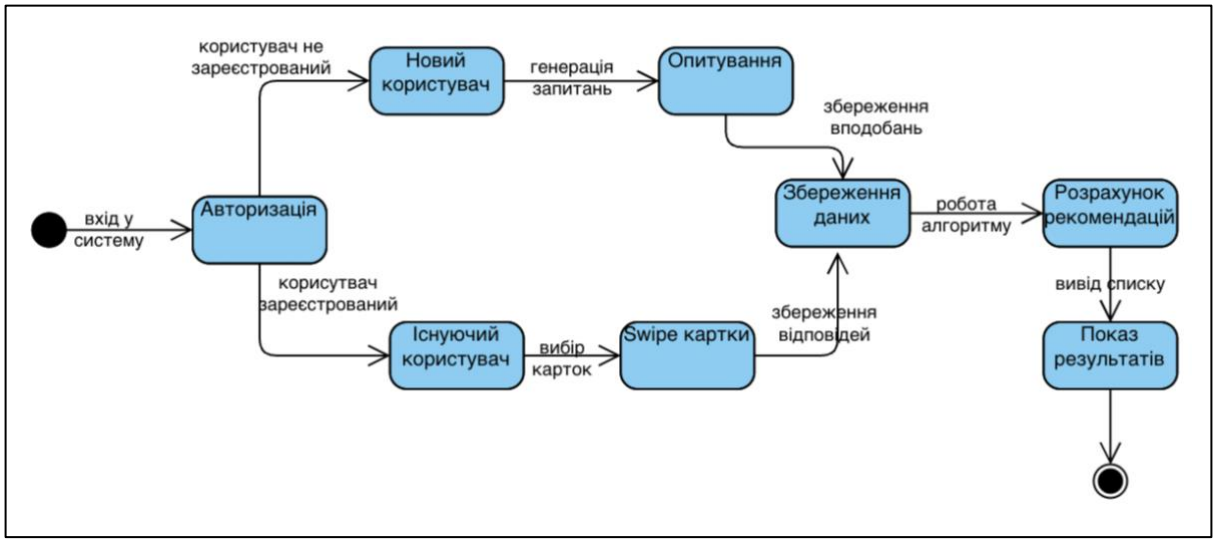

Рисунок 2.5 – Діаграма станів

Переходи між станами включають:

- вхід у систему;
- користувач зареєстрований;
- користувач не зареєстрований;
- генерація запитань;
- збереження вподобань;
- вибір карток;
- збереження відповідей;
- робота алгоритму;

Таким чином, представлена діаграма станів всебічно описує поведінку системи, демонструючи логічні переходи між різними функціональними етапами, вони забезпечують обробку даних користувача та генерацію персоналізованих рекомендацій, адаптованих до його статусу.

2.3 Аналіз та вибір бази даних

Зазвичай, при виборі бази даних, враховуються різні аспекти, але одні з найважливіших аспектів є надійність та здатність швидко й ефективно обробляти великі обсяги даних. З розвитком великих даних компанії тепер мають доступ до безпрецедентних обсягів даних. Програмне забезпечення для керування базами даних є необхідним для зберігання та керування цінними даними організації . СУБД дозволяють легко додавати, змінювати та видаляти дані, усуваючи численні записи та зменшуючи помилки. Тому вибір правильної бази даних для програмних систем стає стратегічним пріоритетом для будь-якої організації. На рисунку 2.5 показано діаграму яка відповідає за найбільш поширені СУБД за 2024 рік.

Як можливо помітити, на першому й другому місцях зазвичай перебуває PostgreSQL і MySQL (рисунок 2.6). Обрані бази даних, користуються таким попитом тому що, вони є надійними, ефективними.

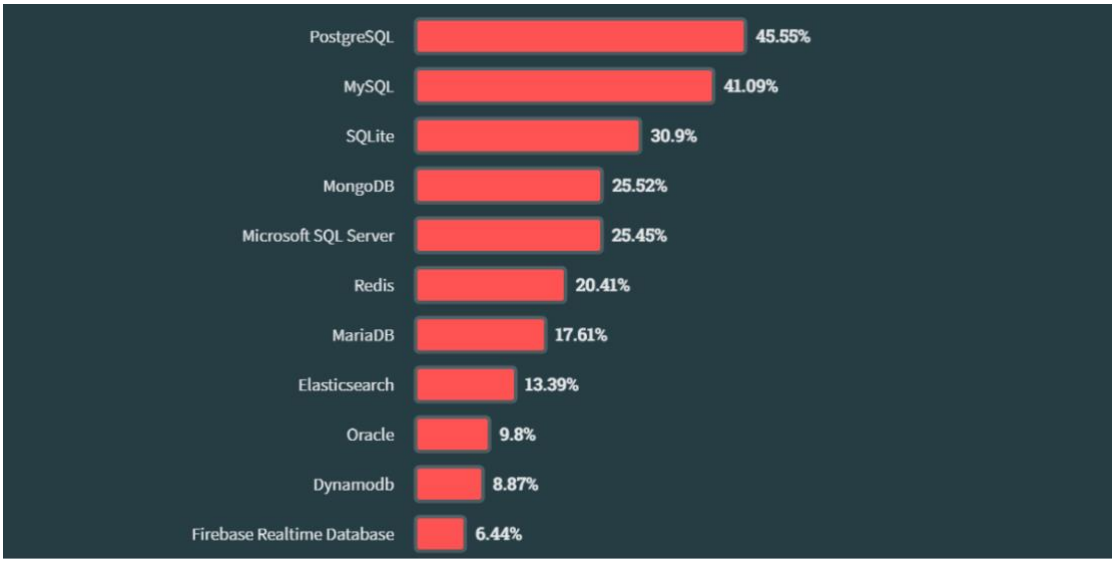

Рисунок 2.6 – 10 найкращих СУБД

Також все більшої популярності набуває MongoDB, завдяки своїй гнучкій моделі даних. MongoDB користується документно-орієнтовану модель даних, де інформація зберігається у форматі JSON, тому це ідеально підходить для збереження динамічних або змінних даних (рисунок 2.7).

Рисунок 2.7 – зберігання даних у форматі JSON

MySQL є однією з найпопулярніших баз даних з відкритим кодом , яка широко використовується для веб-додатків завдяки своїй високій продуктивності та надійності (рисунок 2.8). Безкоштовна версія MySQL пропонує достатню швидкість і надійність для багатьох програм, тоді як платна версія призначена для організацій з особливими потребами.

Його активно застосовують у платформах електронної комерції Magento, WooCommerce та інші для керування товарами та замовленнями, а також як бекенд для багатьох мобільних додатків. Крім того, MySQL знаходить своє застосування в корпоративних додатках, включаючи ERP- та CRM-системи, де потрібне швидке розгортання та економічно ефективне рішення.

MySQL є основою для веб-сайтів та веб-додатків, зокрема для CMS:

– WordPress;

– Joomla;

– Drupal.

Переваги MySQL:

– Безкоштовна модель із відкритим вихідним кодом;

– Простота його використання;

– Швидке встановлення.

Недоліки MySQL:

– Проблеми з продуктивністю коли є великими наборами даних;

– Складне керування пам’ятю.

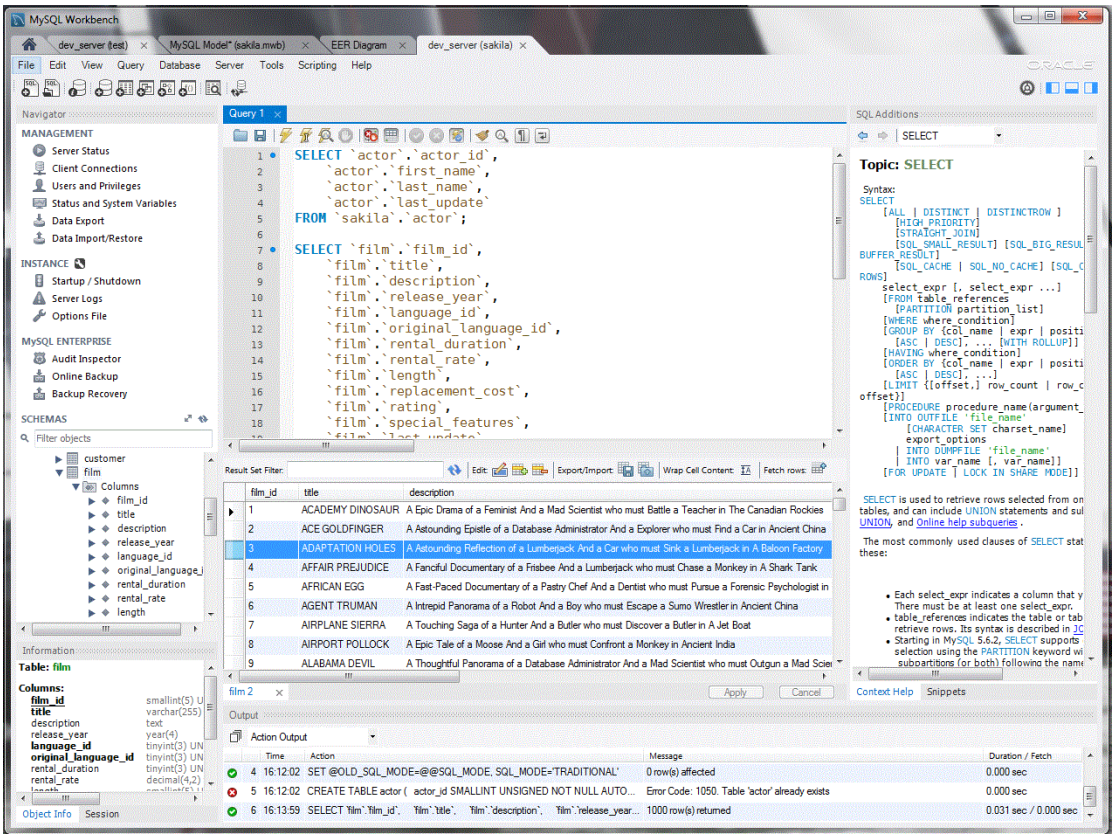

Рисунок 2.8 – MySQL.

PostgreSQL відомий своєю потужністю, гнучкістю та відповідністю сучасним стандартам баз даних. Система відома своєю здатністю ефективно працювати з

різноманітними типами даних, включаючи JSON, що робить її ідеальним вибором для сучасних застосунків, які потребують обробки напівструктурованих даних.

PostgreSQL відзначається високим рівнем надійності, забезпечуючи цілісність даних навіть у критичних умовах роботи, що пояснює її широке використання в багатьох великих компаніях та організаціях. Компанії, які використовують PostgreSQL:

- SNCF;
- Air France;
- Кооперативний банк Casden;
- CNES.

Серед основних переваг PostgreSQL можна виділити не лише гнучку структуру даних, безпеку, надійність, високу продуктивність, кросплатформеність та масштабованість, але й повну відповідність принципам ACID, а саме:

- атомарність;
- узгодженість;
- ізольованість;
- довговічність.

PostgreSQL також пропонує багатий набір функцій для програмування на стороні сервера, включаючи збережені процедури, тригери та правила, що дозволяє реалізувати складну бізнес-логіку безпосередньо в базі даних. В даному проекті буде використовуватись база даних PostgreSQL, оскільки програмна система рекомендацій розробляється на MacOS, і не завжди можливо знайти СУБД, яка буде коректно працювати саме з цією операційною системою.

Для ефективної роботи з PostgreSQL широко використовується pgAdmin 4 (рисунок 2.9). Повнофункціональний інструмент адміністрування та розробки з графічним інтерфейсом, який значно спрощує взаємодію з базою даних. pgAdmin 4 представляє собою сучасний та інтуїтивно зрозумілий інтерфейс для керування базами даних PostgreSQL, що працює на всіх основних платформах, включаючи macOS, Windows та Linux. Він може бути встановлений як настільний додаток або розгорнутий як веб-застосунок, що забезпечує гнучкість у розгортанні та

використанні. pgAdmin 4 надає широкий спектр функцій для адміністрування баз даних, включаючи створення та управління базами даних, таблицями, індексами, обмеженнями, тригерами та іншими об'єктами через зручний графічний інтерфейс.

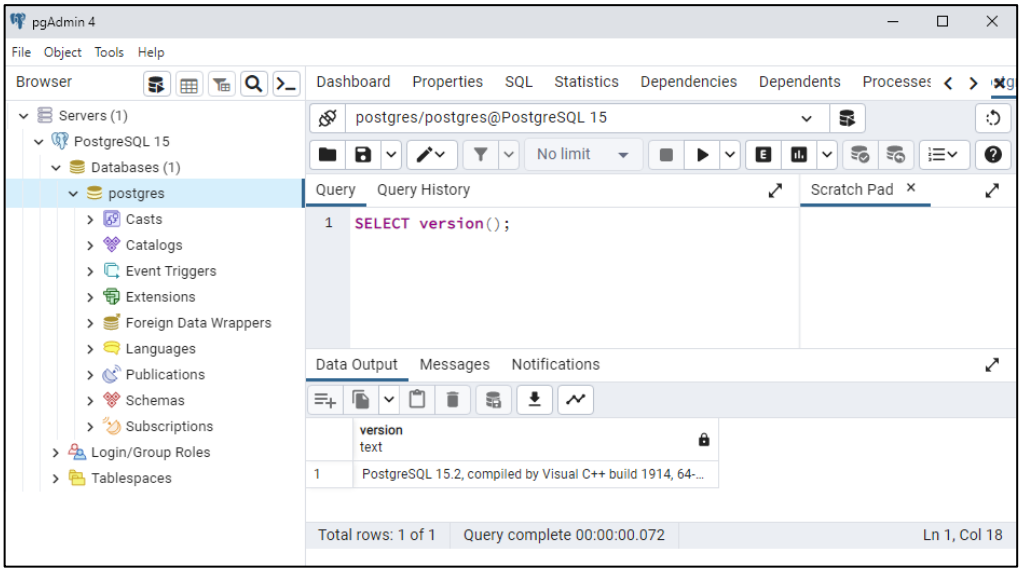

Рисунок 2.9 – pgAdmin 4 та PostgreSQL

Особливо корисною функцією pgAdmin 4 є його повна інтеграція з JSON-функціональністю PostgreSQL, що дозволяє легко переглядати, редагувати та запитувати дані у форматі JSON через графічний інтерфейс. Інструмент має вбудовану систему керування правами доступу, що дозволяє адміністраторам бази даних налаштовувати та контролювати доступ користувачів до різних об'єктів бази даних.

Вибір комбінації PostgreSQL та pgAdmin 4 для проекту системи рекомендацій на платформі macOS є обґрунтованим рішенням, оскільки забезпечує надійну та зручну інфраструктуру бази даних. PostgreSQL відомий своєю стабільною роботою на macOS без необхідності складних налаштувань, а pgAdmin 4 надає всі необхідні інструменти для ефективної розробки та адміністрування.

Після вибору СУБД, варто перейти до розробки ER-діаграми (див. рис. 2.10). ER-діаграма є фундаментальною частиною проєктування бази даних, що надає чітке візуальне представлення того, як різні сутності системи – від користувачів до

наборів даних та рекомендацій – взаємодіють між собою. Це як архітектурний план, що демонструє структуру даних та їхні логічні зв'язки.

Отже, таблиця «users» є сутністю, яка зберігає інформацію про кристувачів і є центральною точкою для зв’язку з іншими таблицями. Ця таблиця є одною стороною у численних зв'язках типу «один до багатьох». Це означає, що один користувач може мати безліч пов'язаних записів в інших частинах бази даних. Таблиця «users» має зв’язок «один до багатьох» з такими сутностями як:

- user_preferences;
- datasets;
- recommendation_sessions;
- feedback_history;

Наприклад, вона пов'язана з «user_preferences» через поле «userId», що вказує на належність вподобань конкретному користувачеві. Аналогічно, один користувач може бути власником або ініціатором багатьох наборів даних, що пов'язує «users» з таблицею «datasets» через зовнішній ключ «userId». Окрім цього, один користувач може проводити численні сеанси рекомендацій, тому «users» пов'язана з «recommendation_sessions» також через «userId», і він може залишати велику кількість записів зворотного зв'язку, що пов'язує users з таблицею «feedback_history» через відповідне поле «userId».

Натомість таблиця «user_preferences», яка містить персональні вподобання користувачів, має зв’язок багато до одного з таблицею «users». Таблиця «datasets», яка зберігає інформацію про набори даних, має такий самий тип зв’язку з таблицею «users», як і «user_preferences». Також в сутності «datasets» є й інші зв’язки, а саме один до багатьох щодо «items» і «recommendation_sessions». Сутність «items», містить окремі елементи, що входять до наборів даних, має зв’язки багато до одного з таблицями «datasets» і «feedback_history», а один до одного з «recommendation_items».

Таблиця «recommendation_sessions» відповідає за збереження інформації про сеанси рекомендацій для користувачів. Допоміжною таблицею до «recommendation_sessions» є «recommendation_items», зберігає кожну окрему

рекомендацію, створену в межах певного сеансу рекомендацій. Сутність feedback_history зберігає історію оцінок і зворотного зв’язку від користувачів.

