Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Система персоналізованих рекомендацій для користувачів навчальної онлайн-платформи

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

КвРІПЗ.2101083.01.11.ПЗ

Виконав студент IV курсу, група ІПЗ-21-1 _____ Ілля ПЕРВАК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. техн. наук, доцент _____ Юрій ФОРКУН
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. техн. наук, доцент _____ Оксана ЯШИНА
Підпис Ім'я, ПРІЗВИЩЕ

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

6 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
02 01 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Перваку Іллі Юрійовичу
Прізвище, ім'я, по батькові студента

1. Тема роботи Система персоналізованих рекомендацій для користувачів навчальної онлайн-платформи

Керівник роботи Форкун Юрій Вікторович, канд. техн. наук, доцент
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 02.06.2025 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області, Проєктування вебзастосунку, Програмна реалізація та тестування вебзастосунку
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Три креслення:
1. Діаграма варіантів використання
2. Структурна діаграма
3. Діаграма переходів

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Праворська Н.І., доцент кафедри ІПЗ	5.05.25	8.05.25
Антиплагіат	Форкун Ю. В., доцент	5.05.25	8.05.25

7. Дата видачі завдання «_02_» січня_ 2025 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапу виконання	Матеріали завершення етапу	Дата початку-завершення етапу
1 Ознайомлення з тематикою дипломного проєктування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	Індивідуальні теми кваліфікаційних робіт	01.12 – 31.12.2023
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	РОЗДІЛ 1 Пояснювальної записки: Дослідження предметної області та постановка задачі. Технічне завдання на розробку ПЗ	01.01 – 20.02.2024
3 Проєктування програмного забезпечення	РОЗДІЛ 2 Пояснювальної записки: Проєктування ПЗ. Детальний проєкт ПЗ	21.02 – 20.03.2024
4 Програма реалізації з використанням відповідних засобів розробки. Тестування ПЗ	РОЗДІЛ 3 Пояснювальної записки: Програма реалізації та тестування програмного забезпечення. Розроблений програмний засіб	21.03 – 30.04.2024
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	Повний текст пояснювальної записки (включаючи анотацію, зміст, вступ, основні розділи, висновки, перелік джерел посилання, додатки)	01.05 – 25.05.2024
6 Попередній захист КвР	Доповідь, презентація	травень 2024
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки	Супровідні документи. Зброшурована пояснювальна записка	26.05 – 30.05.2024
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозиторії ХНУ; підготовка до захисту та захист КвР	Зброшурована пояснювальна записка з усіма підписами та супровідними документами (здається на кафедру). Підготовлений для репозиторію pdf-файл (надсилається керівнику КвР). Тези доповіді та ілюстративні матеріали (презентація). Публічний захист КвР.	з 01.06.2024

Студент _____ Ілля Первак
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Юрій ФОРКУН
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи «Система персоналізованих рекомендацій для користувачів навчальної онлайн-платформи».

Автор роботи: Первак Ілля Юрійович.

Керівник роботи: Форкун Юрій Вікторович.

Пояснювальна записка: 82 с., 19 рис., 18 табл., 3 дод., 40 джерел.

Графічна частина: 3 креслення ф. А3.

БАЗА ДАНИХ, РЕКОМЕНДАЦІЙНА СИСТЕМА, НАВЧАЛЬНА ОНЛАЙН ПЛАТФОРМА, PYTHON, CSV, SBERT.

Мета кваліфікаційної роботи: розроблення та перевірка системи персоналізованих рекомендацій для онлайн-платформи.

У кваліфікаційній роботі проведено аналіз предметної області та її інформаційного забезпечення, визначені функціональні та нефункціональні вимоги до програмної системи, розроблена загальна архітектура застосунку, спроєктована база даних системи рекомендацій та структура застосунку.

Для реалізації програмного продукту використано мови програмування Python, а також інструменти для обробки даних у форматі CSV та використання моделі SBERT для рекомендацій. За допомогою цих засобів розроблено програмне забезпечення для автоматизації процесу рекомендацій на основі користувацьких даних.

Практичне значення результатів роботи полягає в тому, що впровадження розробленого застосунку дозволяє автоматизувати процес персоналізації контенту на онлайн-платформі та значно підвищити ефективність взаємодії користувачів з платформою.

01.06.2025
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ. 2101083.01.11.ПЗ	Пояснювальна записка	82		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
			Графічні документи			
4	А3	КвРІПЗ. 2101083.01.11.Е8	UML-діаграма варіантів	1		
			використання			
5	А3	КвРІПЗ. 2101083.01.11.Е8	Структурна діаграма	1		
6	А3	КвРІПЗ. 2101083.01.11.Е8	Діаграма переходів	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.2101083.01.11.ПЗ		
Виконав		Первак І. Ю.		01.06	Система персоналізованих рекомендацій для користувачів навчальної онлайн-платформи	Літ.	Арк. 1 / Аркушів 1
Керівник		Форкун Ю. В.		01.06			
Н. контр.		Яшина О. М.		01.06		ХНУ, ІПЗ-21-1	
Зав. каф.		Бедратюк Л.П.		01.06			

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API	–	Application Programming Interface
БД	–	база даних
CI/CD	–	Continuous Integration / Continuous Delivery
GDPR	–	General Data Protection Regulation
JWT	–	JSON Web Token
LMS	–	Learning Management System
ML	–	Machine Learning
SaaS	–	Software as a Service
SDK	–	Software Development Kit
UI	–	User Interface
UTM	–	Urchin Tracking Module
UX	–	User Experience

ВСТУП

Стрімкий розвиток онлайн-освіти зумовив вибухове зростання обсягу доступних курсів, що перетворює пошук релевантних матеріалів на складне й часом хаотичне завдання. За даними останніх досліджень, понад 60 % зареєстрованих слухачів не завершують навіть одного курсу, головним чином через відсутність персоналізованого супровод. Інтелектуальні рекомендаційні системи покликані мінімізувати інформаційне перевантаження, підвищити мотивацію та ефективність навчання, адаптуючи контент до індивідуальних освітніх траєкторій користувачів.

Об’єкт дослідження – процес надання та споживання навчального контенту в онлайн-освітніх платформах.

Предмет дослідження – алгоритми та програмно-технічні засоби побудови систем персоналізованих рекомендацій навчального контенту на основі поведінкових даних користувачів.

Мета роботи – розробити та експериментально перевірити систему персоналізованих рекомендацій для онлайн-платформи, що підвищує релевантність курсів і сприяє зростанню показника їх завершення.

Завдання дослідження:

– проаналізувати предметну область онлайн-освіти та визначити основні проблеми добору навчального контенту;

– дослідити сучасні підходи до персоналізації (колаборативну, контентну й гібридну фільтрацію) та оцінити їхню придатність для освітніх систем;

– виконати порівняльний аналіз наявних програмно-технічних рішень, виявити їх переваги та недоліки;

– сформулювати набір функціональних та нефункціональних вимоги до системи рекомендацій;

– виконати проєктування архітектури програмного забезпечення, структури даних та REST API;

– реалізувати прототип системи із застосуванням FastAPI, MongoDB та бібліотеки LightFM;

– провести експериментальне тестування (Precision@K, Recall, MAP) та порівняти результати з базовими методами;

– підготувати практичні рекомендації щодо інтеграції розробленої системи в діючу платформу.

Методологія дослідження поєднує методи системного аналізу, UML-моделювання, алгоритми колаборативної фільтрації, статистичну оцінку точності рекомендацій, а також засоби контейнеризації Docker для забезпечення відтворюваного розгортання.

Новизна полягає в адаптації гібридного підходу до рекомендацій навчального контенту з урахуванням одночасно поведінкових і семантичних факторів, а також у запропонованій процедурі динамічного калібрування моделі під змінні навчальні цілі користувача.

Практичне значення полягає у можливості впровадження розробленої системи на реальних освітніх платформах, що дозволить підвищити коефіцієнт завершення курсів і скоротити час пошуку релевантних матеріалів.

Структура роботи. Пояснювальна записка складається з трьох розділів [1]. У першому розділі проведено дослідження предметної області та сформульовано задачі розроблення. У другому розділі виконано проєктування вебзастосунку й обґрунтовано вибір технологій. У третьому розділі наведено реалізацію системи, результати тестування та оцінювання її ефективності. Роботу доповнюють загальні висновки, перелік використаних джерел та додатки.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Змістовий аналіз предметної області онлайн-освіти

Предметну область дипломної роботи становлять онлайн-освітні платформи – цифрові системи, що забезпечують інтерактивний, структурований доступ до навчального контенту через мережу Інтернет. Їх функціонування базується на гнучкому форматі подання інформації та підтримці індивідуального темпу опанування матеріалу. Ці платформи спрямовані на розширення можливостей користувача щодо вибору тем, тривалості навчання, форми взаємодії та персональних освітніх траєкторій.

З розширенням ринку масових відкритих онлайн-курсів (MOOC) та цифрової трансформації освіти, навчальний контент дедалі частіше виступає не просто ресурсом, а основою для формування індивідуального навчального середовища [2]. З одного боку, це відкриває безпрецедентні можливості для самостійного розвитку, з іншого – створює низку бар'єрів, пов'язаних із надмірною кількістю варіантів. У середовищі, де доступні тисячі курсів, складно обрати ті, що справді відповідають потребам, рівню підготовки та стилю сприйняття інформації конкретного користувача.

Типовою є ситуація, коли студент обирає курс на основі поверхневих критеріїв – назви, рейтингу або кількості переглядів – і вже через кілька занять припиняє навчання через його нерелевантність. Це свідчить про відсутність механізмів індивідуального добору контенту, які здатні враховувати не лише тематику, а й контекст споживання знань. Особливо критичною ця проблема є для нових користувачів, які ще не мають історії взаємодії з платформою.

На початкових етапах взаємодії, коли відсутні поведінкові патерни, системи мають справу з так званим ефектом cold-start. Для його подолання застосовують змішані підходи: коротке стартове анкетування, використання демографічних параметрів, тематичних пріоритетів або перенесення профілю користувача з іншої системи [3]. Такі заходи дозволяють зібрати мінімальну кількість інформації, достатню для побудови первинної моделі рекомендацій.

Функціонально онлайн-платформи дистанційного навчання складаються з кількох рівнів. До базових компонентів належать: модулі аутентифікації, персонального кабінету, доступу до навчальних ресурсів. До підтримувальних – блоки взаємодії (чати, форуми), система оцінювання, збереження прогресу та генерація звітів. Інтелектуальні модулі – це зокрема компоненти аналітики активності, адаптації матеріалу й формування рекомендацій. Архітектурно ці елементи поєднані через API, що забезпечує їхню взаємодію та масштабованість [4].

Ключовими суб'єктами системи виступають:

– слухачі, які потребують доступу до актуального, зрозумілого та структурованого контенту;

– викладачі, що адмініструють курси, відслідковують результати й адаптують навчальний процес;

– адміністратори, відповідальні за системну стабільність, модерацію й технічну підтримку.

Взаємодію приведених основних ролей та напрямки інформаційних потоків наведено на Рисунку 1.1.

Рисунок 1.1 – Контекстна діаграма системи персоналізованих рекомендацій

Інформаційні потреби різних категорій користувачів формуються на основі взаємодії з матеріалами, історії переглядів, виконаних завдань, реакцій на повідомлення системи тощо.

Первинна інформація, яку обробляє система, охоплює такі реквізити:

– події користувача (user_id, session_id, timestamp, тип дії);

– параметри курсу (course_id, категорія, рівень складності, формат);

– контекст використання (час доби, пристрій, тривалість сесії);

– прогрес користувача (відсоток завершення, середній бал, паузи).

Типова схема обробки включає такі етапи:

– збір даних із журналу подій;

– очищення від шумів та помилкових дублікатів;

– злиття з даними про користувача й курс;

– агрегація активностей у профіль взаємодії;

– класифікація й подальше формування персоналізованих траєкторій.

Результатом цього процесу є побудова рекомендаційного блоку: список курсів, що найбільш ймовірно зацікавлять користувача, відновлення незавершеного навчання, пропозиції на основі подібної поведінки інших учнів. Додатково система може показувати аналітику прогресу, запропонувати відповідний модуль або повторно активувати матеріал.

З точки зору захисту даних, система повинна дотримуватись принципів приватності та керування доступом. Для цього застосовуються стандарти авторизації (OAuth 2.0), токенізація сесій (JWT), чітке розмежування прав доступу відповідно до ролей, можливість перегляду або видалення історії. Усі дії з персональними даними мають відповідати міжнародним стандартам безпеки – зокрема, вимогам GDPR та ISO/IEC 27001.

Таким чином, предметна область онлайн-освіти поєднує високий рівень функціональної складності з великою кількістю індивідуалізованих сценаріїв взаємодії. Основною проблемою, на яку спрямовано розв'язання у межах даної роботи, є брак ефективного механізму добору навчального контенту з урахуванням поведінкових, змістових та контекстних факторів [5]. Це обґрунтовує доцільність створення спеціалізованої системи персоналізованих рекомендацій, яка дозволить підвищити результативність навчання, зменшити кількість відмов і сприятиме формуванню сталого освітнього інтересу.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Проблематика добору релевантного навчального контенту в умовах перенасичення інформацією зумовила стрімкий розвиток рекомендаційних технологій, які стали важливою складовою сучасних освітніх платформ [6]. На відміну від традиційних методів навігації – таких як ручна категоризація, тегування або пошук за ключовими словами – системи персоналізації надають можливість формувати динамічні, адаптивні пропозиції, що відповідають індивідуальному профілю слухача. Вони враховують не лише тематику, а й поведінку, послідовність дій, емоційне залучення, успішність виконання завдань, рівень залученості та інші чинники.

Найпоширеніші типи рекомендаційних систем, їх прикладні реалізації та сфери застосування в контексті цифрової освіти становлять основу для визначення рівня технологічної готовності методів персоналізації. У даному контексті важливим є не лише огляд теоретичних підходів, а й аналіз практичних прикладів, що демонструють потенціал до адаптації у межах вітчизняного освітнього середовища. Особливу увагу було приділено сильним і слабким сторонам кожного з розглянутих підходів, з метою формування обґрунтованих технічних рішень на наступних етапах проєктування.

Рекомендаційні системи зазвичай класифікуються на три основні підходи: колаборативна фільтрація, контент-орієнтована фільтрація та гібридні моделі. Кожен з них має власні теоретичні засади, алгоритмічну реалізацію, вимоги до даних і типові сценарії застосування.

Колаборативна фільтрація (Collaborative Filtering) – це метод, що передбачає виявлення схожих користувачів або об’єктів на основі історії їхньої взаємодії [7]. В її основі лежить припущення, що якщо два користувачі оцінювали однакові курси однаково, то вони, ймовірно, матимуть схожі уподобання і в майбутньому. Це дозволяє формувати рекомендації навіть тоді, коли система нічого не знає про новий контент, якщо в наявності достатній обсяг взаємодій. Алгоритми реалізації включають ALS (Alternating Least Squares), SVD (Singular

Value Decomposition), матричну факторизацію, алгоритми кластеризації, а також моделі на основі нейромереж (Neural Collaborative Filtering). Однією з головних переваг цього підходу є його незалежність від атрибутів об'єкта: система не вимагає семантичної розмітки курсів, достатньо лише інформації про взаємодії. Серед слабких місць – ефект cold-start (проблеми з новими користувачами або новими курсами) і розрідженість матриці вподобань.

Хрестоматійним прикладом масштабного впровадження колаборативної фільтрації є Netflix. Незважаючи на те, що ця платформа орієнтована на розважальний контент, її технології персоналізації стали основою для багатьох освітніх сервісів. Зокрема, принципи аналізу спільних уподобань, розбиття за патернами споживання та рекомендації на основі поведінки інших користувачів активно застосовуються в сучасних навчальних середовищах. Наприклад, FutureLearn (FutureLearn Ltd., Велика Британія) або Open edX (edX / 2U Inc., США) використовують аналогічні механізми для формування індивідуалізованих навчальних маршрутів, підлаштованих під стиль, темп і попередній досвід учнів.

Контент-орієнтована фільтрація (Content-Based Filtering) фокусується на властивостях самих об'єктів і зіставленні їх із профілем користувача [8]. Якщо слухач переглянув курс з машинного навчання, система запропонує йому інші курси зі схожими характеристиками – тематика, рівень складності, формат, тривалість, мова викладання. Це забезпечує швидкий старт і дозволяє працювати навіть за мінімальної історії взаємодій. Алгоритми, що використовуються в таких системах, включають TF-IDF, cosine similarity, семантичне кодування ознак, використання онтологій, а також методи з використанням Word2Vec, BERT або Doc2Vec для побудови векторних представлень курсів. Контентна модель менш залежна від інших користувачів, але вона часто схильна до замикання в «інформаційній бульбашці», де рекомендації стають одноманітними, та вузькопрофільними і з часом не здатні дивувати користувача чимось новим.

Amazon – одна з перших компаній, яка продемонструвала ефективність контентно-орієнтованої фільтрації в комерційному середовищі. Хоча її початкове застосування зосереджене на підборі товарів, ті самі алгоритми – аналіз

атрибутів, подібність за ключовими характеристиками, контекст переглядів – були адаптовані для освітніх задач. Сучасні навчальні платформи, такі як Udemy (Udemy Inc., США) та Khan Academy (Khan Academy, США), впроваджують подібні підходи: аналізують метадані курсів, рівень складності, мову подачі та інші параметри для формування персоналізованих рекомендацій [9], навіть коли поведінкової історії ще недостатньо.

Гібридні рекомендаційні системи (Hybrid Recommenders) поєднують переваги двох підходів – вони здатні адаптуватися до нових користувачів, зберігаючи точність для сталих сценаріїв [10]. Існує кілька типів гібридизації: каскадна (послідовна), зважена, змішана, мета-рекомендаційна. У таких системах можуть одночасно враховуватись рейтинг курсу, тематика, результати інших студентів з подібним профілем, відгуки, статистика завершення та навіть часові інтервали між сесіями навчання. Залежно від конкретного підходу, система може змінювати порядок застосування алгоритмів або комбінувати результати з різною вагою. Наприклад, у каскадних моделях контентна фільтрація спрацьовує першою, а колаборативна – уточнює остаточну вибірку.

Перевага гібридних моделей полягає у здатності враховувати як індивідуальні вподобання користувача, так і агреговану поведінку схожих слухачів. Наприклад, система може враховувати траєкторії користувачів із подібним навчальним профілем, комбінуючи ці дані з пріоритетними темами, позначеними самим слухачем. У реальних реалізаціях гібридні моделі також враховують поведінкову реакцію на попередні рекомендації: система знижує релевантність контенту, що регулярно ігнорується, динамічно адаптуючи видачу до змін у поведінці користувача.

Ці якості зробили гібридні підходи одним із найбільш придатних рішень для впровадження в освітніх платформах. Наприклад, Coursera (Coursera Inc., США) реалізує комбіновану модель: на головному екрані користувач бачить рекомендовані курси на основі раніше завершених тем, навчального плану або професійної мети, зазначеної при реєстрації. Інтерфейс містить навігаційний блок «Recommended for you», який динамічно змінюється залежно від прогресу,

активності та історії взаємодій. Блоки рекомендацій розміщені у центральній частині сторінки, супроводжуються коротким описом, рейтингом, тривалістю та кнопкою швидкого доступу до курсу. Крім того, на бічній панелі можуть відображатись картки із незавершеними курсами, повторними пропозиціями або тематичними добірками (див. Рисунок 1.2).

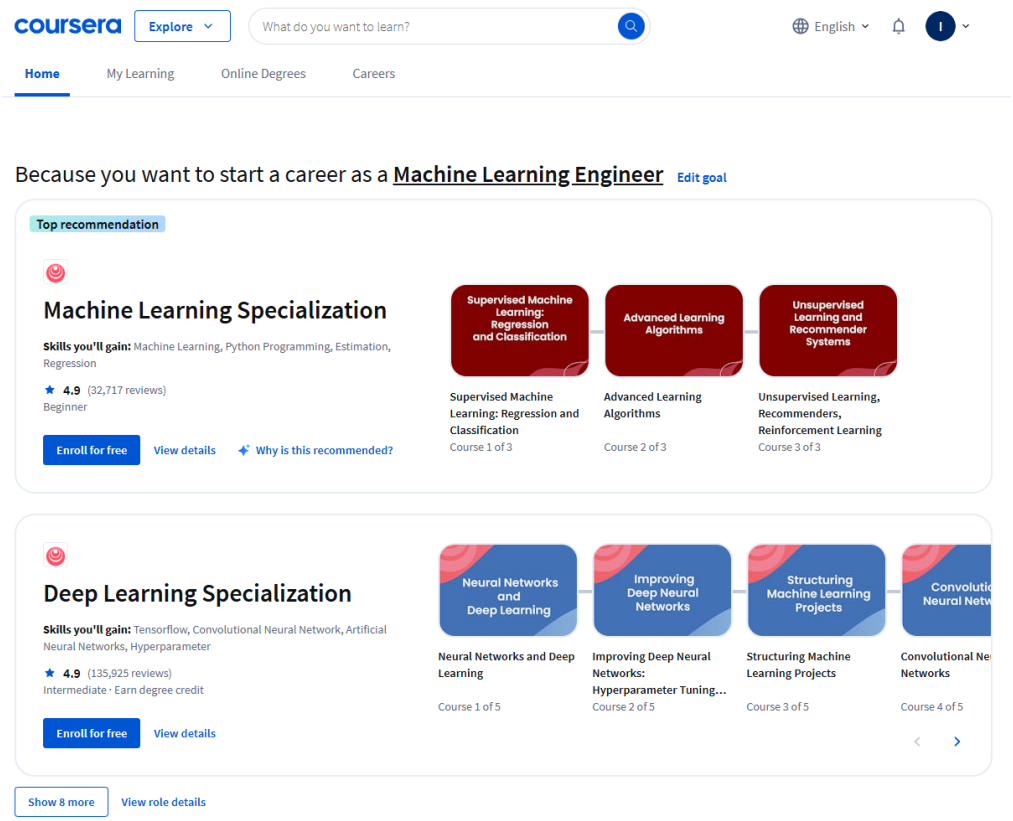


Рисунок 1.2 – Інтерфейс рекомендаційного блоку на платформі Coursera

Другий приклад – платформа Edmodo Edmodo (NetDragon Websoft Inc., Китай) реалізує адаптивні сценарії як для викладачів, які отримують добірки інструментів оцінювання та завдань, так і для учнів, яким система пропонує навчальні модулі відповідно до складності, пройдених тем та рівня успішності.