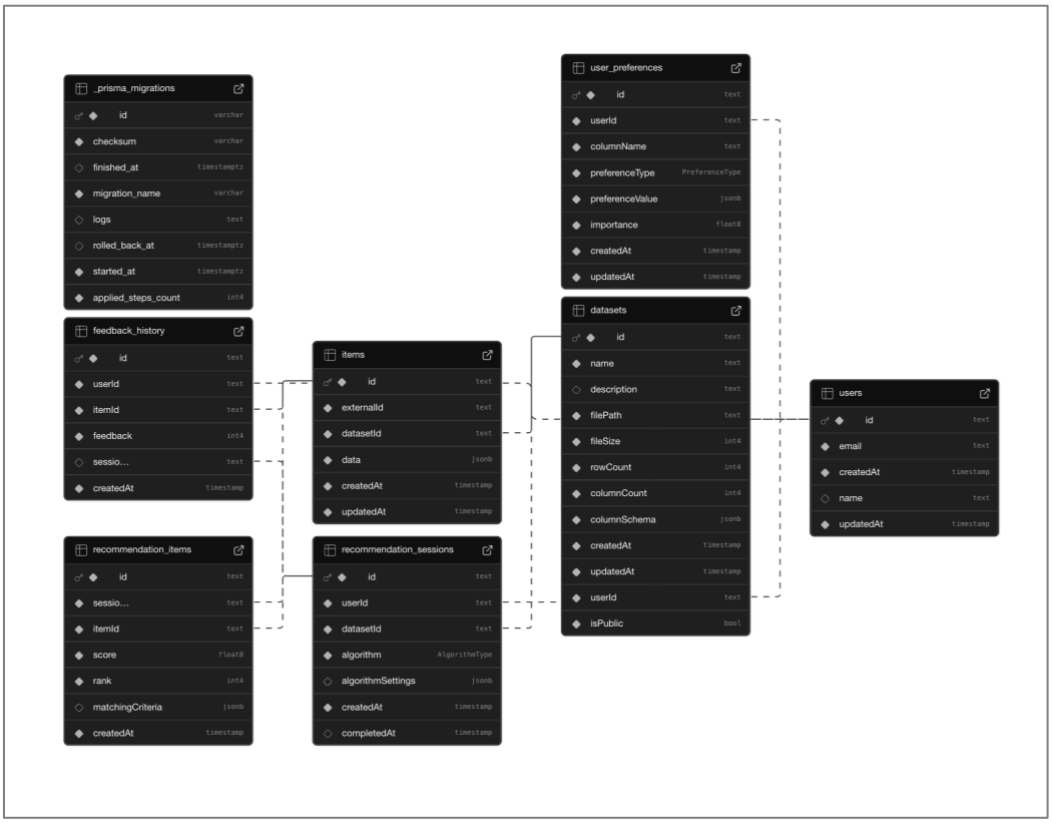

Рисунок 2.10 – ER-діаграма

Наостанок, таблиця «_prisma_migrations» є службовою. Вона створюється інструментами міграції баз даних, наприклад, «Prisma», і використовується для ефективного управління версіями та змінами в схемі бази даних, відстежуючи, які міграції були застосовані. В цілому, представлена ER-діаграма є вичерпною та ефективною моделлю даних для системи, яка зосереджена на користувачах, управлінні даними, генерації рекомендацій та зборі зворотного зв'язку.

2.4 Вибір середовища розробки та технологій

Щоб обрати середовище розробки (IDE) варто зробити аналіз. Вибір технологічного стеку та відповідного середовища розробки є фундаментальним рішенням на етапі проєктування будь-якої програмної системи. Для програмної

системи персоналізованих рекомендацій «Skunk», яка покликана обробляти різнорідні користувацькі дані, забезпечувати динамічну адаптацію та взаємодіяти з сучасними інтелектуальними сервісами, обґрунтований вибір кожної технології є принципово важливим.

Основними критеріями при формуванні технологічного стеку виступали:

- відповідність технології сучасним тенденціям стандартам;
- наявність активної та великої спільноти розробників;
- легкість інтеграції обраних компонентів між собою;
- відповідність вимогам проекту щодо продуктивності, гнучкості та економічної доцільності, особливо на етапі розробки;

Серед найбільш авторитетних програм є декілька, які варто взяти до уваги:

- JetBrains IDE;
- Visual Studio Code.

JetBrains IDE користується попитом завдяки своїм потужним функціям, підтримкою фреймворків, вбудованими інструментами для контролю якості коду, інтерфейсом та багато іншого (рисунок 2.11). Це середовище обираються за його зручність, можливість інтеграції з системами контролю версій, та вищеперерахованими функціями. Але також є великі недоліки, такі як ліцензія на дане середовище, яка коштує тридцять п’ять тисяч гривень на рік, безкоштовна версія можлива лише для студентів. Також, варто враховувати, що JetBrains потребує більше оперативної пам’яті та ресурсів CPU порівняно з легшими альтернативами.

Visual Studio Code, не є повноцінною IDE, але вважається хорошою альтернативною повноцінними середовищам розробки. VS Code є безкоштовним для використання як у особистих, так і в комерційних цілях. Також редактор вихідного коду забезпечує можливість працювати на таких ОС, як Windows, macOS та Linux, він є легким не потребує великих налаштувань, наявна інтеграція з GIT. Відповідно до GitHub Top IDE Index, VS Code посідає четверте місце за популярністю з часткою ринку 10% на основі частоти пошуку на сторінці

завантаження. Існує широкий вибір плагінів сторонніх розробників, доступних для завантаження, щоб покращити досвід кодування.

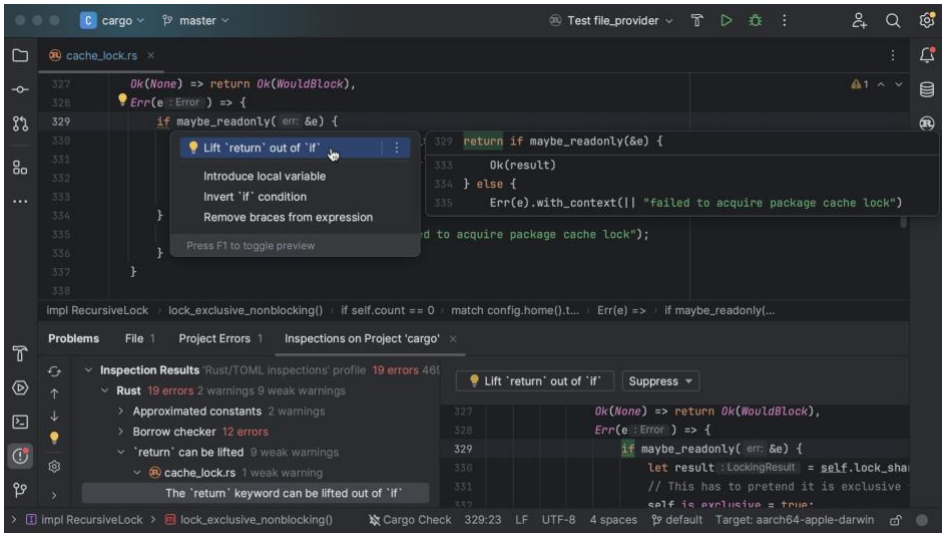

Рисунок 2.11 – JetBrains IDE

З наданої інформації середовищем розробки було обрано Visual Studio Code, що полегшує розробку програмної системи і не потребує витрат, а також додає гнучкості у розробці.

Для забезпечення клієнтської частини використовується Next.js. Next.js — це фреймворк JavaScript з відкритим кодом, яка працює поверх React. Для реалізації основного бекенду, який у системі «Skunk» виокремлений у вигляді мікросервісу рекомендацій та відповідає за обчислювально складні завдання аналізу даних та генерації персоналізованих пропозицій, було обрано мову програмування Python та веб-фреймворк FastAPI. Python традиційно є мовою вибору для завдань у сфері Data Science та машинного навчання завдяки своїй відносній простоті синтаксису, великій кількості високоякісних наукових бібліотек та активній спільноті.

FastAPI, будучи сучасним Python-фреймворком, обраний за його виняткову продуктивність, яка досягається завдяки використанню асинхронного програмування, а також за такі переваги, як автоматична генерація інтерактивної документації API та вбудована валідація даних за допомогою Pydantic. Це дозволяє швидко розробляти надійні API, що є критичним для мікросервісної архітектури.

Саме на Python та FastAPI реалізована логіка роботи з бібліотеками Sentence Transformers для отримання векторних представлень, FAISS для швидкого пошуку схожості та Vowpal Wabbit для реалізації механізмів навчання на основі зворотного зв'язку за допомогою контекстуальних бандитів.

Для прискорення розробки візуальних компонентів інтерфейсу та забезпечення консистентного дизайну використовуються готова бібліотека UI-компонентів Preline UI (рисунок 2.12). Preline UI — це бібліотека компонентів CSS Tailwind із відкритим кодом для будь-яких потреб. Поставляється з прикладами та блоками інтерфейсу користувача, шаблонами, плагінами, системою дизайну Figma.

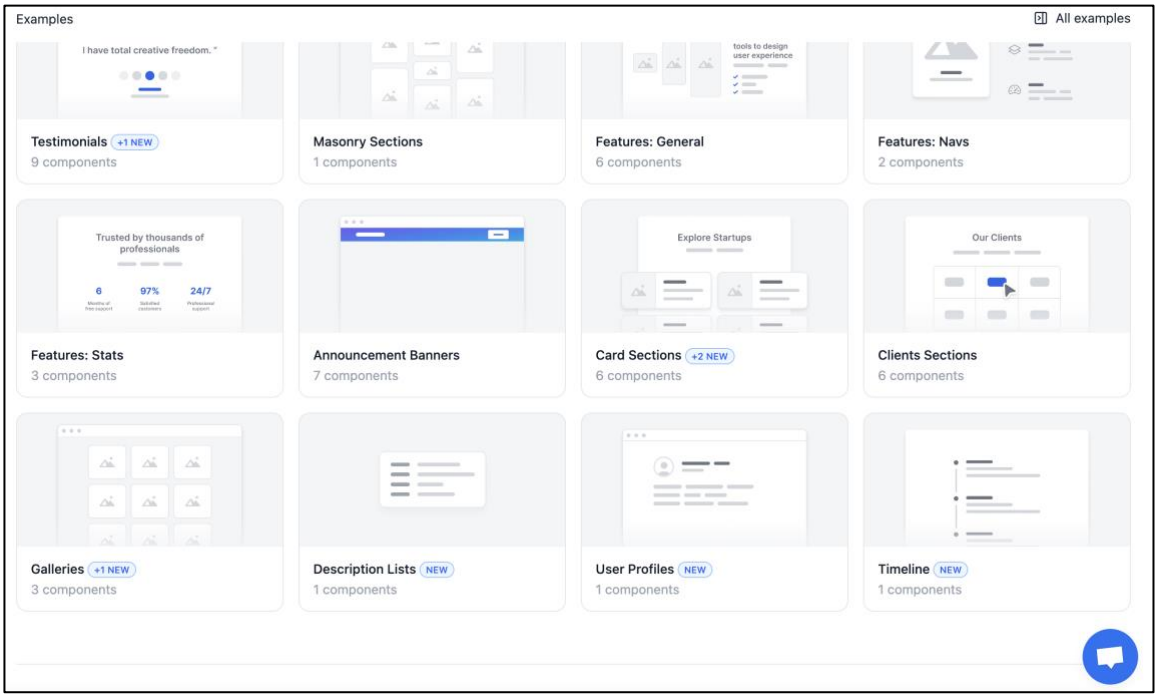

Рисунок 2.12 – Preline UI

Як раніше було визначено основною базою даних є PostgreSQL, для забезпечення безперервної роботи системи, коректного відображення таблиць та оптимізації даних, забезпечує високу продуктивність та безпеку даних, що є ключовим для системи рекомендацій, що розгортається на керованому сервісі Vercel Postgres. Для зберігання наборів даних, що використовуються в системі, обрано платформу Supabase, яка забезпечує легкий доступ до PostgreSQL у хмарному середовищі.

Для взаємодії з базою даних використовується Prisma, що спрощує керування базою даних. Prisma ORM — це ORM нового покоління з відкритим кодом. Він складається з наступних частин:

- Prisma Client, автоматично згенерований і безпечний конструктор запитів для Node.js і TypeScript;
- Prisma Migrate, система міграції;
- Prisma Studio, графічний інтерфейс для перегляду та редагування даних у вашій базі даних.

Для роботи з ШІ, використовується Gemini API. Аналіз проводився серед багатьох ШІ, але ключовими з них були:

- Gemini;
- ChatGPT.

ChatGPT використовує платні токени, для GPT-4 базова вартість складає три цента за 1 000 вхідних токенів і шість центів за 1 000 вихідних токенів. Також, варто зауважити, що ChatGPT потребує більш складних алгоритмів оптимізації, задля коректної роботи програми.

Тим часом, Gemini API, можливо скористуватись, як і безкоштовною версією так і платною, а саме за вхідні і вихідні дані:

- $0.075 за 1 мільйон токенів;
- $0.30 за 1 мільйон токенів.

Для контролю версій коду використовується Git у поєднанні з платформою GitHub, що є загальновизнаним стандартом для сучасної розробки програмного забезпечення та забезпечує ефективну командну роботу, відстеження змін та управління кодовою базою.

Модуль автентифікації та управління користувачами реалізовано через інтеграцію з сервісом Clerk. Clerk є хмарним сервісом, який надає готові рішення для автентифікації та управління користувачами у веб-застосунках. Його головна мета — спростити реалізацію входу, реєстрації, управління сесіями, профілями користувачів та авторизацією без потреби розробляти ці функції з нуля.

Для тестування використовується Jest, який дозволяє перевіряти окремі функції та модулі. Jest — це фреймворк тестування JavaScript, призначений для перевірки правильності будь-якої кодової бази JavaScript. Cypress забезпечує енд-то-енд тестування, перевіряючи взаємодію різних елементів системи. Cypress був розроблений для виконання наскрізних E2E тестів на всьому, що працює в браузері. Типовий тест E2E відкриває програму в браузері та виконує дії через інтерфейс користувача так само, як це зробив би реальний користувач.

Для автоматичного розгортання використовується Vercel, що інтегрується з GitHub і дозволяє миттєво публікувати оновлення після злиття змін. Після злиття змін у головну гілку на GitHub, Vercel автоматично створює нову версію додатка. Це дозволяє миттєво публікувати оновлення, тобто Vercel є хмарною платформою для хостингу веб-додатків.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Опис процесу розробки основних модулів

Попередня гібридна багаторівнева архітектура та обраний технологічний стек були основою для процесу програмної реалізації системи персоналізованих рекомендацій «Skunk». Цей етап має вирішальне значення, оскільки саме тут теоретичні розробки та проектні рішення втілюються в функціонуючий програмний продукт.

Підрозділ даних детально описує створення основних модулів системи, які відповідають за збір і підготовку даних, управління автентифікацією користувачів, забезпечення взаємодії з користувацьким інтерфейсом і координацію роботи з зовнішніми сервісами. Розробка цих основних компонентів є необхідною умовою для ефективного функціонування ядра персоналізованих рекомендацій.

Весь процес розробки супроводжувався дотриманням принципів модульності, що передбачає чіткий розподіл відповідальностей між окремими частинами системи, та ітеративним підходом, що дозволяло поступово нарощувати функціональність та тестувати компоненти на ранніх стадіях.

BFF, шар Backend-To-Frontend, який працює за допомогою Routes API Next.js, є центральною частиною архітектури системи. Він керує клієнтами та сервісом штучного інтелекту. Модуль управління датасетами та їх підготовки був одним із перших компонентів цього шару.

Цей модуль реалізується через API ендпоінт файлів. Він працює з CSV-файлами, які завантажує користувач. Файл валідується на сервері після його отримання. Вкрай важливо, щоб кожен запис у CSV-файлі мав унікальний ідентифікатор «sknq_id». Цей процес виконується у функції «processCSVData», яка є важливою, оскільки вона надає стандартизований спосіб ідентифікації об’єктів у будь-якому завантаженому датасеті, що необхідно для правильної роботи рекомендованих алгоритмів і збору зворотного зв’язку.

Наступним значущим компонентом BFF є модуль генерації запитань до користувача, здійснений через API ендпоінт «questions». Цей модуль інкапсулює

логіку взаємодії з Google Gemini API. Він формує запит до Gemini, передаючи інформацію про попередньо завантажений та оброблений датасет зазвичай у вигляді URI файлу.

Модель Gemini аналізує надані дані та, відповідно до системного промпту, що містить інструкції щодо бажаного типу, кількості та спрямованості запитань, повертає список запитань у форматі JSON. Ці дані потім зберігаються у базі даних PostgreSQL для подальшого користування.

Наведений нижче фрагмент коду демонструє ключові кроки взаємодії з Gemini API. Спочатку ініціалізується клієнт GoogleGenerativeAI з відповідним API-ключем та налаштуваннями моделі, включаючи очікуваний формат відповіді JSON та налаштування безпеки. Функція POST приймає URI файлу та тип MIME, формує запит до моделі, включаючи системний та користувацький промпти, а також посилання на дані файлу. Отримана відповідь від Gemini API обробляється, і згенерований текст, що містить питання та аналіз повертається клієнту.

Передбачається, що ці дані також будуть збережені в базу даних для подальшого використання, пов'язуючи їх з конкретним датасетом.

Наведено код програмної реалізації ілюструє генерацію запитань для користувача:

```
import { NextRequest, NextResponse } from 'next/server';
import { GoogleGenerativeAI, HarmCategory, HarmBlockThreshold } from
'@google/generative-ai';

const apiKey = process.env.GEMINI_API_KEY;
if (!apiKey) {
    console.error('GEMINI_API_KEY is not defined');
    throw new Error('GEMINI_API_KEY is not defined');
}
const genAI = new GoogleGenerativeAI(apiKey);

const model = genAI.getGenerativeModel({
   model: 'gemini-1.5-flash-latest',
   generationConfig: {
      responseMimeType: 'application/json',
   },
   safetySettings: [
    { category: HarmCategory.HARM_CATEGORY_HARASSMENT, threshold:
HarmBlockThreshold.BLOCK_ONLY_HIGH, },
    { category: HarmCategory.HARM_CATEGORY_HATE_SPEECH, threshold:
HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE, },
  ],
});
```

```
export async function POST(req: NextRequest) {
   try {
      const body = await req.json();
      const { fileUri, mimeType, datasetId } = body;

      if (!fileUri || !mimeType || !datasetId) {
        return NextResponse.json({ error: 'Missing fileUri, mimeType, or
datasetId' }, { status: 400 });
      }

      const systemPromptForGemini = "Analyze the provided dataset (via fileData)
and generate 5-8 diverse questions for a user preference questionnaire. The goal
is to understand user preferences to make personalized recommendations based on
this dataset. Questions should cover different aspects and column types
(categorical, numerical, text-based). For each question, specify: 'type'
(SINGLE_CHOICE, MULTIPLE_CHOICE, SLIDER, CARD_CHOICE), 'question' (text of the
question), 'choices' (array of possible answers, or min/max for SLIDER), 'column'
(dataset column it relates to), 'column_type' (numeric, categorical, image, text),
and 'importance_weight' (a float from 0.5 to 2.0). Also, provide
'dataset_analysis': { 'is_suitable_for_analysis': boolean, 'reason': string,
'metadata': {'domain_type': string, 'record_count': number, 'value_distributions':
string, 'key_features': string[]} }. Return as JSON.";
      const userPromptForGemini = "Dataset is provided below. Please generate
questions and analysis.";

      const result = await model.generateContent([
        {text: systemPromptForGemini},
        {text: userPromptForGemini},
         {
            fileData: {
               fileUri,
               mimeType,
            },
         },
      ]);

      const responseText =
result.response.candidates?.[0]?.content?.parts?.[0]?.text;
      if (!responseText) {
         return NextResponse.json({ error: 'No valid response generated from
Gemini' }, { status: 500 });
      }

      const parsedData = JSON.parse(responseText);
      await prisma.dataset.update({
          where: { id: datasetId },
          data: {
             geminiGeneratedQuestions: parsedData.questions,
             datasetMetadata: parsedData.dataset_analysis.metadata,
             geminiAnalysisStatus: 'COMPLETED',
          },
       });

      return NextResponse.json({ response: responseText }, { status: 200 });
   } catch (error: any) {
      console.error('Error generating questions via Gemini:', error);
      return NextResponse.json({ error: `Internal Server Error: ${error.message}`
}, { status: 500 });
   }}
```

Система автентифікації користувачів реалізована через сервіс «Clerk». Коли користувач входить у систему, BFF отримує про нього необхідну інформацію, як-от «ID» та «email», і синхронізує ці дані з базою PostgreSQL. Це дозволяє програмі ідентифікувати користувача під час роботи з датасетами та відповідями.