Окрему групу становлять освітні платформи, які не є прикладами реалізації конкретного типу фільтрації, але демонструють гнучкі підходи до інтеграції персоналізації через відкриті механізми. Зокрема, Moodle (Moodle HQ, Австралія) забезпечує реалізацію як колаборативних, так і контентних сценаріїв через систему плагінів [11]. Stepik використовує внутрішню аналітику прогресу

для адаптації матеріалів. LinkedIn Learning (Microsoft Corp., США) поєднує навчальні рекомендації з професійним контекстом, орієнтуючись на побудову персоналізованих траєкторій з урахуванням кар'єрних цілей користувача.

Порівняльний аналіз ключових підходів до персоналізації та систем, у яких вони реалізовані, подано у табличній формі (див. Таблицю 1.1) з метою наочного зіставлення їх основних характеристик.

Таблиця 1.1 – Порівняльна характеристика підходів до персоналізації та прикладних систем

Назва системи	Тип реалізованого підходу	Механізми персоналізації	Переваги	Недоліки
Coursera	Гібридний	Контентна + колаборативна фільтрація, евристичні правила	Висока адаптивність до траєкторії користувача, персональний feed	Деякі курси платні, рекомендації не завжди враховують локальний контекст
Udemy	Контентно-орієнтований	Категоризація інтересів, фільтрація за метаданими	Великий вибір курсів, швидке формування рекомендацій	Якість контенту нерівномірна через відкриту модель
Future Learn	Колаборативна / соціальна	Кластеризація користувачів, спільне навчання	Соціальна взаємодія, мікро-кваліфікації	Менший асортимент курсів порівняно з глобальними платформами
Edmodo	Гібридний	Поведінкова аналітика, адаптація до рівня	Рекомендації для учнів і викладачів, адаптація до рівня	Менше академічного контенту, фокус на шкільному сегменті
Khan Academy	Контентно-орієнтований	Метадані, тематичні мітки, прогрес-орієнтована адаптація	Безкоштовна модель та простий інтерфейс	Обмежений набір напрямів і мов

Продовження Таблиці 1.1

Назва системи	Тип реалізованого підходу	Механізми персоналізації	Переваги	Недоліки
Stepik	Гібридний	Аналітика прогресу, часткова персоналізація за активністю	Підтримка відкритих курсів, аналітика завдань	Персоналізація вбудована частково, не основна функція
Moodle	Модульна (залежить від плагінів)	Плагіни з підтримкою CF, CBF, рекомендацій на основі ролей	Гнучкість, open-source, локалізованість	Якість рекомендацій залежить від налаштувань
LinkedIn Learning	Гібридний	Алгоритмічний профіль, адаптація до кар'єрних цілей	Інтеграція з LinkedIn, професійна орієнтація	Орієнтація на професійний ринок, не загальноосвітній сегмент

На основі зіставлення вищезазначених систем було виділено низку ключових критеріїв, що мають вирішальне значення для якості рекомендацій у навчальному контексті [12]:

– здатність до адаптації до змін цілей користувача;

– стійкість до неповних даних та холодного старту;

– інтерпретованість і прозорість логіки рекомендацій;

– підтримка зворотного зв'язку та інтеграція результатів взаємодії;

– врахування часових і контекстуальних характеристик користувача.

Сильними сторонами колаборативної фільтрації є висока точність і гнучкість при достатній кількості даних, але вона складно працює в нових умовах. Контентна фільтрація чудово працює для новачків, але замикає в обмеженому наборі варіантів. Гібридні моделі надають баланс, проте вимагають складного налаштування й великих обчислювальних ресурсів. Загалом, огляд наявного програмного забезпечення у сфері освітніх рекомендацій показав, що

оптимальною є орієнтація на комбіновані підходи з можливістю подальшого розширення функціоналу. Водночас, для україномовного або регіонально орієнтованого контенту необхідно враховувати такі фактори, як мова, формат матеріалу, локальні освітні потреби [13] та відповідність нормативним вимогам щодо обробки персональних даних.

Усі наведені висновки та результати аналізу лягли в основу формування вимог до програмного забезпечення, а також слугували обґрунтуванням вибору архітектурного та алгоритмічного рішення для побудови системи персоналізованих освітніх рекомендацій.

1.3 Визначення вимог до програмного забезпечення

На основі аналізу предметної області, класифікації ключових типів користувачів онлайн-освітніх систем та дослідження можливостей і обмежень сучасних аналогів, було сформовано систематизований перелік вимог до цільового вебзастосунку – системи персоналізованих рекомендацій, що має стати функціональним розширенням для існуючої онлайн-платформи навчального контенту. Основною метою даного модуля є забезпечення максимально релевантного добору навчальних матеріалів відповідно до індивідуальних характеристик, вподобань і активностей кожного користувача.

Формування вимог здійснювалось із урахуванням стандартного поділу на дві великі категорії: функціональні (тобто ті, що визначають очікувану поведінку ПЗ у відповідь на зовнішні дії) та нефункціональні (пов’язані з технічними, організаційними та експлуатаційними характеристиками) [14]. Щоб уніфікувати представлення та полегшити навігацію, в тексті використано маркування: F – функціональні вимоги, N – нефункціональні вимоги, T – задачі, які передбачено реалізувати в рамках проєкту. Такий підхід дозволяє здійснити подальше трасування вимог до задач та оцінити ступінь покриття функціоналу.

Функціональні вимоги стосувалися безпосередньо сценаріїв користувацької взаємодії, а саме було передбачено: створення особистого

профілю, визначення освітніх цілей, налаштування параметрів персоналізації, доступ до навчального каталогу, проходження курсів, взаємодію з інтерфейсом та оцінку рекомендацій. Особливу увагу було приділено збору поведінкових даних: до них належать як звичні події (перегляди, вибір теми, оцінки), так і контекстуальні змінні (час доби, тривалість сесії, повторюваність дій, темп проходження тощо) [15]. Система мала враховувати не лише наявність подій, а й їхню частоту, щільність, послідовність та якісні характеристики, що формували повноцінний поведінковий профіль користувача [16]. Було передбачено також реалізацію механізмів зворотного зв'язку, які давали змогу системі адаптуватися під дії користувача – наприклад, через позначення курсів як релевантних або ні, редагування тематичних уподобань, тимчасове або повне вимкнення рекомендаційного модуля. Такі дані використовувались для побудови персоналізованого блоку освітнього контенту, адаптованого під поточні та довгострокові інтереси користувача. Деталізований перелік функціональних вимог наведено в Таблиці 1.2.

Таблиця 1.2 – Функціональні вимоги

Назва вимоги	Категорія	Опис
F1: Формування персоналізованих рекомендацій	Логіка системи	Система формує добірку навчальних курсів з урахуванням активності, уподобань і попереднього досвіду користувача.
F2: Редагування інтересів	Користувач	Користувач має змогу оновлювати свої освітні вподобання або вимикати персоналізацію.
F3: Налаштування профілю	Користувач	У профілі користувача відображаються цілі навчання, сфери зацікавлення та обрані напрями.
F4: Оцінювання релевантності	Користувач	Система враховує відгуки користувача щодо точності та корисності рекомендованого контенту.

Продовження Таблиці 1.2

Назва вимоги	Категорія	Опис
F5: Керування системними параметрами	Адміністратор	Адміністратор може змінювати правила персоналізації, керувати категоріями курсів та переглядати статистику взаємодії.
F6: Користувацький контроль над даними	Користувач	Користувач має можливість переглядати, редагувати або видаляти свої персональні дані, включно з історією активності.
F7: Адаптивність інтерфейсу	UI	Інтерфейс автоматично адаптується до форм-фактору пристрою та рівня технічної підготовки користувача.
F8: Відновлення навчального процесу	Користувач	Система надає можливість продовжити проходження курсу з того місця, де воно було зупинено.

Окрему увагу було приділено вимогам до інтерфейсу користувача як ключового елементу взаємодії з системою [17]. Його функціональність охоплювала адаптивність до різних типів пристроїв, логічну та передбачувану структуру, мінімалістичний і доступний дизайн, а також урахування рівня підготовки користувачів. Передбачалося впровадження режимів перегляду (базового та розширеного), а також динамічна адаптація рекомендацій залежно від змін у поведінці користувача. Зокрема, у разі припинення активності в межах певної тематики, алгоритм мав самостійно коригувати видачу без потреби ручного втручання. Ці вимоги доцільно розглядати як перехідний блок між функціональними та нефункціональними характеристиками, оскільки вони одночасно охоплюють і логіку взаємодії, і якість користувацького досвіду.

Нефункціональні вимоги висувають критерії до якості роботи системи загалом. Вони охоплюють: продуктивність (час відповіді, витрати ресурсів), масштабованість (здатність підтримувати збільшення кількості користувачів без

деградації), безпеку (захист даних, шифрування, контроль доступу), надійність (обробка збоїв, резервування), доступність (підтримка 24/7), відповідність нормативно-правовим вимогам (GDPR, OWASP ASVS [18]) та етичним стандартам (прозорість логіки рекомендацій, запобігання дискримінації). Зокрема, кожен користувач повинен мати змогу ознайомитися зі збереженими про нього даними, скоригувати або видалити їх, а також повністю відмовитись від зберігання поведінкового профілю. Узагальнена структуризація нефункціональних вимог представлена в Таблиці 1.3.

Таблиця 1.3 – Параметри нефункціональних обмежень

Назва вимоги	Категорія	Опис
N1: Швидкість взаємодії	Продуктивність	Середній час відповіді інтерфейсу на типові дії не повинен перевищувати 400 мс при стандартному навантаженні.
N2: Масштабованість під навантаженням	Масштабованість	Система повинна підтримувати стабільну роботу при одночасній активності великої кількості користувачів без втрати продуктивності.
N3: Захист від зовнішніх загроз	Безпека	Платформа повинна відповідати сучасним вимогам до веб-безпеки та забезпечувати захист від несанкціонованого доступу.
N4: Конфіденційність персональних даних	Конфіденційність	Обробка персональних даних має відповідати принципам конфіденційності, включаючи право на перегляд, редагування та видалення інформації.

На підставі викладених вимог було виокремлено перелік задач, які відображають основні напрямки реалізації функціональності, структури інтерфейсу та поведінкової логіки системи. Задачі сформовані як безпосередній

наслідок аналізу функціональних і нефункціональних вимог, та згруповані за змістовними напрямами, які охоплюють різні етапи розробки, тестування та впровадження системи [19]. Зведену їх класифікацію наведено в Таблиці 1.4.

Таблиця 1.4 – Узагальнена структура задач реалізації

Позначення	Формулювання задачі
Т1	Розробити гнучкий алгоритм рекомендацій з комбінуванням контентного аналізу та аналізу поведінки користувачів, з урахуванням зміни інтересів з часом.
Т2	Створити модель бази даних для зберігання багатокомпонентного профілю користувача, історії активності, параметрів персоналізації та відгуків.
Т3	Реалізувати програмний інтерфейс (REST API), що забезпечує зв'язок між клієнтською частиною, алгоритмом рекомендацій і системою управління даними.
Т4	Розробити адаптивний веб-інтерфейс, який дозволяє керувати інтересами, переглядати персоналізований контент і взаємодіяти з рекомендаціями.
Т5	Інтегрувати механізми аутентифікації, авторизації та контролю доступу до персональних даних відповідно до принципів конфіденційності.
Т6	Підготувати та впровадити метрики якості рекомендацій, зокрема Precision@K і Recall@K, з подальшим тестуванням і аналізом результатів.

З метою забезпечення цілісності реалізації, сформульовані задачі було співвіднесено з відповідними функціональними та нефункціональними вимогами. Це дозволило побудувати трасувальну матрицю, що демонструє, які саме вимоги враховуються в межах кожної задачі. Такий підхід дав змогу

уникнути дублювань, виявити можливі прогалини й забезпечити узгодженість між задумом і реалізацією (див. Таблицю 1.5).

Таблиця 1.5 – Відповідність вимог етапам реалізації

Вимога	Опис вимоги	Пов’язана задача	Опис задачі
F1	Формування персоналізованих курсів	T1	Реалізація гібридного алгоритму рекомендацій
F2	Редагування тематичних уподобань	T4	Розробка інтерфейсу для налаштування вподобань
F3	Формування навчального профілю	T2	Створення структури бази даних користувачів
F4	Надання зворотного зв’язку	T6	Збір і аналіз оцінок рекомендацій
F5	Адміністративне конфігурування	T5	Налаштування параметрів персоналізації та керування доступом
F6	Керування персональними даними	T5	Захист і редагування персональних даних
F7	Адаптивний інтерфейс	T4	Розробка інтерфейсу з адаптацією до пристрою
F8	Відновлення призупиненого навчання	T2	Реалізація механізму збереження прогресу
N1	Швидкодія інтерфейсу	T3	Забезпечення швидкої відповіді REST API
N2	Масштабованість до великої кількості користувачів	T5	Оптимізація продуктивності при навантаженні
N3	Захист від несанкціонованого доступу	T5	Імплементація засобів контролю доступу
N4	Конфіденційність персональних даних	T5	Забезпечення відповідності принципам конфіденційності

З огляду на сформульовану функціональність, логіка роботи системи тісно пов’язана з типами користувачів, які з нею взаємодіють. Було виокремлено чотири ключові ролі, для кожної з яких визначено базовий набір дій та очікувану поведінку в межах системи (див. Таблицю 1.6). На основі цих моделей побудовано варіанти використання, що відображають типові сценарії взаємодії з платформою (див. Таблицю 1.7).

Таблиця 1.6 – Перелік акторів системи

Актор	Опис ролі
Гість	Відвідувач без авторизації, який має обмежений доступ до відкритого переліку курсів без персоналізації.
Студент	Авторизований користувач, що проходить курси, отримує персональні рекомендації та взаємодіє з системою.
Викладач	Формує навчальний контент, аналізує активність студентів і підтримує актуальність матеріалів.
Адміністратор	Здійснює налаштування правил персоналізації, контролює параметри системи та має доступ до статистичних даних.

Таблиця 1.7 – Характеристика акторів системи рекомендацій

Назва варіанту	Актор	Опис
Перегляд каталогу курсів	Гість	Ознайомлення з доступними курсами без необхідності авторизації.
Авторизація	Гість	Вхід до системи через email або соцмережі для доступу до функцій.
Формування профілю	Студент	Визначення цілей навчання та формування тематичних інтересів.
Відстеження активності	Студент	Збір і фіксація поведінкових даних: перегляди, вибір, оцінки.
Генерація рекомендацій	Студент	Побудова списку курсів на основі поведінки користувача.
Оцінка рекомендацій	Студент	Визначення релевантності курсів через механізм «подобається / ні».

Продовження Таблиці 1.7

Назва варіанту	Актор	Опис
Коригування уподобань	Студент	Зміна тематичних інтересів або вимкнення персоналізації.
Перегляд аналітики	Викладач	Аналіз активності студентів за допомогою аналітичних звітів.
Редагування курсу	Викладач	Оновлення змісту курсу та управління його структурою.
Керування параметрами системи	Адміністратор	Налаштування правил рекомендацій та перегляд системної статистики.

Визначені категорії користувачів і пов’язані з ними сценарії взаємодії системи було узагальнено у графічному вигляді [20]. Такий підхід дав змогу представити типові дії, межі відповідальності та характерні взаємозв’язки між актором і системою. З цією метою було побудовано use-case діаграму (див. Рисунок 1.3), що відображає структуру взаємодії користувачів із платформою відповідно до їхніх ролей [21].

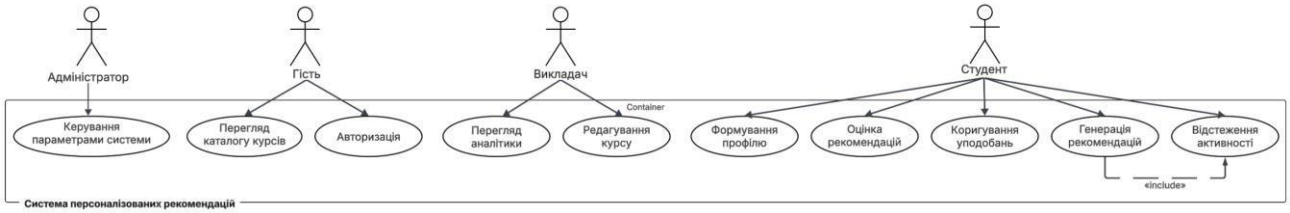


Рисунок 1.3 – Діаграма варіантів використання сиситеми

Для окремих варіантів використання було також сформовано текстові специфікації, що описують їхню структуровану логіку. Як приклад, у сценарії «Генерація персоналізованих рекомендацій» визначено такі елементи:

– актор: студент;

– передумова: користувач авторизований у системі, має збережений профіль із хоча б мінімальною історією дій;

– основний сценарій: користувач переходить на головну сторінку; система ініціює виклик до сховища поведінкових даних, виконує розрахунок рекомендацій на основі гібридної моделі та відображає персоналізований список актуальних курсів;

– альтернативний сценарій: якщо історія дій відсутня, система генерує стартову видачу на основі короткої анкети або рейтингу популярних курсів;

– результат: користувач отримує персоналізовані рекомендації;

– післяумова: факт перегляду фіксується в логах активності для подальшого оновлення профілю користувача.

Таким чином, структуризація вимог, задач, сценаріїв та ролей забезпечила узгоджене бачення функціональності майбутньої системи. Це дозволило досягти цілісного охоплення ключових аспектів взаємодії користувачів із платформою та мінімізувати ризики втрати логіки при реалізації.

1.4 Висновки першого розділу

У межах розділу було сформовано повноцінну систему вимог до майбутнього вебзастосунку, з урахуванням особливостей предметної області, характеру взаємодії користувачів із платформою та сучасних підходів до персоналізації. Узагальнення функціональних та нефункціональних характеристик дозволило задати логічні орієнтири для подальшого проєктування системи [22]. Сформульовані вимоги дали змогу визначити основні задачі розробки, що охоплюють: реалізацію алгоритмів персоналізованих рекомендацій, побудову архітектури з підтримкою масштабованості та безпеки, розробку інтерфейсних компонентів, налаштування сховищ даних користувачів, а також впровадження механізмів контролю якості рекомендацій. Поставлені задачі узгоджуються з очікуваною структурою програмного продукту, відповідають обмеженням та можливостям середовища впровадження і забезпечують логічну наступність для переходу до технічного етапу проєктування системи у розділі 2.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ

2.1 Вибір архітектури та загальна характеристика системи

На основі сформульованих функціональних та нефункціональних вимог, а також із урахуванням специфіки предметної області було обґрунтовано доцільність реалізації системи персоналізованих рекомендацій у вигляді клієнт-серверного вебзастосунку. Такий підхід забезпечує розподіл відповідальностей між інтерфейсною, прикладною та логічною частинами, спрощує масштабування, сприяє повторному використанню модулів і дає змогу реалізувати взаємодію між підсистемами за допомогою REST API.

Архітектура системи побудована за принципом трирівневої моделі [23], яка охоплює такі логічно виокремлені компоненти:

– рівень представлення (клієнтська частина), реалізований із застосуванням React або Streamlit – відповідає за інтерфейс користувача, відображення персоналізованого контенту, збирання зворотного зв'язку та передавання параметрів налаштувань;

– рівень логіки застосунку (серверна частина), побудований на основі FastAPI – виконує оброблення запитів, керує логікою автентифікації, забезпечує взаємодію з модулями обробки та генерації рекомендацій, а також агрегує дані;

– рівень зберігання даних, що реалізується засобами MongoDB – забезпечує збереження профілів користувачів, історії активності, метаінформації курсів, налаштувань персоналізації та зворотного зв’язку.

У межах архітектурного проєктування було обрано парадигму REST як основу для взаємодії між клієнтською та серверною частинами [24]. Передача даних відбувається у форматі JSON, що забезпечує високий рівень сумісності з типовими бібліотеками клієнтського середовища. Запити, які надходять від користувача, обробляються серверною частиною, де FastAPI відповідає за маршрутизацію, перевірку вхідних даних, ініціалізацію викликів до рекомендаційного модуля та формування відповіді. Рекомендаційний модуль реалізовано як окремий логічний компонент, що функціонує на основі гібридного

підходу до фільтрації та отримує необхідну інформацію із бази даних або поточного профілю взаємодії.

Обрання саме FastAPI як серверного каркаса обґрунтовано його асинхронною природою, що дає змогу ефективно обробляти численні одночасні запити, підтримувати швидкодію і масштабованість, а також використовувати вбудовані засоби документації, валідації даних (через Pydantic) та інтеграції з іншими сервісами. MongoDB було обрано через її переваги в роботі з документно-орієнтованими структурами даних, динамічну схему та гнучкість у моделюванні вкладених об’єктів [25], що ідеально підходить для зберігання гетерогенних записів про активність користувачів.

Клієнтська частина системи реалізується відповідно до поставлених вимог на основі сучасних засобів побудови інтерфейсів користувача залежно від конкретного сценарію використання. У разі потреби у багатокомпонентному, кастомізованому інтерфейсі з динамічною взаємодією між елементами застосовується React, якщо ж ідеться про швидке створення адміністративного прототипу або аналітичного інтерфейсу – перевага надається Streamlit. Усі елементи клієнтського рівня взаємодіють із серверною частиною за допомогою HTTP-запитів через бібліотеку Axios або стандартні методи Streamlit, забезпечуючи двосторонній обмін даними з урахуванням поточного стану сесії користувача. Крім цього, інтерфейс може включати адаптивні компоненти, що підлаштовуються під пристрій користувача, механізми збереження локальних налаштувань, а також візуальні індикатори персоналізованих рекомендацій. Такий підхід сприяє покращенню UX і дозволяє ефективніше доносити релевантний контент до цільової аудиторії.

Запропонована структура забезпечує гнучкість і модульність архітектури, що дозволяє адаптувати систему під змінні умови функціонування [26], зміну кількості активних користувачів, а також специфіку поведінки в динаміці. Завдяки незалежності компонентів і чітко визначеним інтерфейсам можлива безпечна заміна, оновлення або масштабування окремих частин системи без ризику порушити цілісність загальної логіки. Так, наприклад, оновлення

алгоритму рекомендацій або впровадження нової моделі обробки не потребує змін в інтерфейсній або базовій інфраструктурі – достатньо адаптувати відповідний модуль. Загальну логіку компонування та взаємодії між модулями системи відображено на Рисунку 2.1.