Коли користувач відповідає на згенеровані питання, ці відповіді зберігаються в базі через спеціальний API. Вони пов’язуються з конкретним користувачем, датасетом і запитанням, формуючи профіль його вподобань.

Щоб отримати персоналізовані рекомендації, BFF надсилає у FastAPI мікросервіс запит із профілем користувача, ідентифікатором датасету та відповідями. Цей запит надсилається через маршрут «/api/recommendation», а у відповідь FastAPI повертає список рекомендацій, які BFF передає на фронтенд. Після цього користувач може залишити фідбек, а саме оцінити, наскільки йому сподобались рекомендації. Ця інформація передається через маршрут «/api/feedback» і використовується для покращення майбутніх результатів.

Наведено код програмної реалізації ілюструє отримання персоналізованих рекомендацій для користувача:

```
import { NextRequest, NextResponse } from 'next/server';

interface Preference {
   questionId: number;
   columnName: string;
   value: string | string[] | number;
   importance?: number;
}

interface RecommendationRequestPayload {
   preferences: Preference[] | Record<string, Preference>;
   datasetId: string;
   strategy?: string;
   userId?: string;
}

const FASTAPI_URL = process.env.FASTAPI_URL

export async function POST(req: NextRequest) {
   try {
      const { preferences, strategy, userId, datasetId } =
         (await req.json()) as RecommendationRequestPayload;

      const preferencesArray = Array.isArray(preferences)
         ? preferences
         : Object.values(preferences);
```

```
      const answersForFastAPI = preferencesArray.map((pref: Preference) => ({
         column_name: pref.columnName,
         answer: pref.value,
         importance: pref.importance || 1.0,
      }));

      const requestBodyForFastAPI = {
         dataset_path: datasetId,
         answers: answersForFastAPI,
         top_n: 5,
         strategy: strategy || 'content_based',
         user_id: userId,
      };

      const endpoint = `${FASTAPI_URL}/api/v1/recommend`;
      const fastApiResponse = await fetch(endpoint, {
         method: 'POST',
         headers: { 'Content-Type': 'application/json', },
         body: JSON.stringify(requestBodyForFastAPI),
      });

      if (!fastApiResponse.ok) {
         const errorText = await fastApiResponse.text();
         console.error('FastAPI error response:', errorText);
         throw new Error(`Error from FastAPI: ${fastApiResponse.statusText}.
Details: ${errorText}`);
      }

      const data = await fastApiResponse.json();
      return NextResponse.json(data, { status: 200 });

   } catch (error: any) {
      console.error('Error getting recommendations:', error);
      return NextResponse.json({ error: `Failed to get recommendations:
${error.message}` }, { status: 500});
   }
}
```

Розробка мікросервісу рекомендацій на FastAPI відбувалася паралельно, з акцентом на створенні чітких API ендпоінтів для прийому запитів від BFF та використання Pydantic моделей для валідації даних. Основна логіка, пов'язана з алгоритмами персоналізації, інкапсульована у відповідних сервісних класах, детальний опис яких буде надано у наступному підрозділі.

Взаємодія між усіма описаними модулями на різних архітектурних рівнях реалізована переважно через REST API, з використанням формату JSON для обміну даними. Реалізація цих основних модулів створила необхідну інфраструктуру для збору даних, взаємодії з користувачем та підготовки до ефективної роботи основного рекомендаційного ядра системи.

3.2 Реалізація алгоритму персоналізованих рекомендацій

Алгоритм персоналізованих рекомендацій є ядром програмної системи, що відповідає за інтелектуальний підбір індивідуальних пропозицій. Його розробка та реалізація відбувалися в рамках спеціалізованого мікросервісу на FastAPI, що дозволило ефективно використовувати потужності мови Python та її багату екосистему бібліотек для машинного навчання та аналізу даних.

Слід зазначити, що підхід до формування рекомендацій у системі є гібридним, поєднуючи як методи, засновані на аналізі контенту та подібності, для генерації початкового набору кандидатів, так і механізми навчання з підкріпленням для динамічної адаптації до переваг конкретного користувача на основі його зворотного зв'язку.

Такий комплексний підхід спрямований на забезпечення високої релевантності пропозицій для широкого спектру довільних датасетів, що можуть бути завантажені користувачами.

Процес генерації рекомендацій починається з отримання мікросервісом даних від Backend-for-Frontend. Ці дані включають профіль користувача, сформований на основі його відповідей на динамічно згенеровані запитання щодо конкретного датасету, та власне сам датасет, з якого необхідно зробити вибірку. Далі вступають в дію реалізовані стратегії рекомендацій.

Однією з основних стратегій для створення початкових рекомендацій у системі є метод, що базується на векторних представленнях, а саме «EmbeddingBasedStrategy» (див. рис. 3.1)

Суть цього підходу в тому, що і самі об'єкти з датасету, і відповіді користувача на запитання перетворюються у спеціальні вектори з чисел їх називають ембедингі. Ці вектори «перекладають» текстові описи та характеристики в математичну форму, яка зберігає сенс і зміст.

Для створення таких векторів використовуються сучасні моделі штучного інтелекту, побудовані на архітектурі «Transformer», які добре розуміють мову та зміст текстів. Це дозволяє порівнювати вподобання користувача з об'єктами з датасету і знаходити найбільш релевантні рекомендації.

У системі «Skunk» для цього застосовується бібліотека «Sentence Transformers» з однією з її популярних моделей, «paraphrase-MiniLM-L3-v2». Ця модель здатна перетворювати текстові фрагменти різної довжини на щільні вектори фіксованої розмірності, де семантично близькі тексти матимуть близькі векторні представлення.

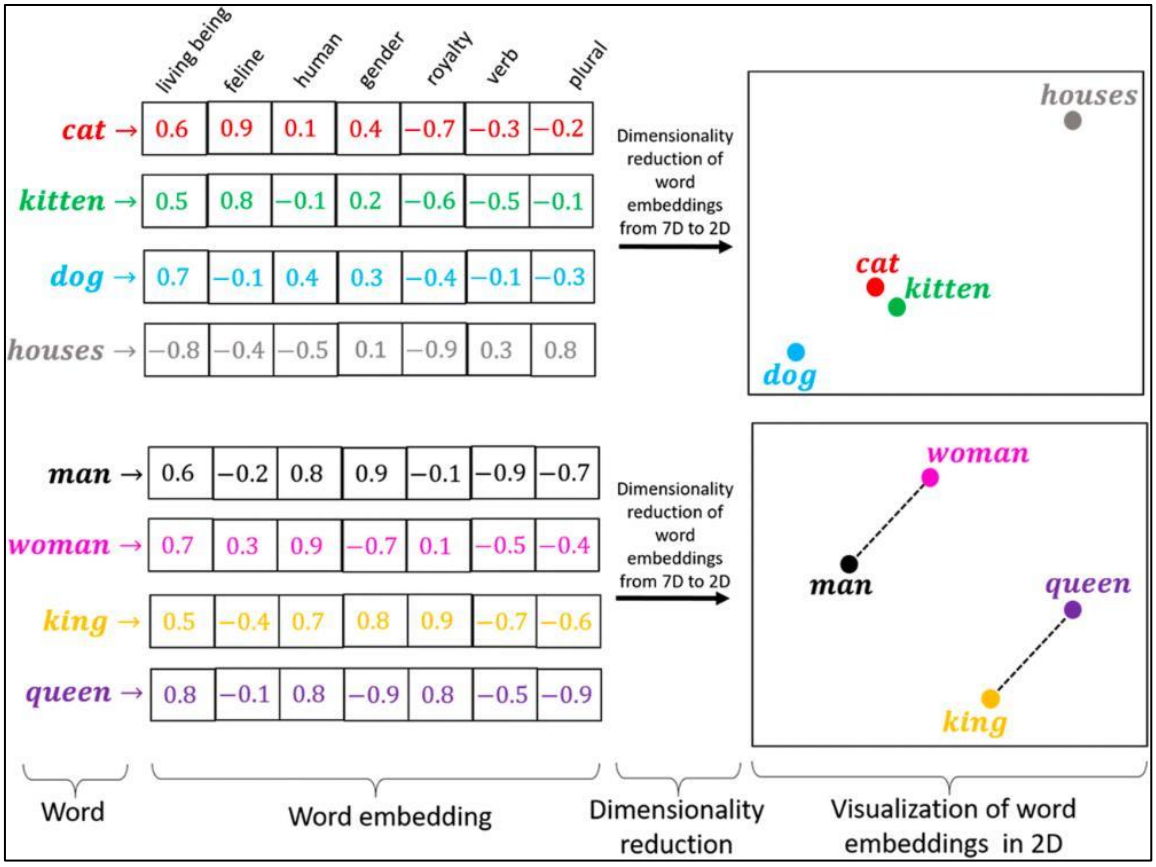

Рисунок 3.1 – EmbeddingBasedStrategy

Таким чином, кожен об'єкт датасету та профіль користувача отримують свої унікальні «координати» у багатовимірному семантичному просторі.

Цей фрагмент коду показує, як працює стратегія рекомендацій на основі векторів ембедингів. Спочатку завантажується модель «SentenceTransformer», яка вміє перетворювати текст у вектори, що зберігають зміст. Для кожного об’єкта в датасеті створюється короткий текстовий опис, при тому об’єднуються значення з колонок, які відповідають наданим користувачем відповідям.

Після цього метод «get_recommendations»:

– формує один текст із відповідей користувача;

– перетворює цей текст у вектор, тобто ембединг;

– також створює вектори для кожного об’єкта з датасету;

– шукає ті об’єкти, які найбільше схожі на вподобання користувача.

Для швидкого пошуку використовується бібліотека FAISS, яка створює індекс з усіх векторів об’єктів і знаходить найближчі до вектора користувача за косинусною схожістю. У результаті система повертає список рекомендацій, які найбільше відповідають вподобанням користувача.

Наведено код програмної реалізації ілюструє роботу алгоритму початкових рекомендацій:

```
from sentence_transformers import SentenceTransformer
import numpy as np
import pandas as pd
import faiss
from typing import List, Dict, Any, Optional

class EmbeddingBasedStrategy:
    def __init__(self, model_name: str = "paraphrase-MiniLM-L-6-v2"):
        self.model: Optional[SentenceTransformer] = None
        try:
            self.model = SentenceTransformer(model_name)
        except Exception as e:
            print(f"Error loading SentenceTransformer model {model_name}: {e}")

    def _compute_embeddings(self, texts: List[str]) -> Optional[np.ndarray]:
        if not self.model or not texts:
            return None
        try:
            return self.model.encode(texts, convert_to_numpy=True,
show_progress_bar=False)
        except Exception as e:
            print(f"Error computing embeddings: {e}")
            return None

    def _prepare_item_text_for_embedding(self, item_data: pd.Series,
user_preference_columns: List[str]) -> str:
        feature_texts = [f"{col.replace('_', ' ').capitalize()}: {item_data[col]}"
                         for col in user_preference_columns if col in item_data
and pd.notna(item_data[col])]
        return ". ".join(feature_texts) if feature_texts else ""

    def _prepare_recommendation_output(self, df: pd.DataFrame, indices: List[int],
scores: List[float]) -> List[Dict[str, Any]]:
        return [{"sknq_id": str(df.iloc[idx].get("sknq_id", idx)),
                 "title": str(df.iloc[idx].get("Title", f"Item {idx}")),
                 "match_score": float(scores[i]),
                 "details": df.iloc[idx].to_dict()} for i, idx in
enumerate(indices)]
```

```
    async def get_recommendations(self, request_data: Dict[str, Any], df:
pd.DataFrame) -> List[Dict[str, Any]]:
        if not self.model: return []

        user_answers = request_data.get("answers", [])
        top_n = request_data.get("top_n", 5)

        user_query_parts = [f"{ans['column_name'].replace('_', ' ')} is
{ans['answer']}"
                            for ans in user_answers if ans.get("column_name") and
ans.get("answer")]
        user_preference_columns = list(set([ans['column_name'] for ans in
user_answers if ans.get("column_name")]))

        if not user_query_parts: return []
        user_query_text = ". ".join(user_query_parts)
        user_embedding = self._compute_embeddings([user_query_text])
        if user_embedding is None or len(user_embedding) == 0: return []

        item_texts_with_indices = []
        for index, item_row in df.iterrows():
            item_text = self._prepare_item_text_for_embedding(item_row,
user_preference_columns)
            if item_text:
                item_texts_with_indices.append({'text': item_text,
'original_index': index})

        if not item_texts_with_indices: return []

        item_embeddings = self._compute_embeddings([item['text'] for item in
item_texts_with_indices])
        if item_embeddings is None or len(item_embeddings) == 0: return []

        index_dim = item_embeddings.shape[1]
        faiss_index = faiss.IndexFlatIP(index_dim)

        item_embeddings_normalized = item_embeddings.astype(np.float32)
        faiss.normalize_L2(item_embeddings_normalized)
        user_embedding_normalized = user_embedding.astype(np.float32)
        faiss.normalize_L2(user_embedding_normalized)

        faiss_index.add(item_embeddings_normalized)

        k = min(top_n, len(item_embeddings_normalized))
        distances, indices_faiss = faiss_index.search(user_embedding_normalized,
k)

        original_df_indices = [item_texts_with_indices[i]['original_index'] for i
in indices_faiss[0]]

        return self._prepare_recommendation_output(df, original_df_indices,
distances[0].tolist())
```

Ключовим аспектом програмної системи, що забезпечує її розвиток та підвищення якості пропозицій з часом, є здатність до динамічної адаптації та навчання на основі зворотного зв'язку від користувача. Ця функціональність

реалізована за допомогою бібліотеки «Vowpal Wabbit» та підходу на основі контекстуальних бандитів.

Контекстуальні бандити – це клас алгоритмів машинного навчання з підкріпленням, які навчаються обирати найкращу «дію» (див. рис. 3.2). У даному випадку – рекомендувати певний об'єкт в залежності від поточного «контексту», тобто характеристики користувача та ситуації. Коли користувач взаємодіє з запропонованими рекомендаціями, наприклад, позначаючи їх як «подобається» чи «не подобається», ця інформація використовується як сигнал підкріплення для донавчання персоналізованої моделі «Vowpal Wabbit».

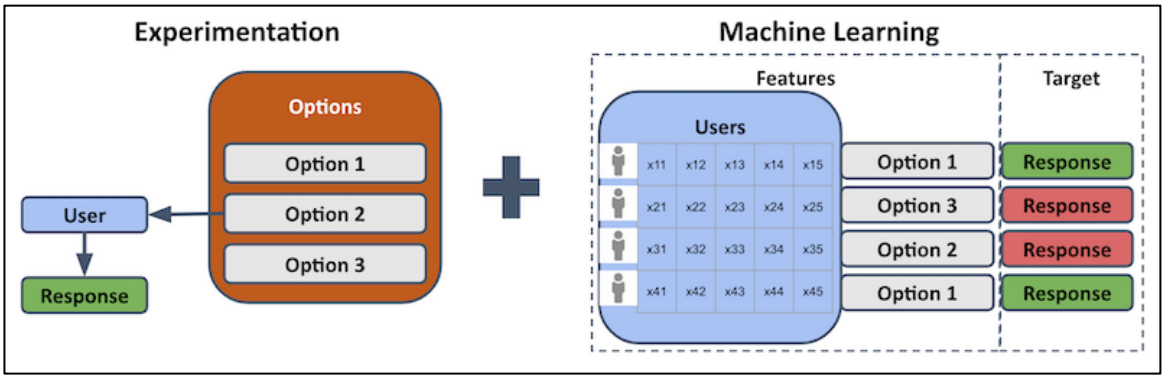

Рисунок 3.2 – Режим контекстуальні бандити

Таким чином, програма поступово «вивчає» індивідуальні смаки користувача та адаптує свої майбутні рекомендації.

Наведений нижче скорочений фрагмент коду з сервісу «VWFeedbackLearner» ілюструє основну логіку запису зворотного зв'язку та використання моделі для переранжування. Метод «record_feedback» приймає інформацію про взаємодію, формує навчальний приклад у специфічному для «VW» форматі та оновлює відповідну модель.

Метод «rerank_recommendations» використовує навчену модель для отримання оцінок для списку кандидатів, які потім використовуються для їх сортування з урахуванням стратегії «epsilon-greedy».