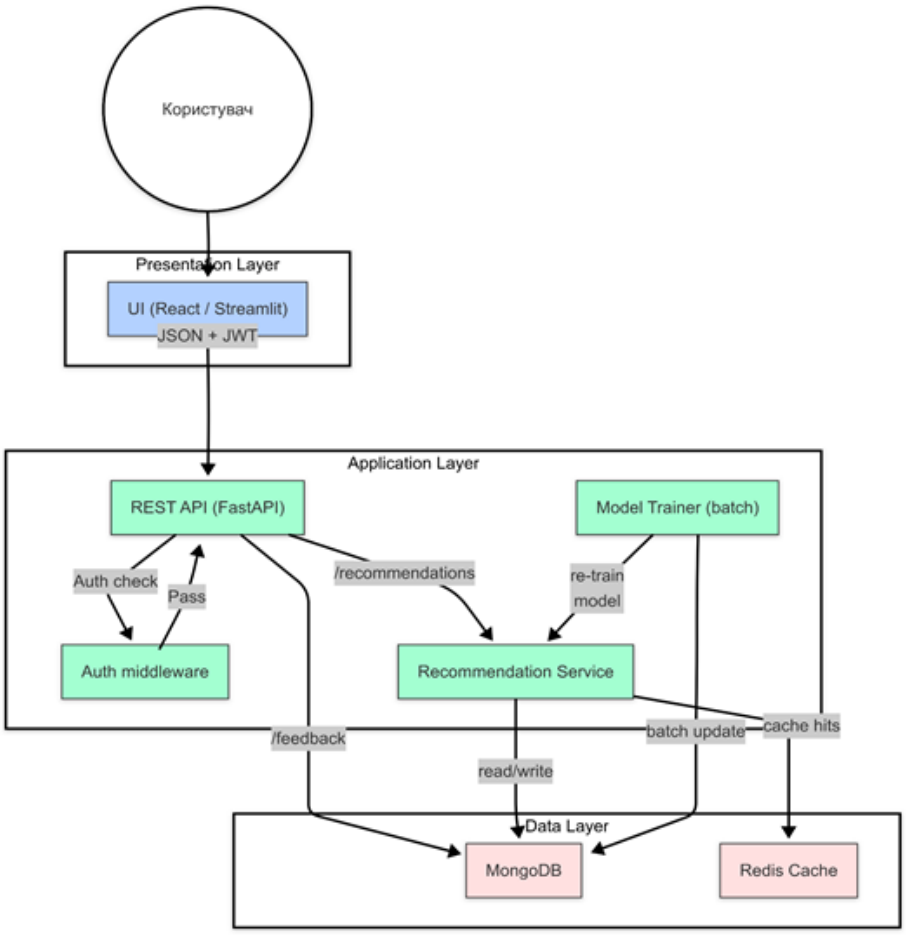


Рисунок 2.1 – Загальна архітектура системи персоналізованих рекомендацій

Важливою характеристикою проєктної архітектури є її придатність до поетапного розгортання. Завдяки поділу на незалежні компоненти можливо реалізовувати кожен блок окремо – наприклад, почати з мінімально життєздатного прототипу (MVP), який включає базові елементи рекомендаційної логіки та відображення контенту, а згодом поступово розширювати функціонал, додаючи адміністрування, аналітику, персональні фільтри тощо.

Крім того, архітектура сприяє реалізації принципів CI/CD (неперервної інтеграції й доставки), що знижує ризики помилок при впровадженні оновлень.

Структура підтримує ізольоване тестування кожного модуля, що дає змогу оперативно виявляти й усувати дефекти на рівні логіки, бази або інтерфейсу. При потребі масштабування (наприклад, у разі навантаження понад прогнозовані межі), система може бути розгорнута у кількох екземплярах із використанням контейнеризації – зокрема, через Docker – що забезпечить стабільність, розділення ресурсів і горизонтальне масштабування без потреби у складних змінах конфігурації.

Завдяки цим особливостям проєктна архітектура системи персоналізованих рекомендацій є придатною до розширення, модернізації, інтеграції з іншими сервісами, а також повністю узгоджується з сучасними вимогами до високонавантажених вебзастосунків у сфері освіти. Водночас вона зберігає достатню гнучкість для адаптації до нових бізнес-вимог, змін нормативної бази чи технологічних оновлень без потреби у кардинальній перебудові всієї системи.

2.2 Інформаційна модель та структура даних

Інформаційна модель системи персоналізованих рекомендацій була розроблена з урахуванням принципів структурованого зберігання даних у форматі документів, які властиві для документоорієнтованої парадигми баз даних. Як основну технологію збереження інформації було обрано MongoDB – розподілену нереляційну систему керування базами даних, яка забезпечує високу гнучкість, масштабованість і продуктивність при роботі з великими обсягами напівструктурованих або неструктурованих даних.

Проєктування структури даних здійснювалося відповідно до поставлених задач формування індивідуальних рекомендацій на основі поведінкових, профільних та контекстуальних характеристик користувачів [27]. Однією з головних вимог до інформаційної моделі було забезпечення швидкого доступу до ключових сутностей, зведення до мінімуму кількості зв'язків, що утворюють «вузькі місця», а також підтримка сценаріїв динамічного оновлення та аналітики.

Враховуючи вказані вимоги, було виокремлено п’ять базових колекцій: «Користувач», «Курс», «Оцінка», «Журнал дій» та «Зворотний зв’язок». Кожна колекція містить набір документів з унікальними ідентифікаторами та структурою, адаптованою під типові операції зчитування, фільтрації й агрегації. Замість використання суворої схеми, характерної для реляційних СКБД, у MongoDB реалізовано підхід до зберігання вкладених об’єктів і масивів, що дозволяє гнучко працювати з даними без постійної необхідності у JOIN-запитах.

Колекція «Користувач» виконує роль центрального профілю, що включає персональні облікові дані, налаштування персоналізації, перелік навчальних цілей, історію активності, а також часові мітки реєстрації та останньої взаємодії. Додатково, з метою адаптації навчального контенту до уподобань користувача, фіксуються параметри взаємодії з попередніми курсами, частота звернення до платформи, тривалість сесій, мова інтерфейсу та обраний візуальний режим. Для підвищення ефективності формування рекомендацій передбачено збереження вподобань, обраних категорій курсів, переваг за типом подачі матеріалу, формату оцінювання, а також історії скасованих або перезапущених курсів, що може вказувати на зміну інтересів або складність матеріалу.

Колекція «Курс» містить інформацію про навчальні модулі, включаючи унікальний ідентифікатор курсу, назву, опис, тематику, рівень складності, тривалість, мову викладання, метатеги, формат (відео, текст, інтерактивний) та інші параметри, необхідні для побудови контент-орієнтованої моделі рекомендацій. Крім того, в окремому піддокументі можуть зберігатися агреговані відгуки та статистика завершення.

Колекція «Оцінка» використовується для зберігання інформації про індивідуальні оцінки курсів користувачами, що дозволяє будувати матрицю вподобань для реалізації колаборативної фільтрації [28]. Кожен документ містить посилання на унікальні ідентифікатори користувача і курсу, числову оцінку (рейтинг), дату та тип оцінювання (явна/неявна взаємодія).

Журнал дій («InteractionLog») слугує джерелом поведінкових даних, у тому числі переглядів, початку/завершення модулів, клацань по елементах інтерфейсу,

запуску відео тощо. Завдяки структурі з високою часовою роздільністю та вказівкою на тип дії, можливо моделювати динамічний контекст користувача й виявляти шаблони поведінки, які покладено в основу рекомендаційної логіки.

Колекція «Зворотний зв'язок» містить опціональні коментарі, текстові оцінки та сигнали від користувачів щодо якості контенту, технічних проблем або точності рекомендацій. Ці дані можуть бути використані як джерело для подальшого вдосконалення системи й коригування моделі видачі персоналізованого контенту.

Взаємозв'язки між сутностями реалізовано не через прямі зовнішні ключі, а через збереження ідентифікаторів (ObjectId) інших документів. Це забезпечує логічну узгодженість при збереженні гнучкої структури та відсутності необхідності у складних транзакціях. Так, у документі оцінки зберігаються ID користувача і курсу, які зіставляються у відповідних колекціях. Аналогічно, події в журналі мають поля, що вказують на джерело дії, її ціль і часові параметри. Загальну логічну структуру інформаційної моделі, яка відображає взаємозв'язки між основними колекціями та логіку організації даних у контексті підтримки рекомендаційних сценаріїв, було візуально представлено у вигляді схеми, що зображена на Рисунку 2.2.

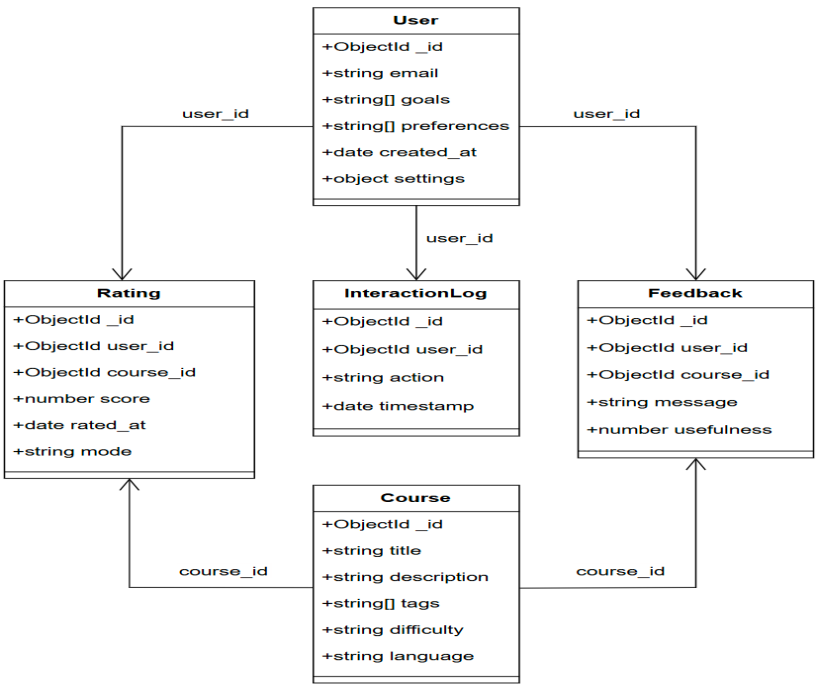


Рисунок 2.2 – Інформаційна модель системи персоналізованих рекомендацій

Для формалізованого представлення структури даних кожної з колекцій нижче подано відповідні таблиці з переліком основних полів, типів даних і їх призначення. Це дозволило точніше відобразити внутрішню будову документів та співвіднести збережувану інформацію з функціональними завданнями рекомендаційної системи.

Структуру документу «Користувач», що включає основні поля профілю користувача, їх типи та функціональне призначення, подано в Таблиці 2.1.

Таблиця 2.1 – Структура документу «Користувач»

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор користувача
email	string	Адреса електронної пошти
goals	string[]	Перелік навчальних цілей
preferences	string[]	Вподобання користувача (категорії, формати, тощо)
created_at	date	Дата створення акаунту
settings	object	Налаштування персоналізації (мова, темна тема, тощо)

Структуру документу «Курс», що містить інформацію про навчальні матеріали, тематику й параметри курсу, наведено в Таблиці 2.2.