Наведено код програмної системи, який реалізує адаптивну систему навчання з підкріпленням для покращення рекомендацій на основі фідбеку користувачів, використовуючи бібліотеку VW і її режим contextual bandits:

```
from vowpalwabbit import pyvw
import os, random
from typing import List, Dict, Any

class VWFeedbackLearner:
    def __init__(self, model_dir: str = "vw_models_sknq", epsilon: float = 0.2):
        self.model_dir = model_dir
        self.default_epsilon = epsilon
        self.vw_models_config = {}
        os.makedirs(self.model_dir, exist_ok=True)

    def _get_or_create_model(self, user_id: str):
        if user_id not in self.vw_models_config:
            try:
                args = f"--cb_explore_adf --quiet --epsilon
{self.default_epsilon}"
                self.vw_models_config[user_id] = {
                    'model': pyvw.vw(args), 'epsilon': self.default_epsilon,
'feedback_count': 0
                }
            except:
                return None
        return self.vw_models_config[user_id]['model']

    def _to_vw_str(self, features: Dict[str, Any]) -> str:
        return " ".join(f"|{ns} " + " ".join(
            f"{k}:{v}" if not isinstance(v, bool) else k for k, v in feats.items()
        ) for ns, feats in features.items())

    def _format_learn_example(self, ctx: str, idx: int, cost: float, prob: float,
acts: List[str]) -> str:
        return "\n".join([f"shared {ctx}"] + [
            f"{i}:{cost}:{prob} {a}" if i == idx else f"{i} {a}" for i, a in
enumerate(acts)
        ])

    def _format_predict_example(self, ctx: str, acts: List[str]) -> str:
        return "\n".join([f"shared {ctx}"] + acts)

    def record_feedback(self, user_id: str, shown: List[Dict[str, Any]],
chosen_id: str,
                        feedback: int, user_feats: Dict[str, Any], df_all):
        vw = self._get_or_create_model(user_id)
        if not vw: return

        ctx = self._to_vw_str({'user': user_feats})
        acts = [self._to_vw_str({'item': d}) for d in shown]
        ids = [str(d.get("sknq_id")) for d in shown]
        idx = ids.index(chosen_id) if chosen_id in ids else -1
        if idx == -1: return

        try:
            probs = vw.predict(self._format_predict_example(ctx, acts))
            prob = probs[idx] if isinstance(probs, list) else 1.0 / len(acts)
        except:
            return
```

```
        cost = 0.0 if feedback == 1 else 1.0
        vw.learn(self._format_learn_example(ctx, idx, cost, prob, acts))
        self.vw_models_config[user_id]['feedback_count'] += 1

    def rerank_recommendations(self, user_id: str, items: List[Dict[str, Any]],
                               user_feats: Dict[str, Any], df_all, top_n: int =
5):
        if not items: return []
        vw = self._get_or_create_model(user_id)
        if not vw:
            random.shuffle(items)
            return items[:top_n]

        ctx = self._to_vw_str({'user': user_feats})
        acts = [self._to_vw_str({'item': i}) for i in items]
        try:
            scores = vw.predict(self._format_predict_example(ctx, acts))
        except:
            random.shuffle(items)
            return items[:top_n]

        scored = [{**d, 'vw_score': s} for d, s in zip(items, scores)]
        if random.random() < self.vw_models_config[user_id]['epsilon']:
            random.shuffle(scored)
        else:
            scored.sort(key=lambda x: x['vw_score'], reverse=True)
        return scored[:top_n]
```

Таким чином, реалізація алгоритму персоналізованих рекомендацій є багатоетапним процесом, що інтегрує сучасні підходи до обробки текстових даних, ефективні методи пошуку за подібністю для генерації початкових кандидатів, а також адаптивні механізми навчання з підкріпленням. Цей гібридний підхід дозволяє системі не лише надавати релевантні пропозиції для широкого спектру довільних датасетів, але й динамічно вдосконалюватися, враховуючи унікальний досвід та реакції кожного користувача.

Важливим напрямком подальшого розвитку може стати дослідження більш складних методів формування характеристик для Vowpal Wabbit, що може покращити точність моделі, а також оптимізація стратегії дослідження для прискорення навчання та забезпечення кращого покриття простору можливих рекомендацій, уникаючи при цьому надмірної експлуатації вже відомих переваг.

3.3 Тестування програмного забезпечення

Основною метою тестування персоналізованої системи було виявлення та подальше усунення максимальної кількості дефектів та невідповідностей на

якомога більш ранніх стадіях розробки. Це дозволяє підвищити загальну якість програмного продукту та рівень задоволеності кінцевих користувачів.

Даний підрозділ присвячений детальному опису обраної методики проведення тестування, що охоплює різні рівні та види перевірок, а також інструменти, що застосовувалися для забезпечення якості системи.

Методика тестуваннч для «Skunk» базувалася на комплексному підході, що передбачав E2E тестування і Unit-тести, зокрема використовуючи такі бібліотеки Jest, React Testing Library, PyTest, Cypress. Jest використовувався для модульного тестування окремих частин, таких як обробка файлів і компоненти інтерфейсу. Cypress застосовувався для наскрізного тестування, що імітує дії користувача, від завантаження файлу до отримання рекомендацій.

Для перевірки використовувалися різні типи CSV-файлів:

- коректні;
- частково зіпсовані;
- з помилками у форматі;
- відсутніми заголовками;
- невірними типами даних.

Створювалися як синтетичні, так і анонімізовані фрагменти реальних наборів. Тестувалися відповіді. Для зовнішніх API використовувалися як справжні запити, так і фальсифіковані.

Для позитивного тестування використовувалися дані, що повністю відповідали очікуваному формату та структурі. Метою було перевірити успішне виконання основних функціональних сценаріїв. Прикладом коректного CSV-файлу міг бути невеликий набір даних з чітко визначеними заголовками та консистентними типами даних у колонках. Такий файл міг містити, наприклад, наступні стовпці та дані:

Наведено код програмної реалізації ілюструє CSV-файл:

```
Title,Genre,Year,Rating
Inception,Sci-Fi,2010,8.8
```

```
The Shawshank Redemption,Drama,1994,9.3
The Dark Knight,Action,2008,9.0
```

Одним із перших та критично важливих сценаріїв, що підлягали тестуванню, був процес завантаження та аналізу датасету користувачем. Перевірялося завантаження CSV-файлу через веб-інтерфейс, далі його передача на Backend-for-Frontend, потім обробка, і збереження у хмарному сховищі Supabase Storage, а також ініціація запиту до Google Gemini API для аналізу та генерації запитань.

Результатом очікувння було отримання валідного JSON-відповіді від Gemini API, коректне збереження метаданих та запитань у базі даних PostgreSQL та відображення відповідного статусу й згенерованих запитань на користувацькому інтерфейсі. Для автоматизації перевірки цього сценарію на рівні BFF було розроблено інтеграційні тести з використанням Jest.

Теж важливо зазначити, що перевірялася реакція системи на спробу завантаження файлу не підтримуваного формату. Очікуваним результатом було відображення користувачеві чіткого повідомлення про помилку та блокування подальшого завантаження.

Наступним етапом тестування був процес опитування користувача та формування його профілю. Перевірялося наскільки коректно йде відображення динамічних запитань, згенерованих для конкретного датасету, на інтерфейсі користувача. Особлива увага приділялася тому, як компонент «PreferenceQuestionnaire» обробляє різні типи запитань та збирає відповіді користувача.

Тестувалося успішне надсилання зібраних відповідей на BFF та їх подальше збереження у базі даних PostgreSQL.

Перевірка генерації та відображення рекомендацій була центральною частиною тестування. Після надання користувачем відповідей на запитання, ініціювався запит на отримання рекомендацій. Тести перевіряли, що BFF успішно формує запит до FastAPI мікросервісу, передаючи необхідні дані. На стороні FastAPI перевірялася коректна обробка цього запиту, застосування відповідної

стратегії рекомендацій та повернення списку релевантних об'єктів. На фронтенді контролювалося коректне відображення отриманих рекомендацій.

Після проведення автоматизованого тестування різних частин системи, інструменти звітування надали узагальнену інформацію про пройдені тести. Нижче наведено імітований приклад такого виводу, що відображає успішне виконання значної кількості тестів (див. рис. 3.3).

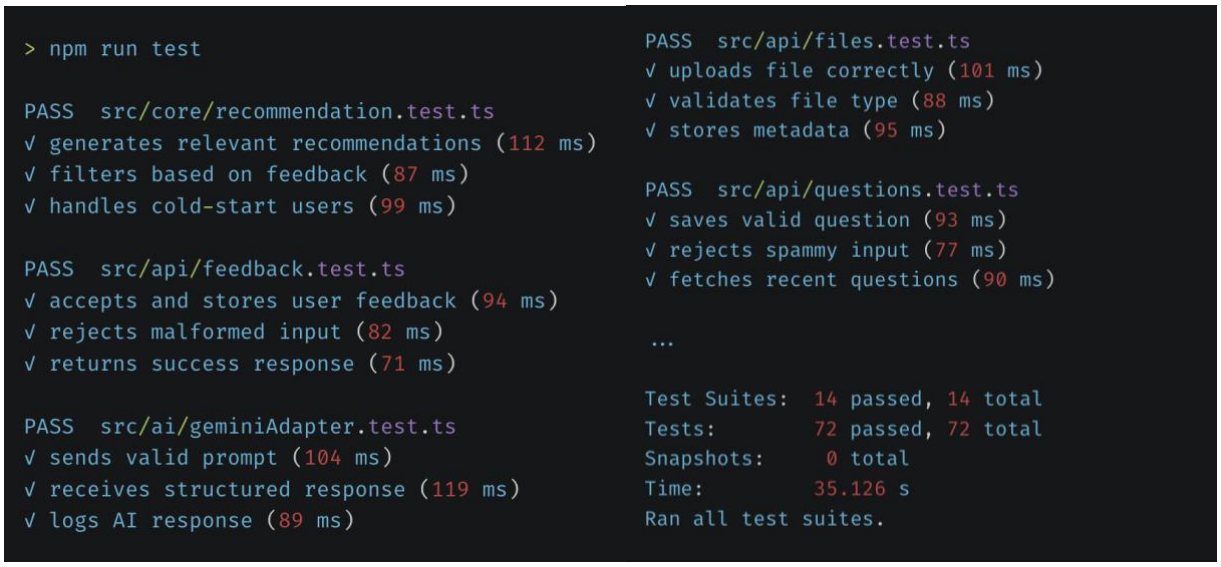

Рисунок 3.3 – Режим контекстуальні бандити

3.4 Реалізація інтерфейсу користувача UI/UX.

Таким чином, інтерфейс програмної системи, який використовується для аналізу даних і генерації індивідуальних рекомендацій, показано на першому зображенні (рис. 3.4). Головний заголовок «Make Insights into your Data» підкреслює мету «Skunk», яка полягає в перетворенні даних на цінні відкриття. Користувачам надається два варіанти взаємодії. Вони можуть завантажити свій власний набір даних у форматах CSV або JSON або використовувати зразкові дані.

Наразі доступні три приклади:

– база даних фільмів, яка містить персоналізовані рекомендації щодо фільмів;

– база даних книг, яка містить інформацію про книги разом із індивідуальними пропозиціями;

– комплект музичних даних, який включає плейлисти.

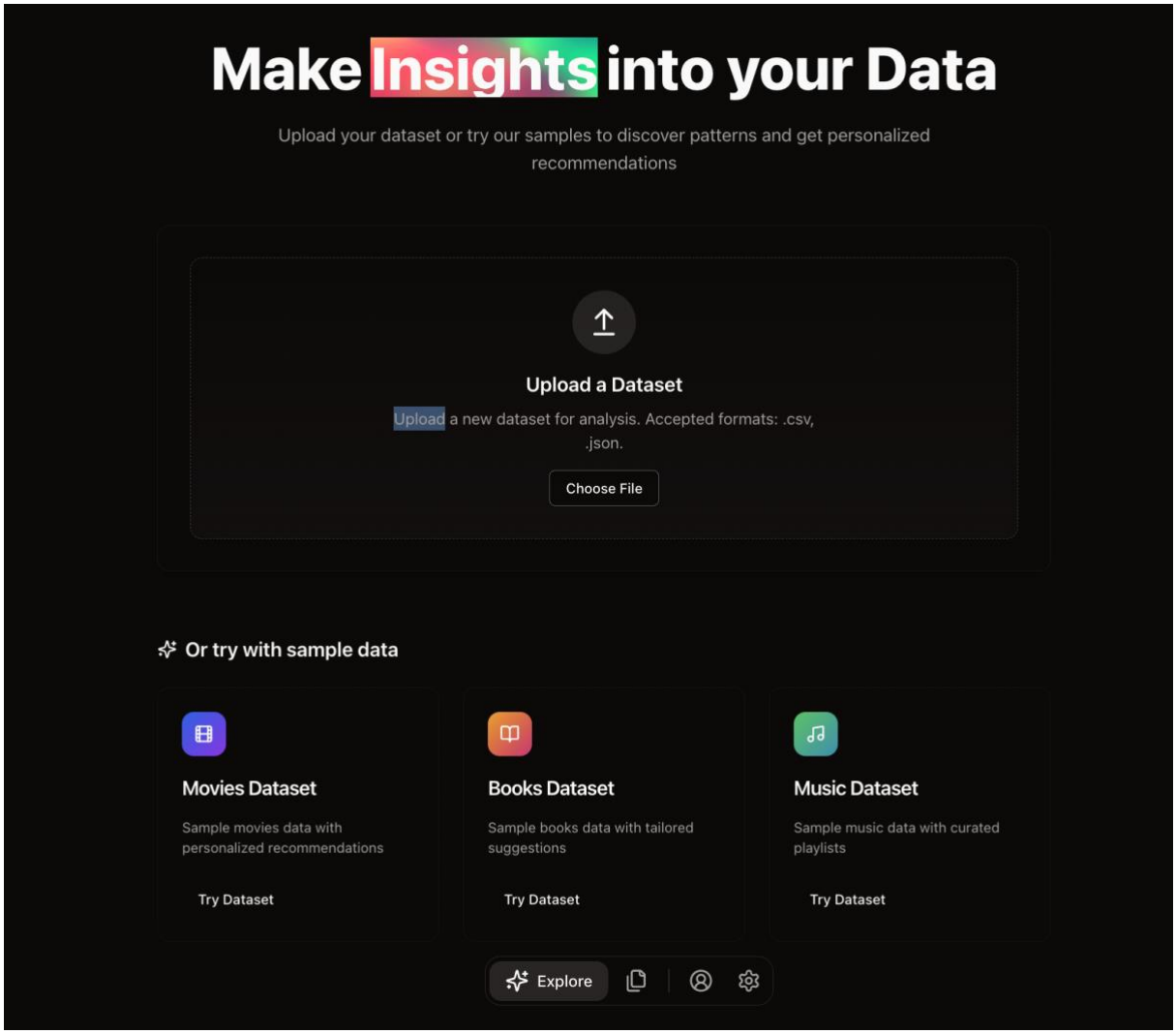

Рисунок 3.4 – Головна та Початкова сторінка датасету

Наприклад, користувач обирає «Movies Dataset» і після цього відбувається аналіз. Після завершення аналізу зразкових даних про фільми зображення показує екран системи. Користувач отримує повідомлення під назвою «Analysis Complete!», яке означає, що процес обробки інформації завершено, а підзаголовок інформує користувача про результати. Коли користувач натискає кнопку «Continue to Insights», він побачить результати у рекомендаційних запитань. Цей екран показує, як система перетворює сирі дані в корисні інсайти (рисунок 3.5).

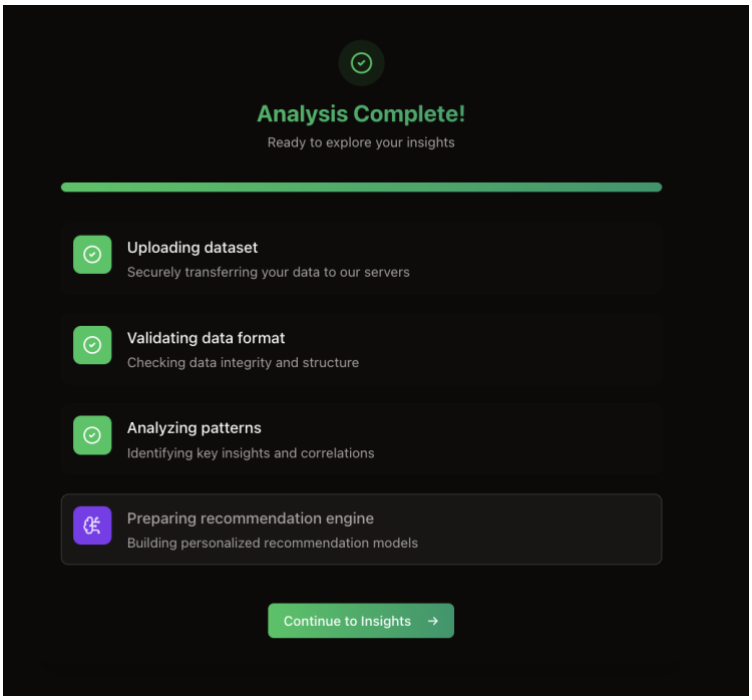

Рисунок 3.5 – Аналіз даних

Отже, після аналізу показано процес, за допомогою якого система створює рекомендаційні запитання за допомогою Gemini API, щоб зібрати інформацію про уподобання користувача, і щоб створити рекомендації, на основі його відповідей і конкретних потреб (рисунок 3.6).

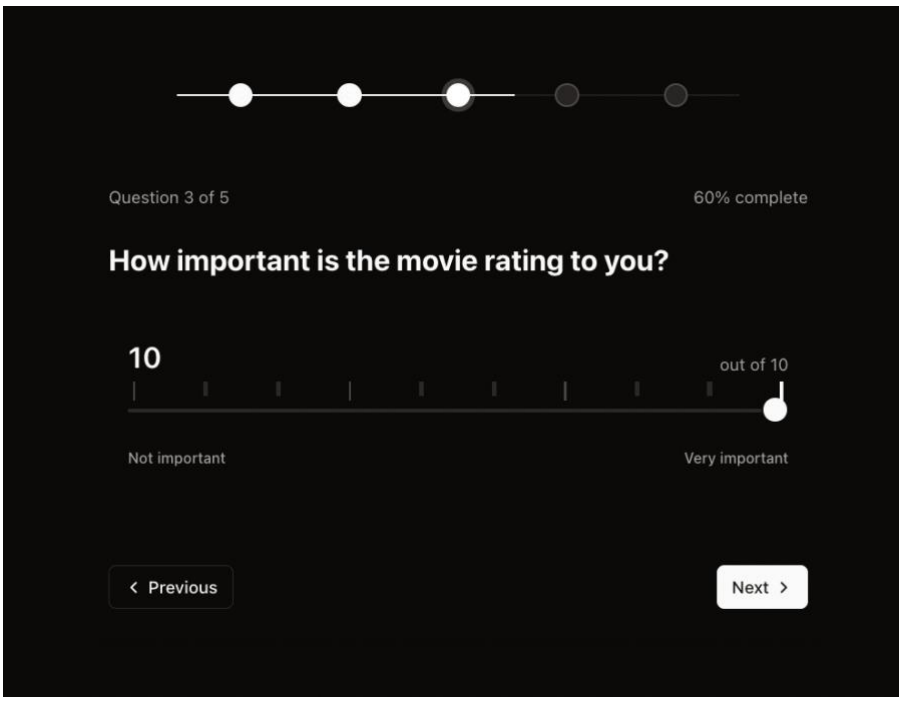

Рисунок 3.6 – Рекомендаційні питання

Система порекомендувала фільм «The Godfather» режисера Френсіса Форда Копполи. Система визначила 72% збігу з уподобаннями користувача, враховуючи його пріоритети щодо критично схвалених фільмів (рисунок 3.7). Основна інформація про фільм включає рейтинг, опис сюжету, акторський склад Марлон Брандо і Аль Пачіно. Рекомендація ґрунтується на ключових факторах такі як жанр, високий рейтинг і статус режисера. Система оцінює якість пропозиції як «High», 88% релевантності.