Таблиця 2.2 – Структура документу «Курс»

Поле	Тип	Призначення
id	ObjectId	Унікальний ідентифікатор курсу
title	string	Назва курсу
description	string	Короткий опис змісту курсу
tags	string[]	Тематичні мітки для класифікації та пошуку
difficulty	string	Рівень складності (початковий, середній, просунутий)
language	string	Мова викладання

Структуру документу «Оцінка», що використовується для зберігання результатів взаємодії між користувачами та курсами, зокрема числових оцінок і контексту їх надання, наведено в Таблиці 2.3.

Таблиця 2.3 – Структура документу «Оцінка»

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор оцінки
user_id	ObjectId	Ідентифікатор користувача, який залишив оцінку
course_id	ObjectId	Ідентифікатор курсу, який було оцінено
score	number	Числова оцінка (рейтинг)
rated_at	date	Дата надання оцінки
mode	string	Тип взаємодії (явна або неявна оцінка)

Колекція «Журнал дій» структурно зберігає ключові атрибути подій, пов’язаних з активністю користувача в системі. У ній фіксуються дії, їхні часові мітки та типи взаємодій, що надалі використовуються для формування поведінкових шаблонів. Типову структуру документа наведено в Таблиці 2.4.

Таблиця 2.4 – Структура документу «Журнал дій»

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор запису дії
user_id	ObjectId	Ідентифікатор користувача, що виконав дію
action	string	Тип дії (перегляд, натискання, запуск відео тощо)
timestamp	date	Час виконання дії

Інформацію про зворотний зв’язок, залишений користувачами, а також відповідні текстові повідомлення та оцінки, подано в Таблиці 2.5.

Таблиця 2.5 – Структура документу «Зворотний зв’язок»

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор відгуку
user_id	ObjectId	Ідентифікатор користувача, який залишив відгук
course_id	ObjectId	Ідентифікатор курсу, якого стосується відгук
message	string	Текст повідомлення або коментаря
usefulness	number	Суб’єктивна оцінка корисності (якщо надається)

У результаті моделювання було досягнуто раціонального балансу між гнучкістю та структурованістю: обрана модель підтримує адаптивне масштабування [29], дозволяє ефективно зберігати й обробляти як статичні, так і динамічні дані, забезпечує відповідність сучасним підходам до побудови інтелектуальних систем у сфері освіти. Вона слугує основою для ефективного функціонування рекомендаційного модуля та може бути розширена за потреби інтеграції нових типів контенту або користувацьких сценаріїв.

2.3 Проєктування серверної частини

Серверна частина системи персоналізованих рекомендацій розроблена із використанням FastAPI – високопродуктивного асинхронного веб-фреймворку на мові Python, який дозволяє реалізувати RESTful-архітектуру з мінімальними накладними витратами та високою швидкодією. FastAPI обрано через його інтеграцію з бібліотекою Pydantic для автоматичної валідації вхідних даних [30], а також підтримку типізації, асинхронного виконання та автогенерації документації OpenAPI. Сервер виконує роль центральної ланки між клієнтською

частиною та внутрішніми модулями системи, забезпечуючи маршрутизацію запитів, обробку даних, перевірку чинності токена автентифікації, виданого основною платформою і взаємодію з базою даних MongoDB.

У контексті даного застосунку серверна частина відповідає за координацію не лише CRUD-операцій, а й логіки рекомендацій, ведення журналу активностей, обробки оцінок, генерації сесій персоналізації та обробки зворотного зв'язку. Реалізовано ізольовану обробку запитів через контролери, які прив'язані до окремих маршрутів. Така структура дозволяє уникати дублювання логіки та підвищує масштабованість проєкту. Крім того, організація коду передбачає відокремлення внутрішніх бізнес-правил (в окремих сервісах) від шарів маршрутизації та доступу до даних, що відповідає принципам Clean Architecture. Таким чином, серверна частина не лише реагує на зовнішні запити, а й виконує роль координуючого середовища, що управляє логікою персоналізації та відгуку системи на поведінку користувача [31].

На рівні архітектури застосунку серверна частина розташована між клієнтським інтерфейсом (зовнішній рівень, реалізований за допомогою React або Streamlit) і внутрішніми службовими та логічними модулями. Всі взаємодії здійснюються через HTTP-протокол з використанням чітко визначених маршрутів. Запити від користувача надходять через UI, передаються до FastAPI-сервера, де проходять початкову перевірку (валідація Pydantic-моделями), після чого спрямовуються до відповідних обробників – компонента перевірки токен, блоку збереження даних або рекомендаційної логіки. Обробка запитів організована за принципами REST: методи GET використовуються для запиту даних [32] (наприклад, отримання персоналізованих рекомендацій), POST – для надсилання нової інформації (реєстрація оцінок, логів), PUT – для оновлення, DELETE – для видалення (опційно).

Особливу увагу зосереджено на перевірці чинності токена, виданого базовою навчальною платформою. Для захисту маршрутів застосовується формат JWT (JSON Web Token), що передається у заголовку HTTP-запиту. Підпис токена перевіряється у проміжному шарі middleware; таким чином неавторизовані запити

відсіюються ще до етапу маршрутизації. Після успішної верифікації запит передається до відповідного контролера.

Взаємодія між серверною частиною і функціональними модулями системи персоналізованих рекомендацій подана на Рисунку 2.3. У цій схемі наочно відображено маршрути передачі даних між клієнтським інтерфейсом, API, модулями авторизації, обробки даних, генерації рекомендацій та підсистемою зберігання даних.

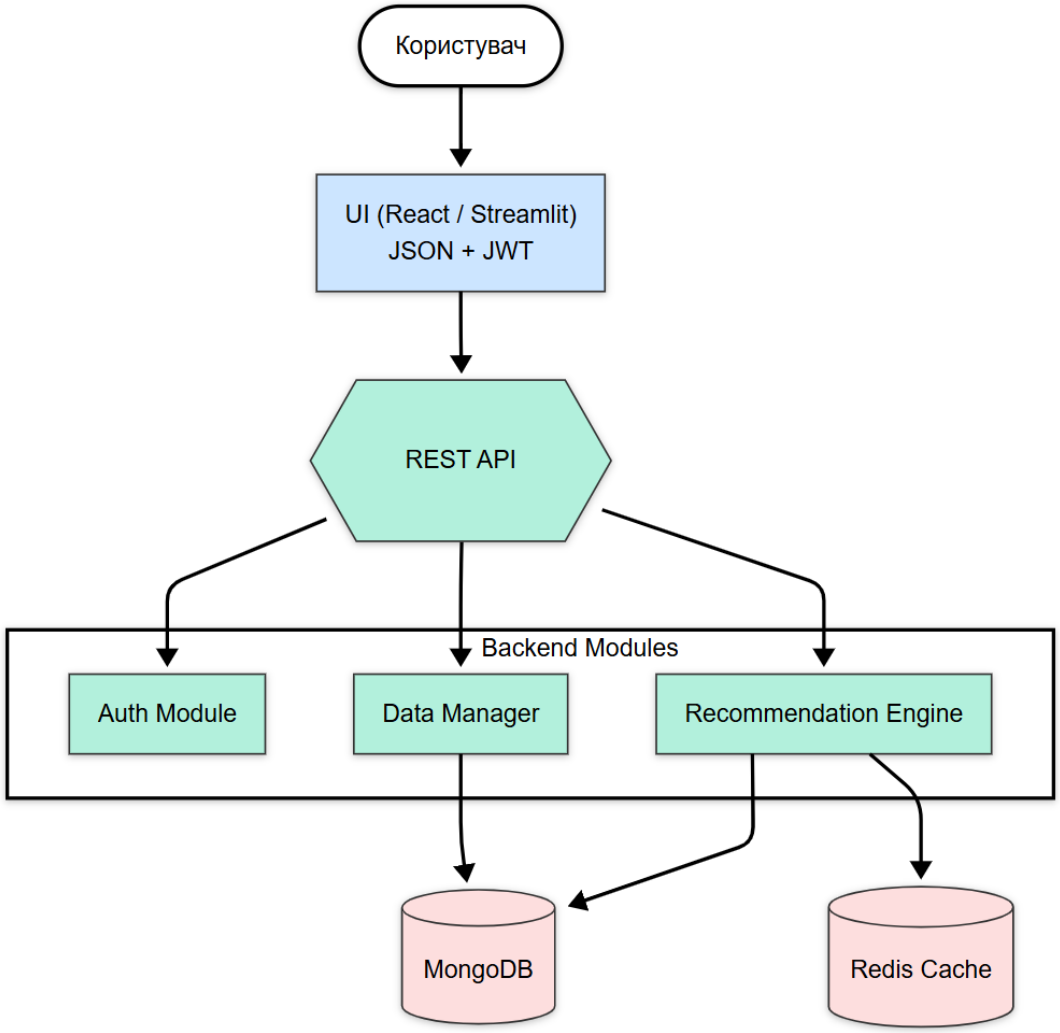


Рисунок 2.3 – Взаємодія серверної частини з модулями системи рекомендацій

Окремо було виділено ключові маршрути REST API, що реалізовано у межах серверної частини. Наведено інформацію про метод запиту, кінцеву точку (endpoint), стислий опис функціональності та відповідний обробник. До обов'язкових належать маршрути для отримання рекомендацій, реєстрації

оцінок, логування взаємодій користувача та збереження зворотного зв’язку. Повну структуру наведено в Таблиці 2.6.

Таблиця 2.6 – Основні маршрути REST API застосунку

Метод	Шлях	Опис	Обробник
GET	/recommendations	Отримання персоналізованих рекомендацій для користувача	Recommendation Engine
POST	/rating	Надсилання оцінки курсу користувачем	Rating Service
POST	/log	Логування взаємодій користувача з інтерфейсом	Logger
POST	/feedback	Збереження зворотного зв’язку	Feedback Handler
GET	/auth/check	Перевірка чинності токена, виданого LMS	Auth Middleware
GET	/admin/metrics	Отримання метрик моделі (точність, recall, кількість рекомендацій за добу)	AdminController
POST	/admin/retrain	Ініціювання повторного навчання моделі з оновленими даними	TrainingService

Підсумовуючи, можна зазначити, що серверна частина системи побудована з урахуванням принципів модульності, масштабованості та безпеки. Вона виконує роль не лише комунікаційного посередника між клієнтом і БД, а й є точкою входу для основних бізнес-процесів застосунку. Завдяки використанню FastAPI, реалізація API є простою для масштабування і подальшого розширення, що дозволяє інтегрувати нові функції (наприклад, A/B тестування, моніторинг якості рекомендацій, аналітичні сервіси) без порушення існуючої структури. Таким чином, серверна частина не лише задовольняє поточні потреби взаємодії, а й забезпечує технічну стійкість до змін, адаптацію до зростання навантаження та гнучкість у впровадженні нових рішень. Це робить її ключовим елементом архітектури, здатним підтримувати довготривалу еволюцію застосунку без суттєвого рефакторингу.

2.4 Проєктування клієнтської частини

Клієнтська частина системи персоналізованих рекомендацій відіграє ключову роль у забезпеченні зручної та інтуїтивно зрозумілої взаємодії користувача з функціоналом платформи. Її проєктування здійснюється відповідно до принципів адаптивності, модульності та компонентної структури, що дозволяє ефективно реалізовувати як типові сценарії (перегляд курсу, оцінювання, визначення освітніх інтересів), так і більш складні дії (аналіз рекомендацій, зворотний зв'язок, налаштування персоналізації).