Користувач може впливати на майбутні рекомендації через кнопки «Remove Similar», «Notional» та «More Like This». Цей екран демонструє, як система поєднує структурний аналіз даних жанри, рейтинги із зворотнім зв'язком, щоб забезпечити персоналізований підхід, адаптований до індивідуальних уподобань.

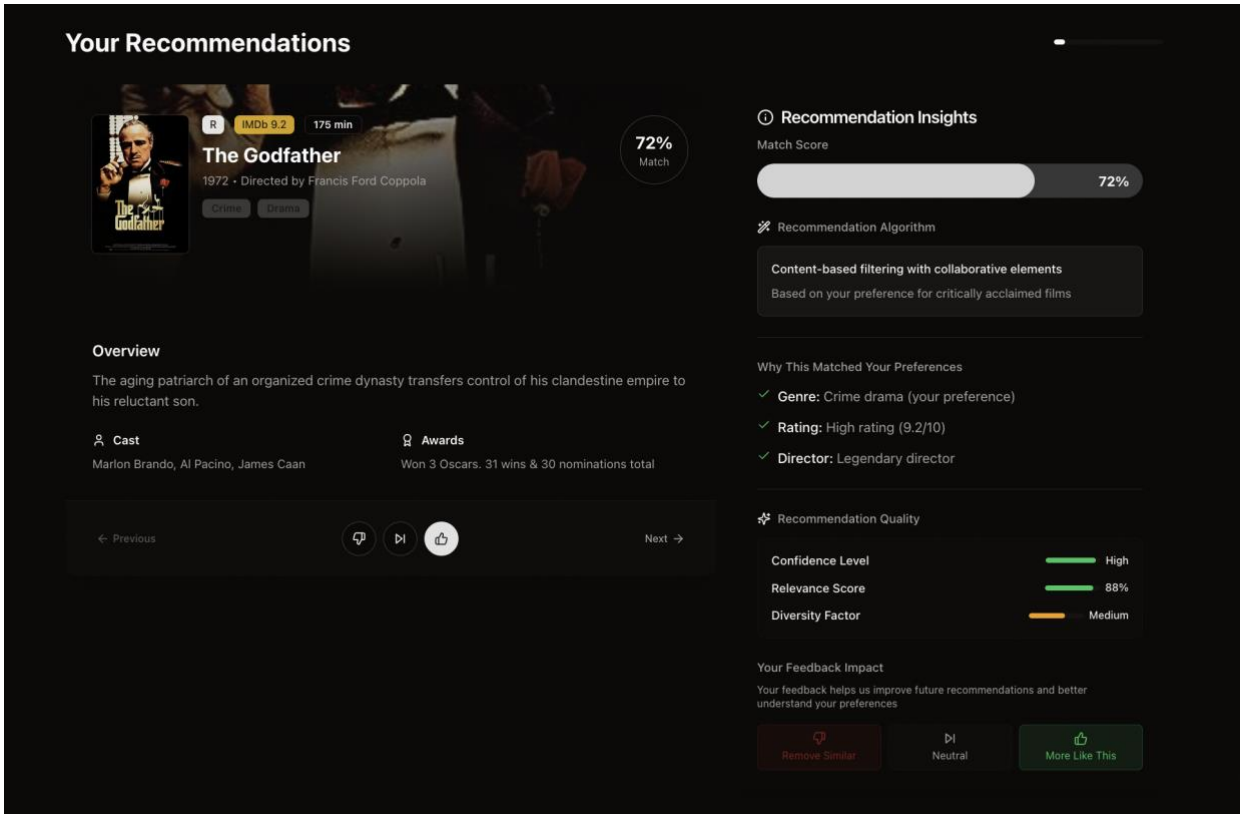

Рисунок 3.7 – Рекомендаційні питання

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи було успішно розроблено та протестовано програмну систему персоналізованих рекомендацій «Skunk», призначену для аналізу користувацьких даних та надання персональних пропозицій. Розробка системи базувалася на глибокому дослідженні предметної області, аналізі існуючих рішень та чітко визначених вимогах, що дозволило створити функціональний та актуальний програмний продукт.

На першому етапі роботи було проведено змістовний аналіз предметної області персоналізованих рекомендаційних систем. Було досліджено ключові типи даних, що використовуються для формування рекомендацій, розглянуто основні підходи та алгоритми, такі як колаборативна фільтрація, контентно-орієнтовані методи та гібридні моделі. Важливо також те що було проведено аналіз провідних платформ, таких як Netflix, Spotify та Welcome to the Jungle (Otta), що дозволило виявити як і позитивні сторони програм, так і зокрема проблему «холодного старту» та необхідність динамічної системи.

На основі проведеного аналізу було сформульовано функціональні та нефункціональні вимоги до програмної системи, з наголосом на її універсальності та інтерактивності, а також здатності навчатися на основі зворотного зв'язку.

Другий етап був присвячений проєктуванню програмного забезпечення. Було обрано для системи гібридну архітектуру, яка поєднує переваги різних технологічних стеків як Next.js для реалізації користувацького інтерфейсу та шару Backend-for-Frontend, і FastAPI на мові Python для спеціалізованого мікросервісу, який інкапсулює ядро рекомендаційної програми. Описано детально основні компоненти системи на кожному архітектурному рівні, їхні функції та взаємодію, що було візуалізовано за допомогою діаграми потоків даних та станів.

Також було спроєктовано структуру реляційної бази даних PostgreSQL для зберігання інформації про користувачів, датасети, згенеровані запитання, відповіді та зворотний зв'язок. Вибір технологічного стеку, що включає також Google Gemini API для аналізу даних та генерації запитань, Supabase Storage для зберігання файлів

та Clerk для автентифікації, був ретельно обґрунтований з точки зору функціональності, продуктивності та зручності розробки. Окрему увагу було приділено проєктуванню користувацького інтерфейсу UI/UX з метою забезпечення його інтуїтивності та простоти використання.

На третьому етапі було здійснено програмну реалізацію основних модулів. Детально описано процес розробки компонентів BFF, відповідальних за завантаження та підготовку датасетів, взаємодію з Gemini API для генерації запитань, збереження даних у PostgreSQL та оркестрацію запитів до мікросервісу рекомендацій.

Також було розглянуто реалізацію ключових алгоритмів персоналізації в рамках FastAPI мікросервісу, зокрема стратегії на основі векторних представлень із застосуванням Sentence Transformers та FAISS для генерації початкових рекомендацій та механізму навчання на основі зворотного зв'язку з використанням Vowpal Wabbit для динамічної адаптації. Наведено приклади кодової реалізації ключових аспектів фронтенд-частини, що відповідає за взаємодію з користувачем. Було описано комплексну методику тестування з використанням відповідних інструментів Jest, React Testing Library, PyTest, Cypress. Детально розглянуто підходи до вибору та підготовки тестових даних, включаючи як синтетичні, так і анонімізовані реальні набори.

Таким чином, поставлена мета кваліфікаційної роботи, а саме розроблення програмної системи персоналізованих рекомендацій на основі аналізу користувацьких даних, була досягнута.

Розроблена система програмна «Skunk» демонструє інноваційний підхід до персоналізованих рекомендацій, дозволяючи користувачам аналізувати власні набори даних та отримувати індивідуалізовані рекомендації завдяки інтеграції з сучасними інтелектуальними сервісами та алгоритмами машинного навчання.

Практичне значення результату полягало у створенні програми, яка може бути використана як основа для розробки комерційних або дослідницьких систем у сферах, де потрібен динамічний підхід до аналізу даних та надання персоналізованих пропозицій.

Можливі напрямки подальшого розвитку системи включають:

- розширення підтримуваних форматів вхідних даних;
- інтеграцію з більшою кількістю джерел даних;
- вдосконалення алгоритмів машинного навчання;
- можливість підтримувати не лише датасети.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Розуміння алгоритмів рекомендацій у соціальних мережах [Електронний ресурс] – Режим доступу: https://knightcolumbia.org/content/understanding-social-media-recommendation-algorithms (дата звернення – 16.01.2025). – Назва з екрану.
2. Як працюють алгоритми рекомендацій [Електронний ресурс] – Режим доступу: https://www.scientificamerican.com/article/how-recommendation-algorithms-work-and-why-they-may-miss-the-mark/ (дата звернення – 16.01.2025). – Назва з екрану.
3. Як ШІ приймає рішення [Електронний ресурс] – Режим доступу: https://www.scientificamerican.com/article/new-tool-reveals-how-ai-makes-decisions/ (дата звернення – 17.01.2025). – Назва з екрану.
4. Система рекомендацій на основі ШІ: типи, варіанти використання, розробка та впровадження [Електронний ресурс] – Режим доступу: https://www.leewayhertz.com/build-recommendation-system/ (дата звернення – 16.01.2025). – Назва з екрану.
5. Система рекомендацій на основі вмісту [Електронний ресурс] – Режим доступу: https://medium.com/%40bindhubalu/content-based-recommender-system-4db1b3de03e7 (дата звернення – 23.01.2025). – Назва з екрану.
6. Система рекомендацій на основі вмісту [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/ml-content-based-recommender-system/ (дата звернення – 23.01.2025). – Назва з екрану.
7. Посібник із фільтрації на основі вмісту в системах рекомендацій [Електронний ресурс] – Режим доступу: https://www.turing.com/kb/content-based-filtering-in-recommender-systems (дата звернення – 23.01.2025). – Назва з екрану.
8. Фільтрація на основі вмісту [Електронний ресурс] – Режим доступу: https://www.ibm.com/think/topics/content-based-filtering (дата звернення – 23.01.2025). – Назва з екрану.

9. Лінійна регресія [Електронний ресурс] – Режим доступу: https://msn.khmnu.edu.ua/pluginfile.php/647068/mod_resource/content/3/ML_Lab2.pdf (дата звернення – 23.01.2025). – Назва з екрану.
10. Метод опорних векторів [Електронний ресурс] – Режим доступу: https://msn.khmnu.edu.ua/pluginfile.php/647058/mod_resource/content/1/Lec_Support_vector_machines.pdf (дата звернення – 05.02.2025). – Назва з екрану.
11. Як побудувати ефективну рекомендаційну систему [Електронний ресурс] – Режим доступу: https://dou.ua/forums/topic/47504/ (дата звернення – 05.02.2025). – Назва з екрану.
12. Приклади рекомендаційних систем [Електронний ресурс] – Режим доступу: https://research.aimultiple.com/recommendation-system/ (дата звернення – 05.02.2025). – Назва з екрану.
13. Історія рекомендаційної системи [Електронний ресурс] – Режим доступу: https://arxiv.org/pdf/2209.01860 (дата звернення – 05.02.2025). – Назва з екрану.
14. Tapestry Routing example [Електронний ресурс] – Режим доступу: https://www.researchgate.net/figure/Tapestry-routing-example-Path-taken-by-a-message-from-node-0325-for-node-4598-in_fig1_2494137 (дата звернення – 07.02.2025). – Назва з екрану.
15. Tapestry [Електронний ресурс] – Режим доступу: https://www.slideshare.net/slideshow/tapestry-231544636/231544636 (дата звернення – 07.02.2025). – Назва з екрану.
16. Пояснення рекомендаційної системи Spotify [Електронний ресурс] – Режим доступу: https://www.spotify.com/us/safetyandprivacy/understanding-recommendations (дата звернення – 07.02.2025). – Назва з екрану.
17. Annoy [Електронний ресурс] – Режим доступу: https://github.com/spotify/annoy (дата звернення – 07.02.2025). – Назва з екрану.
18. Voyager [Електронний ресурс] – Режим доступу: https://github.com/spotify/voyager (дата звернення – 07.02.2025). – Назва з екрану.

19. Voyager Explanation [Електронний ресурс] – Режим доступу: https://www.youtube.com/watch?v=kOpL1NbvlM4 (дата звернення – 07.02.2025). – Назва з екрану.
20. Bert Explanation [Електронний ресурс] – Режим доступу: https://h2o.ai/wiki/bert/ (дата звернення – 07.02.2025). – Назва з екрану.
21. Рекомендації Netflix [Електронний ресурс] – Режим доступу: https://research.netflix.com/research-area/recommendations (дата звернення – 07.02.2025). – Назва з екрану.
22. Пояснення рекомендаційної системи Netflix [Електронний ресурс] – Режим доступу: https://help.netflix.com/en/node/100639 (дата звернення – 07.02.2025). – Назва з екрану.
23. Що таке гібридна архітектура? [Електронний ресурс] – Режим доступу: https://www.ibm.com/think/topics/hybrid-cloud-architecture (дата звернення – 11.03.2025). – Назва з екрану.
24. Стиль мікросервісної архітектури [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices (дата звернення – 11.03.2025). – Назва з екрану.
25. Монолітна архітектура в програмному забезпеченні [Електронний ресурс] – Режим доступу: https://www.techtarget.com/whatis/definition/monolithic-architecture (дата звернення – 11.03.2025). – Назва з екрану.
26. Монолітна архітектура [Електронний ресурс] – Режим доступу: https://www.ibm.com/think/topics/monolithic-architecture (дата звернення – 11.03.2025). – Назва з екрану.
27. Побудова гібридної архітектури: використання потужності мікросервісів і монолітів [Електронний ресурс] – Режим доступу: https://medium.com/@pandeyarpit88/building-a-hybrid-architecture-harnessing-the-power-of-microservices-and-monoliths-80f79b310d75 (дата звернення – 11.03.2025). – Назва з екрану.

28. Data Flow Diagram [Електронний ресурс] – Режим доступу: https://www.visual-paradigm.com/guide/data-flow-diagram/what-is-data-flow-diagram/ (дата звернення – 11.03.2025). – Назва з екрану.
29. Levels in DFD [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/levels-in-data-flow-diagrams-dfd/ (дата звернення – 13.03.2025). – Назва з екрану.
30. Топ 10 СУБД [Електронний ресурс] – Режим доступу: https://www.itta.net/en/blog/top-10-best-databases-to-use-in-2024/ (дата звернення – 13.03.2025). – Назва з екрану.
31. Монго БД [Електронний ресурс] – Режим доступу: https://www.mongodb.com/ (дата звернення – 13.03.2025). – Назва з екрану.
32. PostgreSQL [Електронний ресурс] – Режим доступу: https://www.postgresql.org/ (дата звернення – 13.03.2025). – Назва з екрану.
33. ERD [Електронний ресурс] – Режим доступу: https://www.lucidchart.com/pages/er-diagrams (дата звернення – 13.03.2025). – Назва з екрану.
34. Причини використання коду Visual Studio для інтегрованого середовища розробки [Електронний ресурс] – Режим доступу: https://medium.com/codex/reasons-to-use-visual-studio-code-for-your-integrated-development-environment-9d242da8b1aa (дата звернення – 15.03.2025). – Назва з екрану.
35. JetBrains IDE відгуки [Електронний ресурс] – Режим доступу: https://www.peerspot.com/products/jetbrains-ides-reviews#products-show (дата звернення – 15.03.2025). – Назва з екрану.
36. Visual Studio Code [Електронний ресурс] – Режим доступу: https://code.visualstudio.com/ (дата звернення – 15.03.2025). – Назва з екрану.
37. Переваги та недоліки Next.js [Електронний ресурс] – Режим доступу: https://pagepro.co/blog/pros-and-cons-of-nextjs/ (дата звернення – 15.03.2025). – Назва з екрану.

38. Next.js [Електронний ресурс] – Режим доступу: https://nextjs.org/ (дата звернення – 15.03.2025). – Назва з екрану.
39. Preline UI [Електронний ресурс] – Режим доступу: https://preline.co/ (дата звернення – 09.04.2025). – Назва з екрану.
40. Gemini API [Електронний ресурс] – Режим доступу: https://ai.google.dev/api?lang=python (дата звернення – 09.04.2025). – Назва з екрану.
41. E2E Testing [Електронний ресурс] – Режим доступу: https://docs.cypress.io/app/get-started/why-cypress#End-to-end-Testing (дата звернення – 09.04.2025). – Назва з екрану.

ДОДАТОК А

Код програмних модулів

Наведений код програмної реалізації збирає відповіді на опитувальник і передає їх далі для генерації персоналізованих рекомендацій:

```
import React, { useState, FormEvent, useEffect } from 'react';
import { Question, QuestionResponse, UserResponses } from
'./questions/QuestionTypes';
import QuestionRenderer from './questions/QuestionRenderer';

interface PreferenceQuestionnaireProps {
   questions: Question[];
   onComplete: (responses: UserResponses) => void;
   isLoading?: boolean;
}

const PreferenceQuestionnaire: React.FC<PreferenceQuestionnaireProps> = ({
   questions,
   onComplete,
   isLoading = false,
}) => {
   const [responses, setResponses] = useState<Record<string,
QuestionResponse>>({});
   const [allAnswered, setAllAnswered] = useState(false);

   useEffect(() => {
      const answeredCount = Object.keys(responses).length;
      setAllAnswered(answeredCount === questions.length && questions.length > 0);
   }, [responses, questions]);

   const handleResponseChange = (response: QuestionResponse) => {
      setResponses(prev => ({ ...prev, [String(response.questionId)]: response
}));
   };

   const handleSubmit = (e: FormEvent) => {
      e.preventDefault();
      if (allAnswered) {
        const responseArray = Object.values(responses);
        onComplete(responseArray);
      }
   };

   if (questions.length === 0 && !isLoading) {
    return <p className="text-muted-foreground">No questions available for this
dataset yet.</p>;
   }

   return (
      <form onSubmit={handleSubmit} className="space-y-6">
         {questions.map(question => (
            <QuestionRenderer
               key={question.id}
               question={question}
               value={responses[String(question.id)]?.value}
```

```
                onChange={handleResponseChange}
                disabled={isLoading}
            />
          ))}
          {questions.length > 0 && (
             <div className="mt-8">
                 <button
                 type="submit"
                 disabled={!allAnswered || isLoading}
                 className="px-6 py-2 rounded-lg text-white font-medium bg-blue-600
hover:bg-blue-700 disabled:bg-gray-400 disabled:cursor-not-allowed"
                 >
                 {isLoading ? 'Processing...' : 'Get Recommendations'}
                 </button>
             </div>
          )}
       </form>
   );
};
```

Наведено код програмної реалізації ілюструє ключову логіку завантаження датасету:

```
import { NextRequest, NextResponse } from 'next/server';
import Papa from 'papaparse';
import { writeFile, unlink } from 'fs/promises';
import path from 'path';
import os from 'os';

interface FileUploadRequest {
   fileContent: string;
   fileName: string;
}

const processCSVData = (csvContent: string): string => {
   const { data: rows, errors } = Papa.parse(csvContent, {
      header: true,
      skipEmptyLines: 'greedy',
   });

   if (errors.length > 0) {
      console.error('CSV parsing errors:', errors.map(e => e.message).join('; '));
      throw new Error(`Failed to parse CSV data due to: ${errors[0].message}`);
   }
    if (!rows || rows.length === 0) {
        throw new Error('CSV file is empty or contains only headers.');
    }

   const processedData = (rows as Record<string, unknown>[]).map((row, index:
number) => ({
      sknq_id: index + 1,
      ...row,
   }));

   return Papa.unparse(processedData);
};

export async function POST(req: NextRequest) {
   try {
      const { fileContent, fileName } = (await req.json()) as FileUploadRequest;
```

```
      if (!fileContent || !fileName || !fileName.toLowerCase().endsWith('.csv')) {
         return NextResponse.json({ error: 'Invalid file data or not a CSV file.'
}, { status: 400 });
      }

      let processedContent;
      try {
         processedContent = processCSVData(fileContent);
      } catch (error: any) {
         return NextResponse.json({ error: `Error processing CSV:
${error.message}` }, { status: 400 });
      }

      const tempFilePath = path.join(os.tmpdir(), `processed-${Date.now()}-
${fileName}`);
      await writeFile(tempFilePath, processedContent, 'utf-8');

      const supabasePath = `supabase://sknq-datasets/${fileName}`;
      const googleAIFileUri = `google-ai-file-manager://files/${fileName}`;
      const supabasePath = await uploadToSupabase(tempFilePath, fileName);
      const googleAIFileUri = await getGoogleAIFileUri(tempFilePath, fileName);


      await unlink(tempFilePath);

     await prisma.dataset.create({ data: { userId, fileName, supabasePath,
googleAIFileUri } });

      return NextResponse.json({
        success: true,
        fileUri: googleAIFileUri,
        fileName: fileName,
        mimeType: 'text/csv',
        datasetId: supabasePath
      }, { status: 200 });

    } catch (error: any) {
      console.error('File upload error:', error);
      return NextResponse.json({ error: `Internal Server Error: ${error.message}`
}, { status: 500 });
    }
}
```

Цей React-компонент HomePage реалізує повний процес взаємодії користувача з системою рекомендацій від завантаження датасету до отримання персоналізованих рекомендацій.

```
import React, { useState, useCallback } from 'react';
import PreferenceQuestionnaire from '@/app/components/PreferenceQuestionnaire';
import { Question, UserResponses } from '@/app/components/questions/QuestionTypes';

export default function HomePage() {
  const [file, setFile] = useState<File | null>(null);
  const [status, setStatus] = useState<string | null>(null);
  const [error, setError] = useState<string | null>(null);
  const [questions, setQuestions] = useState<Question[]>([]);
  const [recommendations, setRecommendations] = useState<any[] | null>(null);
```

```
  const [datasetId, setDatasetId] = useState<string | null>(null);
  const [processing, setProcessing] = useState(false);

  const handleFileChange = useCallback((e: React.ChangeEvent<HTMLInputElement>) => {
    const f = e.target.files?.[0];
    if (!f || f.type !== 'text/csv') {
      setError('Please upload a valid CSV file.');
      return;
    }
    setFile(f);
    setError(null);
    setQuestions([]);
    setRecommendations(null);
    setDatasetId(null);
  }, []);

  const handleUpload = useCallback(async () => {
    if (!file) return;
    setProcessing(true);
    try {
      const fileText = await file.text();
      const res = await fetch('/api/files', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ fileName: file.name, fileContent: fileText }),
      });
      const upload = await res.json();
      setDatasetId(upload.datasetId);

      const qRes = await fetch('/api/questions', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(upload),
      });
      const { response } = await qRes.json();
      const parsed = JSON.parse(response);

      if (Array.isArray(parsed.questions)) {
        setQuestions(parsed.questions.map((q: any, i: number) => ({ id: q.id || i +
1, ...q })));
        setStatus('Questions generated. Please answer below.');
      } else {
        throw new Error('No questions generated.');
      }
    } catch (e: any) {
      setError(e.message);
    } finally {
      setProcessing(false);
    }
  }, [file]);

  const handleComplete = async (answers: UserResponses) => {
    if (!datasetId) return;
    setProcessing(true);
    try {
      const res = await fetch('/api/recommendation', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ preferences: answers, datasetId }),
      });
      const data = await res.json();
      setRecommendations(data.items);
```

```
      setStatus('Recommendations ready.');
    } catch (e: any) {
      setError(e.message);
    } finally {
      setProcessing(false);
    }
  };

  return (
    <div className="container mx-auto p-4">
      <section className="mb-8 bg-card p-6 rounded-lg shadow-lg">
        <h2 className="text-2xl font-semibold mb-4">1. Upload Your Dataset</h2>
        <input type="file" accept=".csv" onChange={handleFileChange} className="mb-4"
/>
        <button onClick={handleUpload} disabled={!file || processing}
className="btn">
          {processing ? 'Analyzing...' : 'Upload & Analyze'}
        </button>
        {status && <p>{status}</p>}
        {error && <p className="text-red-500">{error}</p>}
      </section>

      {questions.length > 0 && !recommendations && (
        <section className="mb-8 bg-card p-6 rounded-lg shadow-lg">
          <h2 className="text-2xl font-semibold mb-4">2. Answer Questions</h2>
          <PreferenceQuestionnaire questions={questions} onComplete={handleComplete}
isLoading={processing} />
        </section>
      )}

      {recommendations && (
        <section className="bg-card p-6 rounded-lg shadow-lg">
          <h2 className="text-2xl font-semibold mb-4">3. Your Recommendations</h2>
          {recommendations.length > 0 ? (
            recommendations.map((item, i) => (
              <div key={i} className="border p-4 rounded mb-2">
                <h3 className="font-bold">{item.title || `Item ${i + 1}`}</h3>
                <p>Score: {item.match_score?.toFixed(2)}</p>
              </div>
            ))
          ) : (
            <p>No recommendations found for your preferences.</p>
          )}
        </section>
      )}
    </div>
  );
}
```

Після успішного запису фідбеку, система може ініціювати запит на отримання оновлених, вже більш персоналізованих, рекомендацій, викликавши, функцію handleQuestionnaireComplete з поточними або оновленими преференціями.

```
interface RecommendationItem {
  sknq_id: string;
  title?: string;
  match_score?: number;
}
```

```
const submitFeedback = async (
  value: -1 | 0 | 1,
  item: RecommendationItem,
  responses: UserResponses
) => {
  if (!userId || !item.sknq_id || !currentDatasetId) {
    setError("Missing user, item or dataset data");
    return;
  }

  setIsProcessing(true);
  try {
    const res = await fetch('/api/feedback', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        itemId: item.sknq_id,
        feedback: value,
        datasetId: currentDatasetId,
        userId
      })
    });

    if (!res.ok) throw new Error((await res.json()).error || 'Feedback failed');

    if (responses?.length > 0) {
      await handleQuestionnaireComplete(responses);
    }
  } catch (err: any) {
    setError(`Feedback error: ${err.message}`);
  } finally {
    setIsProcessing(false);
  }
};

return (
  recommendations?.items?.length > 0 && (
    <section className="p-6 rounded-lg shadow-lg bg-card">
      <h2 className="text-2xl font-semibold mb-4">3. Your Recommendations</h2>
      {recommendations.items.map((item, i) => (
        <div key={item.sknq_id || i} className="p-4 mb-2 border rounded-md">
          <h3 className="font-bold">{item.title || `Item ${item.sknq_id}`}</h3>
          <p>Match Score: {item.match_score?.toFixed(2)}</p>
          <div className="mt-2 space-x-2">
            <button
              onClick={() => submitFeedback(1, item,
Object.values(responsesStateFromQuestionnaire))}
              className="px-4 py-1 bg-green-600 text-white rounded"
            >
              Like
            </button>
            <button
              onClick={() => submitFeedback(0, item,
Object.values(responsesStateFromQuestionnaire))}
              className="px-4 py-1 bg-gray-500 text-white rounded"
            >
              Neutral
            </button>
            <button
              onClick={() => submitFeedback(-1, item,
Object.values(responsesStateFromQuestionnaire))}
```

```
              className="px-4 py-1 bg-red-600 text-white rounded"
            >
              Dislike
            </button>
          </div>
        </div>
      ))}
    </section>
  )
);
```

Для автоматизації перевірки цього сценарію на рівні BFF було розроблено інтеграційні тести з використанням Jest.

```
import { POST as filesHandler } from '@/app/api/files/route';
import { POST as questionsHandler } from '@/app/api/questions/route';
import { NextRequest } from 'next/server';

jest.mock('@/lib/supabase-client', () => ({
  supabase: {
    storage: {
      from: jest.fn().mockReturnThis(),
      upload: jest.fn().mockResolvedValue({ data: { path: 'test-file.csv' }, error:
null }),
    },
  },
}));
jest.mock('@google/generative-ai', () => ({
    GoogleGenerativeAI: jest.fn().mockImplementation(() => ({
        getGenerativeModel: jest.fn().mockReturnThis(),
        generateContent: jest.fn().mockResolvedValue({
            response: {
                candidates: [{ content: { parts: [{ text: JSON.stringify({ questions:
[{id:1, question:"Q1", type: "SINGLE_CHOICE", choices: ["A", "B"], column: "col1",
column_type: "categorical"}], dataset_analysis: { is_suitable_for_analysis: true,
reason: "OK", metadata: {domain_type: "movies", record_count: 100, key_features:
["genre", "rating"]} }}) }] } }]
            }
        }),
    })),
    HarmCategory: { HARM_CATEGORY_HARASSMENT: 'HARM_CATEGORY_HARASSMENT',
HARM_CATEGORY_HATE_SPEECH: 'HARM_CATEGORY_HATE_SPEECH' },
    HarmBlockThreshold: { BLOCK_ONLY_HIGH: 'BLOCK_ONLY_HIGH', BLOCK_MEDIUM_AND_ABOVE:
'BLOCK_MEDIUM_AND_ABOVE' },
}));


describe('API Dataset Processing Flow', () => {
  it('should successfully upload a valid CSV file, trigger analysis, and return
generated questions', async () => {
    const mockFileContent = "id,title,genre\n1,Movie A,Action\n2,Movie B,Comedy";
    const mockFileName = "dataset.csv";

    const fileReq = new NextRequest('http://localhost/api/files', {
      method: 'POST',
      body: JSON.stringify({ fileContent: mockFileContent, fileName: mockFileName }),
    });
```

```
    const fileRes = await filesHandler(fileReq);
    const fileData = await fileRes.json();

    expect(fileRes.status).toBe(200);
    expect(fileData.success).toBe(true);
    expect(fileData.fileUri).toContain(mockFileName);
    expect(fileData.datasetId).toBeDefined();

    const questionReq = new NextRequest('http://localhost/api/questions', {
      method: 'POST',
      body: JSON.stringify({
        fileUri: fileData.fileUri,
        mimeType: 'text/csv',
        datasetId: fileData.datasetId
      }),
    });
    const questionRes = await questionsHandler(questionReq);
    const questionData = await questionRes.json();
    const parsedQuestionResponse = JSON.parse(questionData.response);

    expect(questionRes.status).toBe(200);
    expect(parsedQuestionResponse.questions).toBeDefined();
    expect(parsedQuestionResponse.questions.length).toBeGreaterThan(0);
    expect(parsedQuestionResponse.questions[0].question).toEqual("Q1");

expect(parsedQuestionResponse.dataset_analysis.is_suitable_for_analysis).toBe(true);
  });

  it('should return an error for non-CSV file upload attempts', async () => {
    const mockFileContent = "this is not a csv";
    const mockFileName = "test.txt";
    const fileReq = new NextRequest('http://localhost/api/files', {
        method: 'POST',
        body: JSON.stringify({ fileContent: mockFileContent, fileName: mockFileName
}),
    });
    const fileRes = await filesHandler(fileReq);
    expect(fileRes.status).toBe(400);
    const fileData = await fileRes.json();
    expect(fileData.error).toContain('Invalid file data or not a CSV file.');
  });
});


        import pytest
        import pandas as pd
        from app.services.vw_feedback_learner import VWFeedbackLearner # Припустимий
  шлях
        from unittest.mock import MagicMock, patch

        @pytest.fixture
        def vw_learner():
            learner = VWFeedbackLearner(model_dir="test_vw_models")
            # Мокуємо реальний VW екземпляр, щоб не створювати/завантажувати файли
            mock_vw_instance = MagicMock()
            mock_vw_instance.predict.return_value = [0.8, 0.1, 0.05, 0.03, 0.02] #
  Приклад ймовірностей/оцінок
            learner._get_or_create_model = MagicMock(return_value=mock_vw_instance)
            learner.save_user_model = MagicMock() # Мокуємо збереження
            return learner
```

```
    def test_rerank_recommendations_returns_top_n_with_scores(vw_learner):
        user_id = "test_user_1"
        candidate_items_details = [
            {"sknq_id": "item1", "title": "Item 1", "featureA": "X"},
            {"sknq_id": "item2", "title": "Item 2", "featureA": "Y"},
            {"sknq_id": "item3", "title": "Item 3", "featureA": "X"},
            {"sknq_id": "item4", "title": "Item 4", "featureA": "Z"},
            {"sknq_id": "item5", "title": "Item 5", "featureA": "Y"},
        ]
        user_profile_features = {"preferenceA": "X_is_high"}
        # Припустимо, що df_all_items та df_columns передаються або доступні в
сервісі
        # Для тесту ми можемо передати лише список колонок, які використовуються
для фіч
        df_columns_for_features = ["featureA"]
        top_n = 3

        # Мокуємо _extract_features_as_vw_string, щоб повернути прості рядки
        with      patch.object(vw_learner,      '_extract_features_as_vw_string',
return_value="|features mock_feature=1"):
            reranked = vw_learner.rerank_recommendations(
                user_id,      candidate_items_details,      user_profile_features,
df_columns_for_features, top_n
            )

        assert len(reranked) == top_n
        for item in reranked:
            assert "vw_score" in item

        # Перевіряємо, що predict був викликаний
        vw_learner._get_or_create_model.assert_called_with(user_id)
        model_mock = vw_learner._get_or_create_model(user_id)
        assert model_mock is not None
        model_mock.predict.assert_called_once()


    def test_record_feedback_calls_learn(vw_learner):
        user_id = "test_user_2"
        recommendations_shown = [{"sknq_id": "itemA", "featureX": 1}, {"sknq_id":
"itemB", "featureX": 2}]
        chosen_item_sknq_id = "itemA"
        feedback_value = 1
        user_profile = {"age_group": "adult"}
        df_cols = ["featureX"]

        with      patch.object(vw_learner,      '_extract_features_as_vw_string',
return_value="|features mock_feature=1"), \
             patch.object(vw_learner.vw_models_config.get(user_id,
{}).get('model', MagicMock()), 'predict', return_value=[0.7, 0.3]):

            vw_learner.record_feedback(user_id,           recommendations_shown,
chosen_item_sknq_id, feedback_value, user_profile, df_cols)

        model_mock = vw_learner._get_or_create_model(user_id)
        assert model_mock is not None
        model_mock.learn.assert_called_once()
        assert vw_learner.vw_models_config[user_id]['feedback_count'] == 1
```