Структура системи управління змістом охоплює логіку організації персоналізованого контенту, включає компоненти для налаштування інтересів і збереження зворотного зв'язку, керування навчальним прогресом і збереження зворотного зв'язку. Вміст інтерфейсу формується динамічно на основі даних з API та з урахуванням контексту користувача.

Згідно з обраною архітектурною моделлю, клієнтський інтерфейс базується на компонентному підході з поділом на логічно незалежні модулі. Основу структури становлять наступні елементи:

– головна сторінка з вітальним блоком і рекомендаційною панеллю [34];

– сторінка профілю користувача з можливістю перегляду та коригування освітніх пріоритетів;

– сторінка перегляду курсу з деталями, рейтингами, кнопкою початку/продовження;

– форма оцінки та зворотного зв'язку;

– адмін-панель модуля.

Усі сторінки пов'язано між собою через клієнтський роутинг, реалізований за допомогою бібліотеки React Router. Таким чином, забезпечується безперервна взаємодія без перезавантаження сторінки. Логіку відображення елементів побудовано за умовами: наприклад, якщо користувач не авторизований, його перенаправляє на сторінку входу; якщо рекомендацій ще немає – показується запит на заповнення вподобань.

Для кожної сторінки визначено набір відповідальних елементів інтерфейсу. Наприклад, на головній сторінці це Header, RecommendationsList, CourseCard; на сторінці профілю – PreferencesEditor, SaveButton. Взаємодія з сервером відбувається через окремий модуль API, що інкапсулює усі HTTP-запити (Axios), централізовано обробляє помилки та відповіді сервера [35]. На Рисунку 2.4 представлено загальну структуру клієнтської частини системи, яка демонструє основні компоненти інтерфейсу та їх взаємозв'язки.

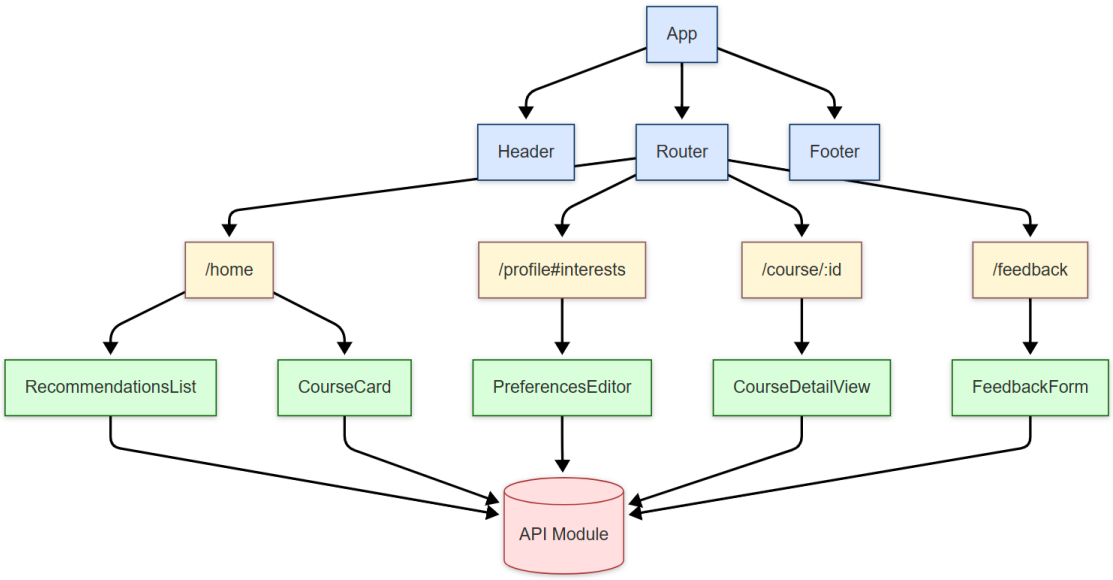


Рисунок 2.4 – Структура клієнтської частини системи рекомендацій

У структурному плані застосунок побудований за принципом розділення логіки, подання та даних (наближено до моделі MVC):

– модель – це контекст користувача, налаштування, поточні рекомендації;

– подання – набір інтерфейсних елементів;

– контролер – реалізовано у вигляді callback-функцій і useEffect-хуків, що ініціюють завантаження або оновлення даних.

Перечислені компоненти реалізовано у вигляді структурованої сукупності багаторазово використовуваних елементів інтерфейсу, що охоплюють як базові віджети для взаємодії з користувачем, так і спеціалізовані блоки для роботи з персоналізованим контентом. Їх організація базується на принципах інкапсуляції, повторного використання та гнучкої конфігурації. Щоб деталізувати цей підхід,

нижче було наведено перелік основних інтерфейсних елементів, які складають основу візуальної частини застосунку та відповідають за реалізацію функціональних сценаріїв користувача. У Таблиці 2.7 наведено перелік основних елементів інтерфейсу, які використовуються у клієнтській частині системи з коротким описом їх функціонального призначення.

Таблиця 2.7 – Перелік основних компонентів клієнтської частини

Назва компонента	Призначення
Header	Відображення заголовка, логотипу та пунктів навігації
Footer	Виведення службової інформації (наприклад, копірайт)
RecommendationsList	Список персоналізованих рекомендацій користувача
CourseCard	Окрема картка курсу з назвою, описом та дією
PreferencesEditor	Редактор освітніх інтересів користувача (теги, бажані теми
PreferencesEditor	Редактор освітніх інтересів користувача
CourseDetailView	Відображення повної інформації про обраний курс
FeedbackForm	Форма оцінки, зворотного зв’язку та коментарів

Інтерфейс модуля містить компактну панель навігації з логотипом та пунктами “Рекомендації” і “Налаштування інтересів”, решту посилань забезпечує базова платформа. Активна сторінка підсвічується відповідним індикатором, а в окремих розділах можуть використовуватись елементи навігації, що відображають ієрархію сторінок (breadcrumbs) для кращої орієнтації. Меню є фіксованим і доступним на більшості сторінок, що забезпечує сталість навігаційної структури.

Окремі функціональні можливості передбачають локальне збереження налаштувань (theme, language) через localStorage, а також підтримку темної теми. Для підвищення доступності застосовано семантичну розмітку (aria-labels),

логічне групування елементів та мінімізацію кількості дій для досягнення цілі (перегляд детальної інформації про курс, надсилання зворотнього зв'язку).

Загальний сценарій взаємодії користувача з клієнтською частиною передбачає наступну послідовність: спочатку авторизація, отримання дійсного токена платформи, далі налаштування інтересів, отримання індивідуального переліку рекомендацій, ознайомлення з деталями курсів, оцінювання переглянутого контенту та перегляд особистої статистики активності. Кожен з етапів реалізовано у вигляді окремої сторінки інтерфейсу, яка оновлюється відповідно до поточних дій користувача. Рисунок 2.5 ілюструє логіку переходів між сторінками інтерфейсу відповідно до сценаріїв взаємодії користувача.

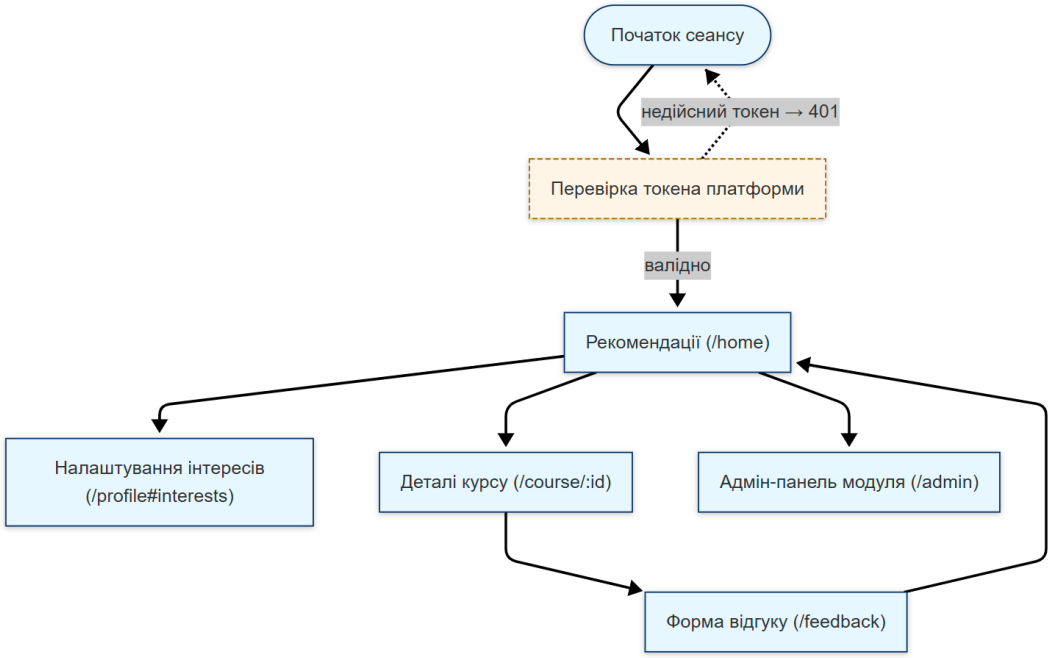


Рисунок 2.5 – Логічна схема переходів між сторінками вебінтерфейсу

Таким чином, клієнтська частина побудована з урахуванням кращих практик проєктування інтерфейсів: розділення відповідальностей, повторне використання компонентів, адаптивність і забезпечення доступності. Її структура повністю узгоджується з архітектурними рішеннями, прийнятими в попередніх підрозділах, і забезпечує надійний інтерфейс для взаємодії користувача з рекомендаційною системою.

2.5 Прототипи інтерфейсу та дизайн

Прототипування клієнтського інтерфейсу дало змогу на етапі проєктування перевірити зручність розміщення інформації, послідовність переходів та відповідність макету вимогам доступності. Робота виконувалася ітераційно: спершу були створені низькодеталізовані вайрфрейми, що окреслювали базові зони екрана, після чого кожну ітерацію узгоджували з консультантом проєкту та адаптували до нових зауваг щодо юзабіліті.

На підставі аналізу сценаріїв було виокремлено чотири логічні зони контенту:

– персоналізовані добірки курсів;

– детальні сторінки навчальних матеріалів;

– налаштування профілю користувача;

– форма зворотного зв’язку.

Кожна зона отримала зрозумілу назву, а її розташування не перевищувало двох кроків від головної сторінки, чим мінімізувалося когнітивне навантаження. Додатково було введено індикатори прогресу та системні повідомлення, що сповіщали про збереження налаштувань або успішне надсилання відгуку.

Логіку переходів реалізовано засобами клієнтського маршрутизатора React Router [33]. Структура містила маршрути підсистеми /home, /course/:id, /profile#interests, /feedback. Спроба відкрити захищений шлях у разі недійсного токена платформа повертає HTTP 401, після чого виконується глобальне перенаправлення LMS. Послідовність дій узгоджувалася із сценаріями авторизації, налаштування профілю, отримання рекомендацій, перегляду курсу та надання відгуку, що забезпечувало передбачуваність навігації.