ДОДАТОК Б

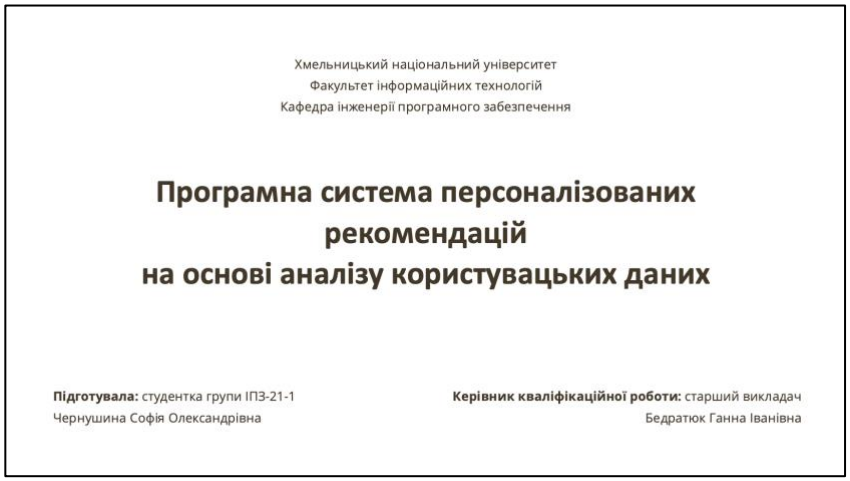

Рисунок Б.1 – Слайд 1

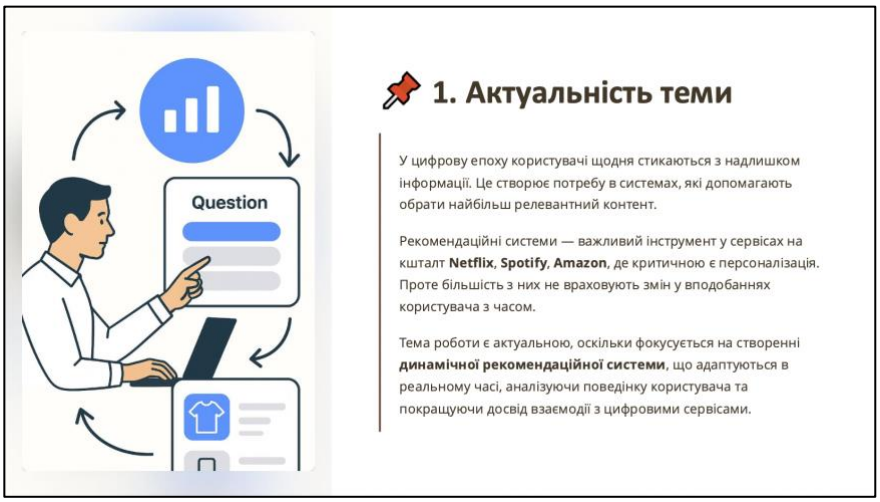

Рисунок Б.2 – Слайд 2

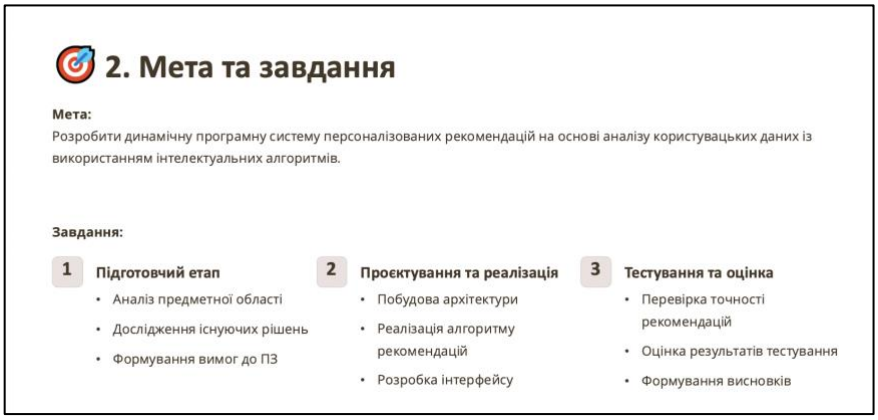

Рисунок Б.3 – Слайд 3

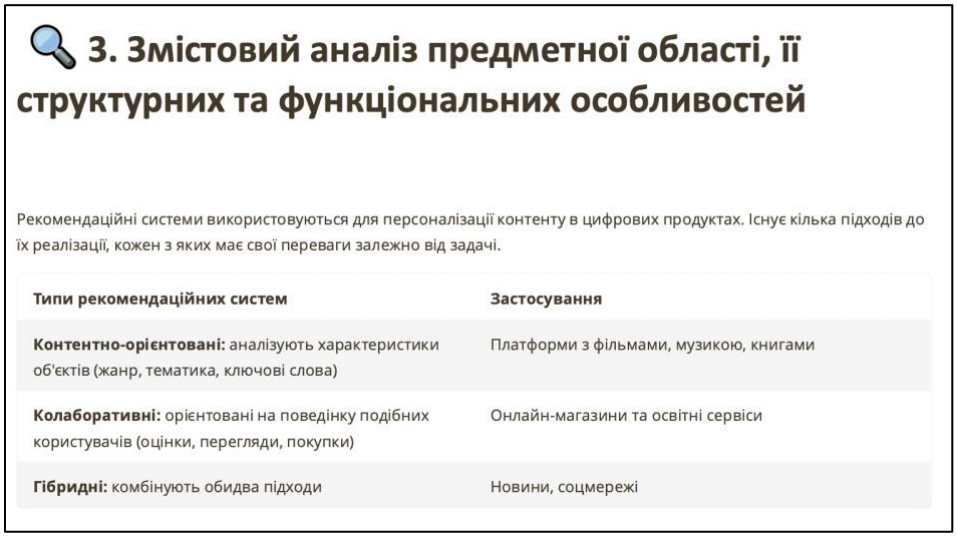

Рисунок Б.4 – Слайд 4

Рисунок Б.5 – Слайд 5

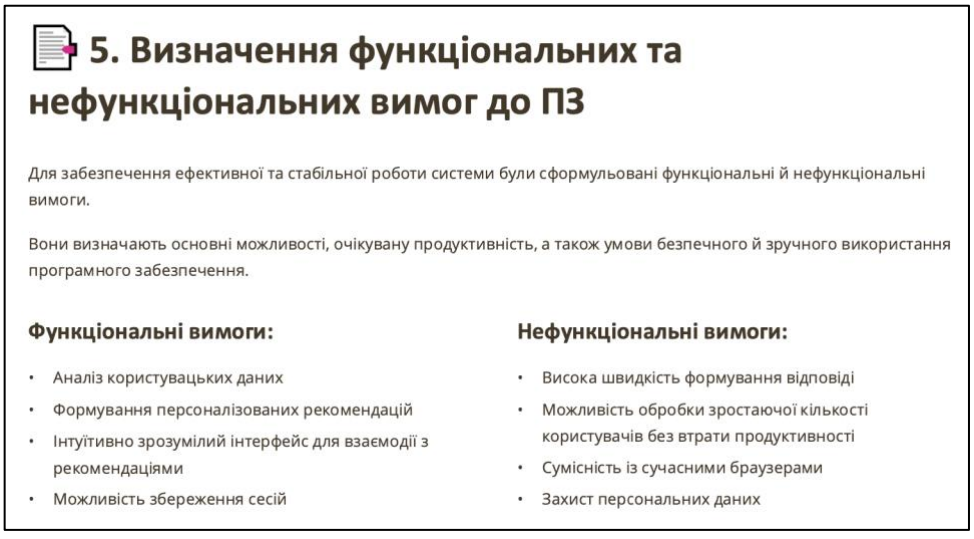

Рисунок Б.6 – Слайд 6

Рисунок Б.7 – Слайд 7

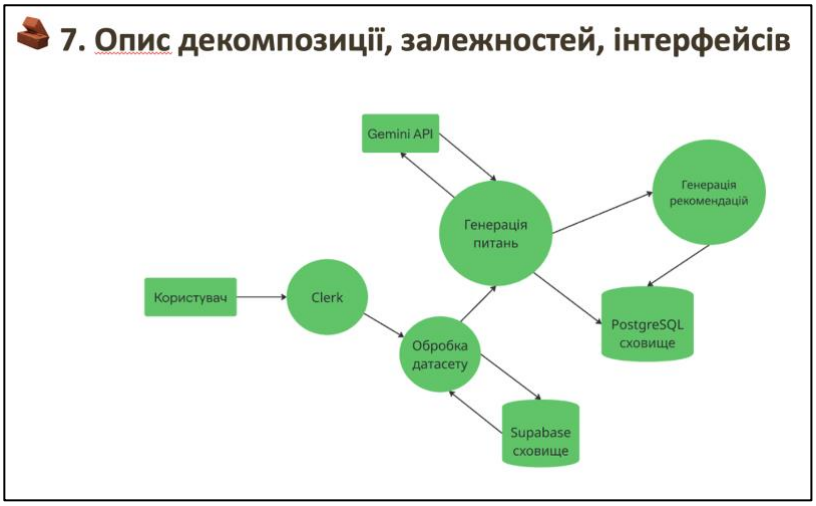

Рисунок Б.8 – Слайд 8

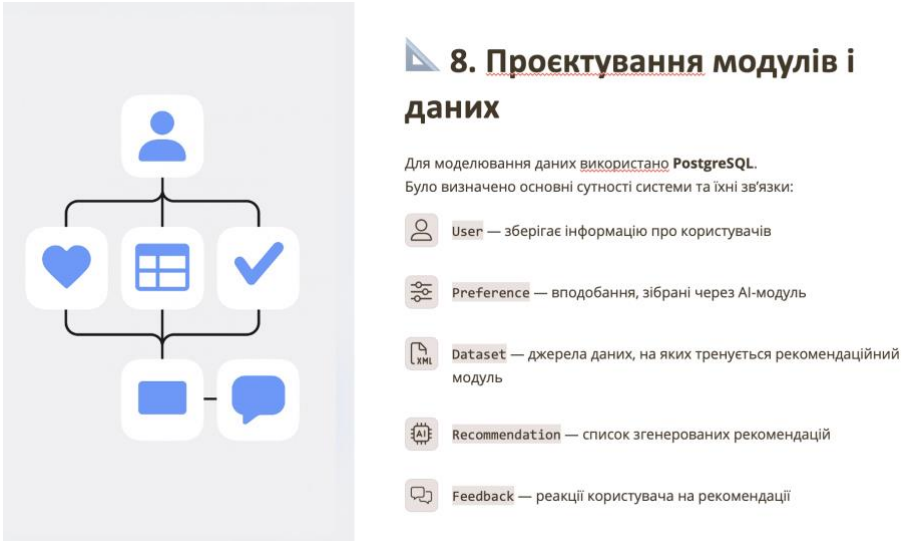

Рисунок Б.9 – Слайд 9

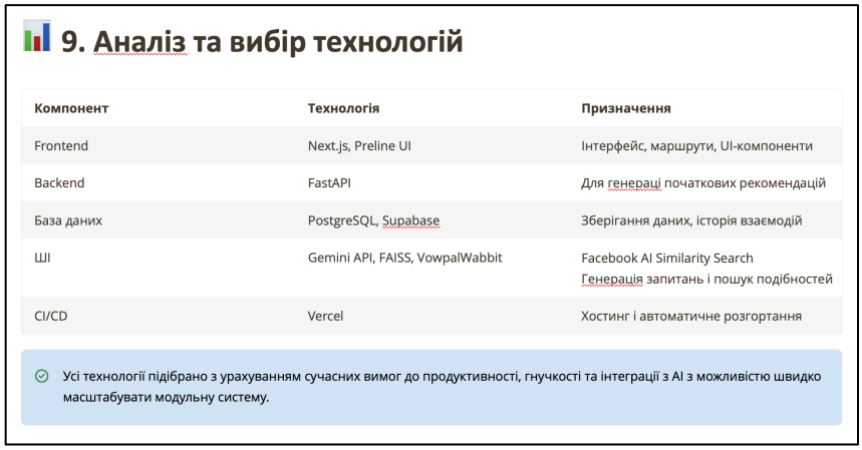

Рисунок Б.10 – Слайд 10

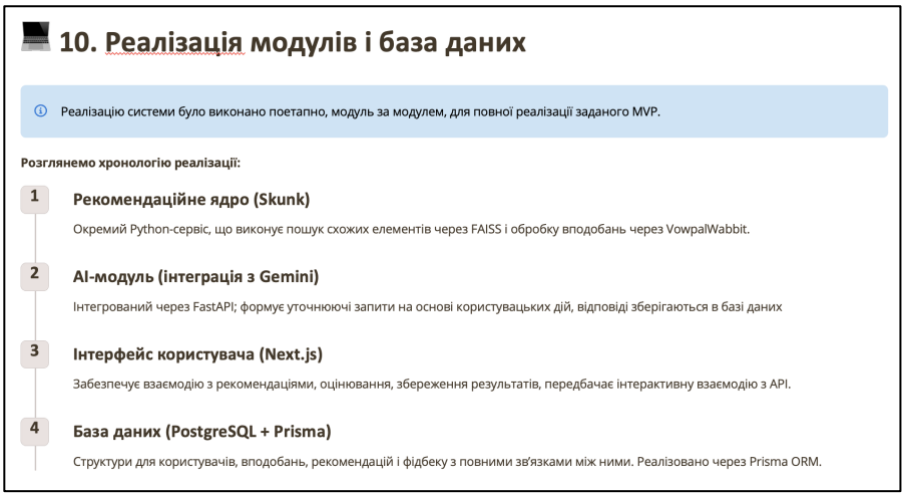

Рисунок Б.11 – Слайд 11

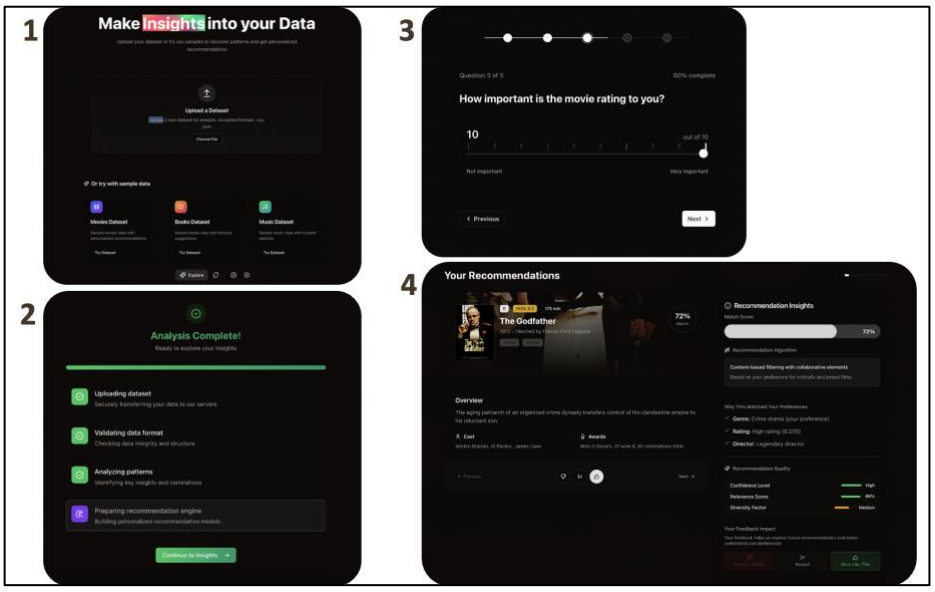

Рисунок Б.12 – Слайд 12

11. Вимоги до технічного та програмного забезпечення

Система потребує встановлення сучасних компонентів для забезпечення стабільної роботи та підтримки всіх функціональних можливостей:

- **Node.js 18+** — для запуску Next.js-інтерфейсу та скриптів
- **PostgreSQL 14+** — як основна система керування базами даних
- **Сучасний браузер (Chrome, Arc, Safari, Brave)** — для коректної роботи клієнтського інтерфейсу

Обладнання:

Для розгортання та тестування системи достатньо базових ресурсів середнього рівня:

- **CPU:** не менше 4 ядер для обробки запитів та AI-обчислень
- **RAM:** рекомендовано від 8 ГБ для збереження станів і обробки даних
- **Операційна система:** підтримка Windows, macOS або Linux

Рисунок Б.13 – Слайд 13

12. Тестування ПЗ

Було проведено всебічне тестування функціональних та інтеграційних компонентів системи для забезпечення її стабільності й продуктивності.

- **Юніт-тести (Jest):** перевірено основні функції ядра рекомендацій, компоненти інтерфейсу та валідацію даних
- **E2E тестування (Cypress):** імітовано сценарії реального використання: реєстрація, отримання рекомендацій, збереження вподобань

Результати тестування:

- Усі тести пройдено **успішно** (72 з 72 ✅)
- **Точність рекомендацій**: понад 85%
- **Середній час відповіді**: 300–400 мс
- **Стабільність**: підтримка ≥10 одночасних сесій без збоїв

Тестування засвідчило готовність MVP до реального використання.

```
> npm run test

PASS  src/core/recommendation.test.ts
√ generates relevant recommendations (112 ms)
√ filters based on feedback (87 ms)
√ handles cold-start users (99 ms)

PASS  src/api/feedback.test.ts
√ accepts and stores user feedback (94 ms)
√ rejects malformed input (82 ms)
√ returns success response (71 ms)

PASS  src/api/files.test.ts
√ uploads file correctly (101 ms)
√ validates file type (88 ms)
√ stores metadata (95 ms)

PASS  src/api/questions.test.ts
√ saves valid question (93 ms)
√ rejects spammy input (77 ms)
√ fetches recent questions (90 ms)

...

Test Suites:  14 passed, 14 total
Tests:        72 passed, 72 total
Snapshots:    0 total
Time:         35.126 s
Ran all test suites.
```

Рисунок Б.14 – Слайд 14

13. Висновки

№	Технологія	Призначення
1	Дослідження предметної області та постановка задачі	Проведено аналіз цифрових сервісів, де критична персоналізація (Otta, Netflix тощо); досліджено алгоритми рекомендацій; визначено проблему відсутності адаптації до змін уподобань; сформульовано мету та завдання проєкту
2	Проєктування програмного забезпечення	Розроблено архітектуру системи з використанням Next.js, Python backend і Supabase; визначено модулі: генерація питань, збереження відповідей, формування рекомендацій; спроєктовано структуру бази даних
3	Програмна реалізація	Реалізовано базову систему рекомендацій з content-based та reinforcement learning підходами; створено вебінтерфейс з Preline UI; реалізовано генерацію питань, обробку відповідей та систему персоналізованих рекомендацій
4	Тестування програмного забезпечення	Проведено модульне тестування логіки рекомендацій; протестовано відповідність рекомендацій інтересам користувачів; сформовано попередні висновки та визначено напрямки покращення системи

Рисунок Б.15 – Слайд 15

Рисунок Б.16 – Слайд 16

ДОДАТОК В

Графічні матеріали

ДІАГРАМА ПОТОКІВ ДАНИХ КОНТЕКСТНОГО ТА ПЕРШОГО РІВНЯ ДЕКОМПОЗИЦІЇ

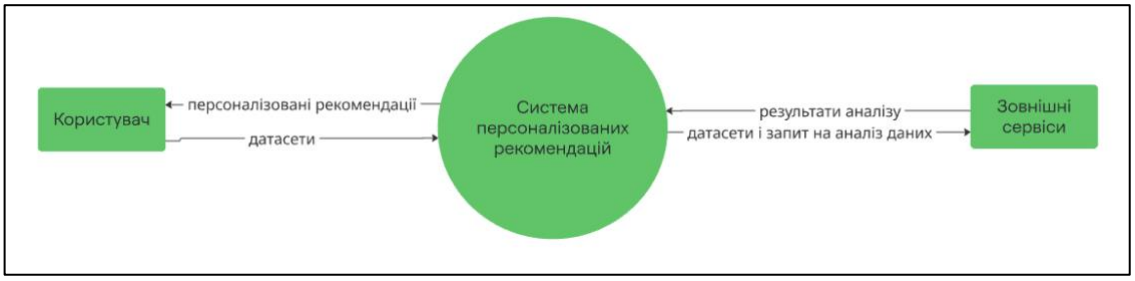

Рисунок В.1 – Діаграма варіантів використання контекстного рівня

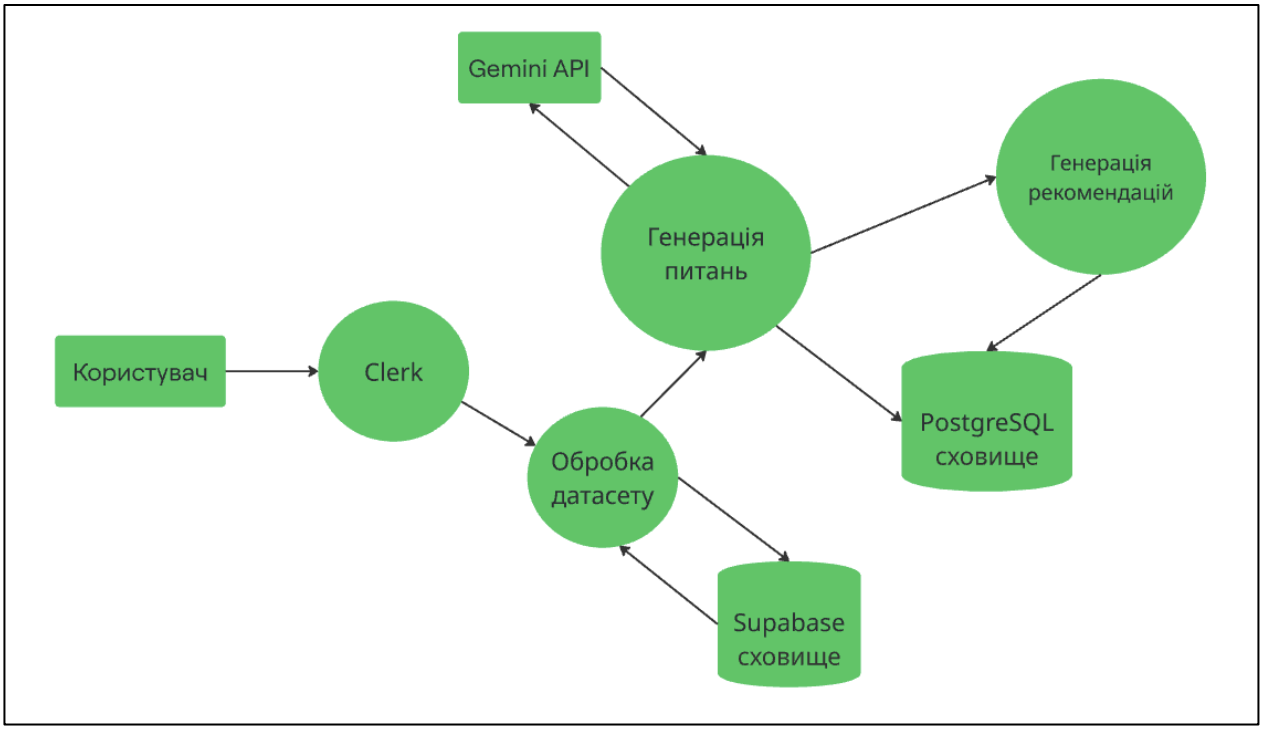

Рисунок В.2 – Діаграма варіантів використання першого рівня

АРХІТЕКТУРА СИСТЕМИ

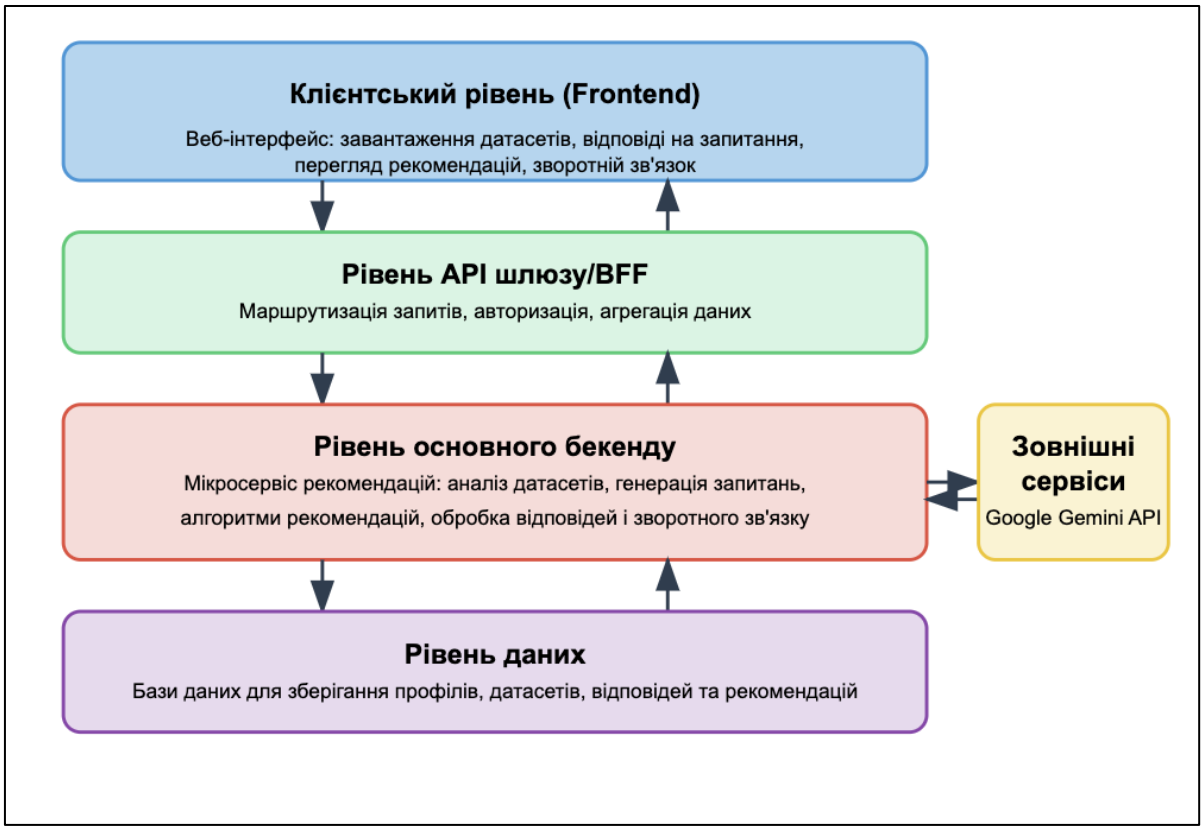

Рисунок В.3 – Архітектура системи

ДІАГРАМА СТАНІВ

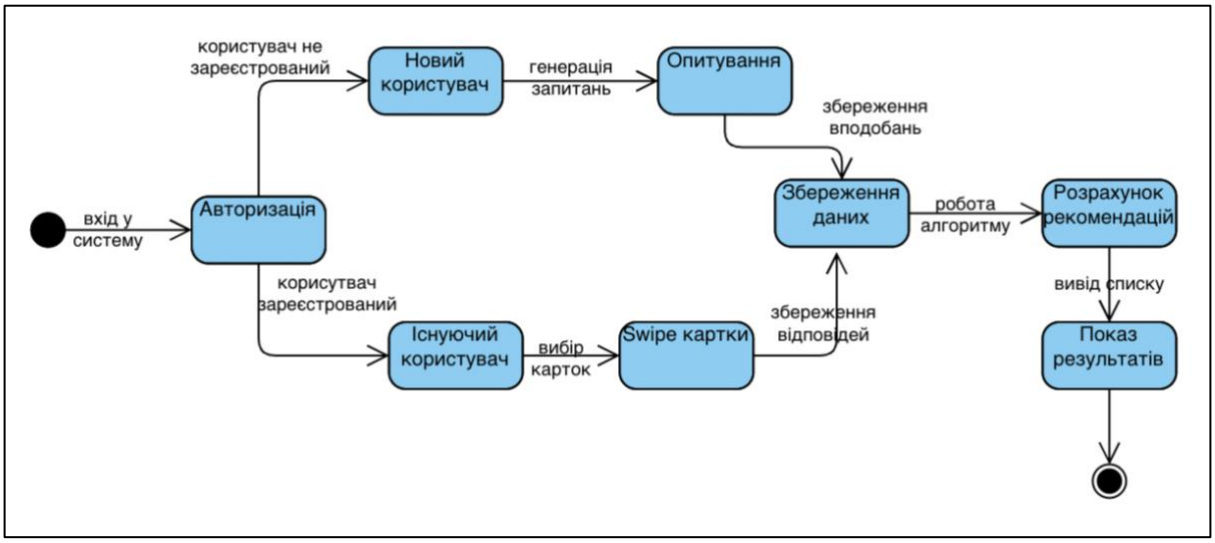

Рисунок В.4 – Діаграма станів

ER ДІАГРАМА

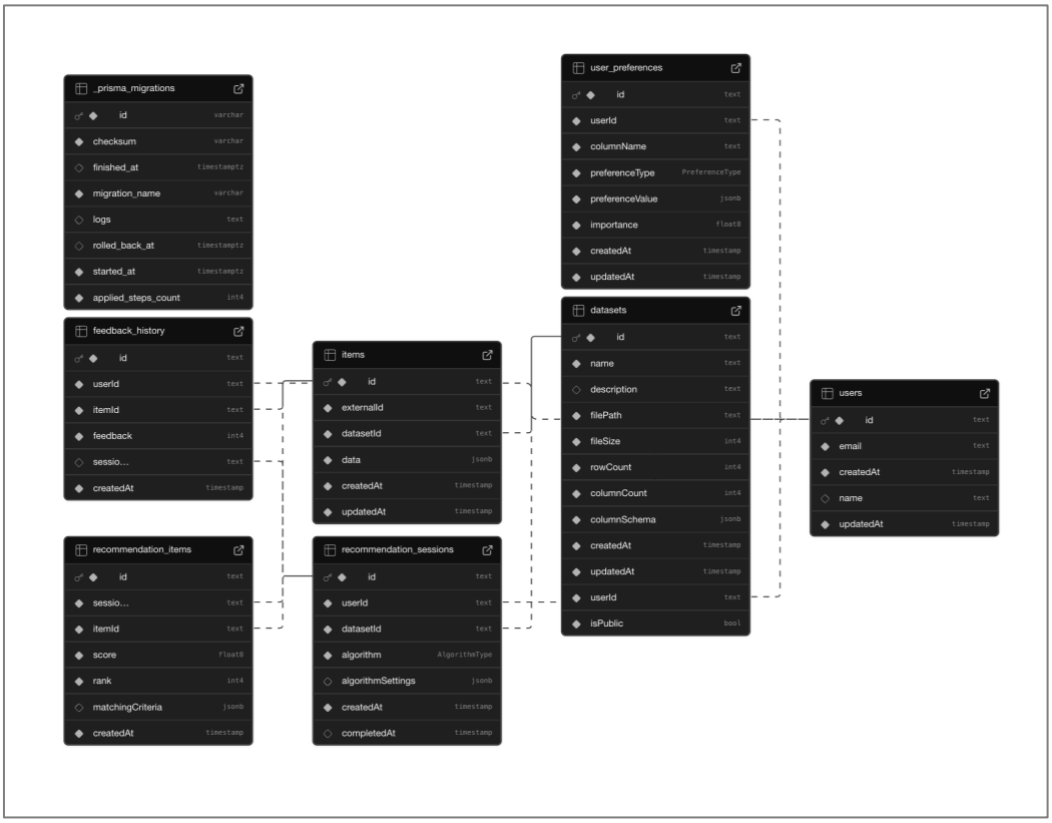

Рисунок В.5 – ER-діаграма

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗ-21-1
Чернушина С. О.
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»: Програмна система персоналізованих рекомендацій на основі аналізу користувацьких даних.

(керівник роботи – Бедратюк Ганна Іванівна)
Прізвище, ім'я, по батькові

2.01.2025
Дата

Підпис студента

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Чернушина Софія Олександрівна,

студентка IV курсу спеціальності 121 – Інженерія програмного забезпечення, група ІПЗ-21-1

здобувач вищої освіти (шифр та назва спец-ті, курс, академічна група)

підтверджую, що ознайомившись з Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті та Кодексом академічної доброчесності і **зобов'язуюсь** дотримуватися їх вимог під час освітнього процесу, проведення наукової діяльності, виконання організаційно-адміністративних функцій тощо.

Усвідомлюю, що у разі порушення мною принципів академічної доброчесності нестиму відповідальність перед академічною спільнотою ХНУ згідно з нормами, визначеними Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, законодавства України.

2 січня 2025 р.

Підпис

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Чернушиної Софії Олександрівни
факультет ІТ, IVкурс, група ІПЗ-21-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлена. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщена та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

3.05.2025
дата

підпис

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 3.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 15%**

	Document		Sum coincidence on the DB	
ID: 242709 Title: БКР_Програмна система персоналізованих рекомендацій на основі аналізу користувацьких даних Added in a DB: 2025-06-02 Authors: Чернушина Софія Heads: Бедратюк Ганна Іванівна Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	83496	1235	4034 (5%)	51 (4%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Чернушина Софія Співавтор:

Назва: БКР_Програмна система персоналізованих рекомендацій на основі аналізу користувацьких даних

Науковий керівник:
Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:3.1%
Коефіцієнт подібності 2:0.3%

Мікропробіли: 0
Заміна букв: 0
Інтервали: 0
Білі знаки: 0

Дата створення звіту: 2025-06-01 16:03:27.0

Після аналізу Звіту подібності констатую наступне:

☐ Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.

☐ Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.

☐ Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.

Обгрунтування:

Дата експерт

01.06.2025 

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Чернушина Софія Олександрівна

Тема Програмна система персоналізованих рекомендацій на основі аналізу користувацьких даних.

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень _____; кількість сторінок записки _____

1.Короткий зміст пояснювальної записки та прийнятих рішень Вступ окреслює актуальність створення динамічної системи персоналізованих рекомендацій у контексті зростаючого попиту на штучний інтелект та проблеми інформаційного перевантаження. Метою роботи є розробка системи «Skunk», яка адаптується під користувачів, аналізує їх поведінку та надає персоналізовані рекомендації в реальному часі. Для досягнення мети поставлено завдання: аналіз предметної області, дослідження методів аналізу даних, вибір архітектури та технологій, проєктування системи, реалізація алгоритмів рекомендацій, розробка інтерфейсу та тестування. Предметом дослідження є програмні засоби та алгоритми для побудови динамічної системи рекомендацій, а об’єктом — процес формування персоналізованих рекомендацій на основі аналізу даних користувачів. Практична значущість полягає у можливості застосування системи для покращення персоналізації в різних онлайн-сервісах.

2. Висновок про відповідність роботи поставленому завданню Завдання, сформульовані у вступі, включають комплексний аналіз предметної області, дослідження методів аналізу даних, обґрунтування архітектури та технологій, проєктування системи, реалізацію алгоритмів, розробку інтерфейсу та тестування. У висновках (текст висновків не надано, тому оцінка базується на аналізі основної частини) результати демонструють реалізацію системи «Skunk» з гібридною багаторівневою архітектурою, модулем обробки даних, інтеграцією з Google Gemini API, базою даних PostgreSQL, а також розробкою інтерфейсу для взаємодії користувачів. Тестування проведено з використанням бібліотек Jest і Cypress. Враховуючи це, можна констатувати, що отримані результати в цілому відповідають поставленим завданням, проте деякі аспекти, зокрема деталізація технічних рішень і тестування продуктивності, розкриті недостатньо.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи Розділ 1 систематизує предметну область рекомендаційних систем, інтегрує приклади популярних платформ і методів фільтрації, а також формулює функціональні та нефункціональні вимоги до системи «Skunk». Позитивно, що розділ ілюструє важливість рекомендаційних систем і демонструє розуміння потреб користувачів. Водночас, опис предметної області є дещо поверхневим, недостатньо розкриті технічні деталі алгоритмів, а вимоги потребують уточнення та деталізації, зокрема щодо форматів даних і обробки помилок.
Розділ 2 проєктує архітектуру системи, обираючи гібридну багаторівневу модель, що забезпечує масштабованість і гнучкість. Позитивним є те, що архітектура узгоджена з особливостями системи та передбачає розподіл на функціональні рівні. Однак, відсутність розгляду альтернативних архітектурних підходів, шаблонів проєктування, детальної декомпозиції модулів, опису взаємодії компонентів, а також відсутність обґрунтування вибору технологій і цільових платформ суттєво обмежує повноту розділу.

Розділ 3 реалізує основні компоненти системи, включаючи модулі обробки даних, інтеграцію з API, базу даних та інтерфейс користувача. Позитивно, що реалізація відповідає архітектурі, описано обробку запитів і відповіді, а також проведено тестування з використанням сучасних бібліотек. Водночас, опис реалізації є загальним, відсутні деталі про управління подіями, фізику (що логічно для неігрової системи), а також відсутні системні вимоги, деталі середовища розробки, тестування продуктивності та візуалізація результатів тестування.

4. Позитивні сторони роботи Робота демонструє глибоке розуміння актуальності проблеми персоналізованих рекомендацій у сучасному цифровому середовищі та інтегрує приклади популярних платформ, що робить тему зрозумілою і релевантною. Архітектурне рішення гібридної багаторівневої системи є технічно обґрунтованим і забезпечує масштабованість та гнучкість. Реалізація модулів відповідає проєктним рішенням, а використання сучасних технологій, таких як FastAPI, PostgreSQL, Next.js, а також бібліотек для тестування, свідчить про практично орієнтований підхід. Інтерфейс користувача розроблено з урахуванням зручності, що підвищує користувацький досвід.

5. Негативні сторони роботи Робота має обмежену деталізацію технічних аспектів, зокрема алгоритмів машинного навчання та методів обробки даних, що знижує її методологічну повноту. Вимоги до програмного забезпечення сформульовані не завжди чітко, відсутні конкретні формати даних і механізми обробки помилок. Архітектурний розділ не охоплює альтернативні підходи та шаблони проєктування, а також не містить обґрунтування вибору технологій і цільових платформ. У реалізації бракує детального опису управління подіями, тестування продуктивності та візуалізації результатів тестування. Відсутні системні вимоги та опис середовища розробки, що ускладнює оцінку готовності продукту до впровадження.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота присвячена актуальній та практично значущій темі створення динамічної системи персоналізованих рекомендацій із застосуванням сучасних методів штучного інтелекту. Зміст роботи відповідає темі та поставленим завданням, демонструє розуміння предметної області і практичну реалізацію системи «Skunk». Проте, робота має обмежену глибину технічного аналізу, недостатньо деталізовані вимоги та архітектурні рішення, а також неповне тестування, що знижує її методологічну та технічну повноту. Програмний продукт є функціональним, реалізований із застосуванням сучасних технологій, але потребує доопрацювання в частині документації, тестування продуктивності та деталізації технічних аспектів. Загалом, робота демонструє добрий рівень самостійності та аналітичного мислення здобувача.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана в повному обсязі, відповідає основним методичним вимогам, має актуальну тему та практичну значущість. Пояснювальна записка структурована, містить обґрунтовані висновки, проте має недоліки в деталізації технічних аспектів і тестуванні. Розроблений програмний продукт функціональний, реалізований із застосуванням сучасних технологій, але потребує доопрацювання в частині документації та тестування продуктивності. Здобувач демонструє достатній рівень володіння матеріалом і аналітичного мислення, хоча деякі розділи розкриті поверхнево. Враховуючи вищезазначене, робота заслуговує оцінки «добре».

РЕЦЕНЗЕНТ Бобилєва Ольга Олександрівна, д.т.н., доцент, зав. каф. КІІС

"02" 06 2025 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ КАФЕДРИ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи «Програмна система персоналізованих рекомендацій на основі аналізу користувацьких даних»
Автор Чернушина Софія Олександрівна
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»
Рівень вищої освіти Бакалавр
Спеціальність 121 – Інженерія програмного забезпечення
Науковий керівник: Бедратюк Ганна Іванівна, старший викладач

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих методів та технологій, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;

2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

3) окремі виявлені збіги є загальновживаними фразами або виразами;

4) система фіксувала технічні особливості (наприклад, поєднання латиниці й українських індексів), а не модифікацію тексту.

Сумарний обсяг запозичень — 3.4%, що відповідає науковим стандартам і не впливає на якість кваліфікаційної роботи.

Дата 1.06.25

Завідувач кафедри _____ Підпис Ганна Бедратюк Ім'я, ПРІЗВИЩЕ

Гарант освітньої програми _____ Підпис Ганна Бедратюк Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Підпис Ганна Бедратюк Ім'я, ПРІЗВИЩЕ