Після затвердження маршрутизації було змодельовано середньо деталізовані макети у Figma. Для кожного ключового екрана фіксувалися: сітка 12 колонок, відступи кратні 4 px, контентні блоки–плейсхолдери. На заключній ітерації макети перевірялися на відповідність принципам доступності за допомогою плагіна Stark.Усі прототипи стилізувалися у нейтральній палітрі Tailwind (#F3F4F6 – фон, #2563EB – акцент) зі шрифтом Inter; контрастність

тексту перевищувала 4,5:1. На Рисунку 2.6 наведено макет головної сторінки: у верхній частині розташовано фіксований Header із пунктами навігації, під ним – блок «Рекомендовані курси» з трьома збільшеними картками CourseCard (назва, опис, рейтинг, кнопка переходу).

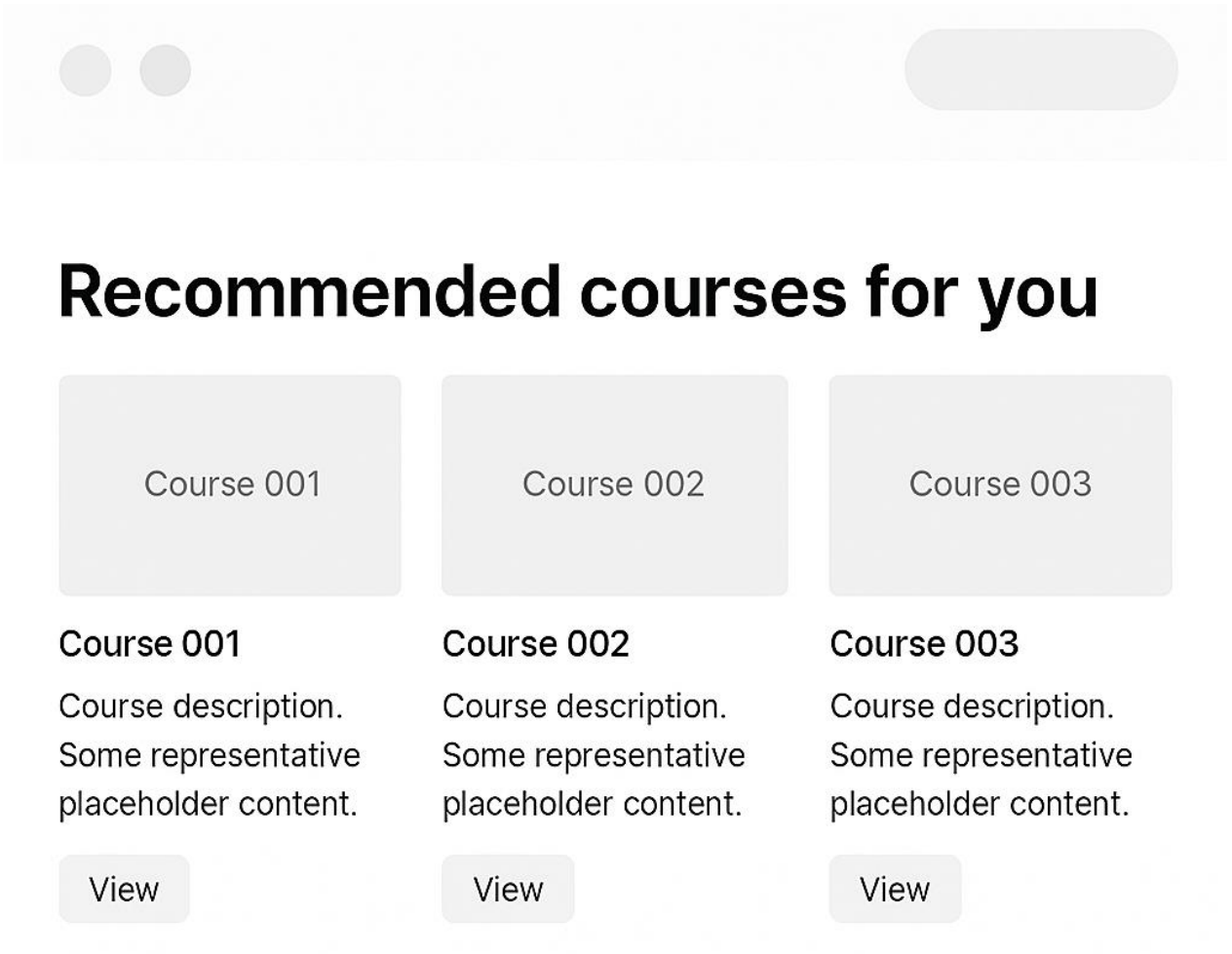


Рисунок 2.6 – Прототип сторінки з персоналізованими рекомендаціями

Під час створення прототипів було враховано вимоги WCAG 2.1: контрастність тексту не нижча ніж 4,5:1, інтерактивні компоненти мали чіткий фокус-стан, а макет коректно масштабувався до екранів від 320 px. Використання утилітарних класів Tailwind спрощувало подальше масштабування інтерфейсу та забезпечувало стилістичну послідовність між прототипом і майбутньою реалізацією. Таким чином, отримані прототипи підтвердили відповідність навігаційної моделі очікуванням цільової аудиторії.

2.6 Обґрунтування вибору технологій

Системний аналіз наявних технологічних рішень для фронтенд і бекенд-розробки вебзастосунку системи персоналізованих рекомендацій дозволив визначити інструментарій, що найкраще відповідає вимогам масштабованості, продуктивності та інтеграції з алгоритмами машинного навчання, описаними у попередніх підрозділах.

Критерії відбору технологій сформувалися з огляду на нормативні та експлуатаційні чинники й охоплювали такі аспекти:

– швидкість розробки та зрозумілість для команди;

– продуктивність і здатність до горизонтального масштабування;

– наявність зрілої екосистеми бібліотек і плагінів;

– підтримка типізації, тестованість і безпечна обробка даних;

– сумісність із сервісами кешування та контейнеризацією;

– адаптивність інтерфейсу для різних типів клієнтських пристроїв;

– простота CI/CD-підключення та можливість автоматизованого розгортання;

– відповідність тенденціям розвитку вебстандартів і універсальність;

– загальна вартість володіння (TCO), включаючи ліцензування та інфраструктурні витрати, що скорочуються завдяки open-source-характеру обраних інструментів.

На фронтенд-рівні було проаналізовано класичний стек Dynamic HTML (DHTML), що поєднує HTML5, CSS3 та імперативні JavaScript-скрипти з прямим керуванням об’єктною моделлю документа (DOM) і системою подій браузера. У традиційній реалізації керування елементами виконувалося через методи querySelector, addEventListener тощо. Такий підхід підтримував крос-браузерність і низький поріг входу, однак мав обмеження: відсутність чіткої компонувальної моделі, громіздкість обробки складних ієрархій елементів і значне навантаження на головний потік при частих змінах стану. З огляду на це було обрано React, що завдяки віртуальному DOM та системі Synthetic Events

автоматизує оновлення інтерфейсу й абстрагує застосунок від ручної роботи з DOM-подіями.

Для стилізації застосовано Tailwind CSS. Порівняно з класичними каскадними таблицями стилів і готовими UI-фреймворками (Bootstrap, Bulma) Tailwind забезпечив атомарний підхід до опису стилів, мінімізував конфлікти класів і зменшив фінальний обсяг CSS завдяки механізму PurgeCSS [36]. Адаптивні утиліти Tailwind дозволили швидко формувати інтерфейсні шаблони для різних роздільних здатностей без ручного прописування медіазапитів.

Щодо сценарних засобів у складі вебсторінок було розглянуто класичну технологію Ajax (XMLHttpRequest) та її реалізацію через jQuery.ajax(). Попри поширеність, такий інструментарій вимагав значного обсягу імперативного коду й не підтримував автоматичну типізацію даних. У сучасному стеку функціональність Ajax реалізовано засобами Fetch API та бібліотекою Axios, що забезпечили Promise-орієнтований підхід, централізовану обробку перехоплювачів запитів і нативну роботу з JSON-схемами.

На бекенд-шарі порівнювалися два підходи. Перший ґрунтувався на мові PHP у поєднанні з фреймворком Laravel. Переваги включали зрілу спільноту, потужний ORM-механізм Eloquent і підтримку шаблонізатора Blade. Недоліки ж, включали наступне: низька ефективність асинхронних обчислень, громіздкий цикл оновлень пакетів і відсутність нативної інтеграції з бібліотеками машинного навчання. Другий підхід передбачав використання Python із фреймворком FastAPI. FastAPI на базі ASGI-сервера Uvicorn демонструє високу продуктивність при невисокому споживанні ресурсів, підтримує автоматичну OpenAPI-документацію та перевірку типів за допомогою pydantic. Важливо, що Python має природну сумісність із бібліотеками NumPy, Pandas і Scikit-learn, залученими у модулі рекомендацій.

Щодо системи керування базами даних розглядалися MySQL та MongoDB. MySQL із декларативною мовою SQL-запитів забезпечує ACID-транзакції й чітко нормалізовану схему, однак ускладнює гнучке розширення профілю користувача та зберігання вкладених масивів тегів. MongoDB з документною структурою BSON дозволяє динамічно змінювати схему, виконувати агрегувальні запити без

JOIN-операцій і масштабувати кластери горизонтальним розподілом даних. Доступ до даних реалізовано через PyMongo; Eloquent / SQLAlchemy розглядалися як ORM-альтернативи для реляційних СКБД.

Обмін даними між клієнтом і сервером базується на REST-парадигмі поверх протоколу HTTPS із форматом JSON. Від застосування XML-RPC, SOAP та передачі даних у форматах XML чи CSV відмовлено через надмірну вербозність і складність інтеграції з JavaScript-клієнтами. Для авторизації використано JWT-токени з алгоритмом HS256, що забезпечує незалежність сесій і масштабованість без стану. Передбачено також використання WebSocket як механізму для надсилання push-оновлень – наприклад, у разі появи нових персоналізованих рекомендацій або зміни статусу навчального контенту без потреби ручного оновлення сторінки.

DevOps-інфраструктуру підтримують Docker-контейнери, що уніфікують середовища розробки та продакшен-запуску, а також GitHub Actions, які автоматизують CI/CD-цикли й запускають Pytest-тести з контролем покриття > 90 %. Це рішення мінімізує людський фактор та спрощує розподілену роботу над репозиторієм. Для захисту середовища від зловживань і несанкціонованого доступу реалізовано ізоляцію сервісів у межах контейнерів, контроль конфігурацій середовища та централізоване логування. Додатково враховано основні принципи безпеки: уникнення XSS шляхом серверної фільтрації даних, захист від CSRF-атак через токенізацію запитів, а також обмеження частоти запитів для запобігання зловживанню (rate limiting).

Порівняльна оцінка класичного та обраного стеків за визначеними критеріями подана в Таблиці 2.8.

Таблиця 2.8 – Порівняння ключових технологій за критеріями відбору

Шар	Класичний стек	Обраний стек
Клієнт	HTML + CSS + jQuery	React + Tailwind
Транспорт	Ajax (XMLHttpRequest)	Fetch / Axios

Продовження Таблиці 2.8

Шар	Класичний стек	Обраний стек
Сервер	PHP + Laravel	Python + FastAPI
База даних	MySQL	MongoDB

Узагальнюючи зроблений вибір технологій, можна стверджувати, що запропонований стек FastAPI, MongoDB, React та Tailwind є найбільш придатним для реалізації системи персоналізованих рекомендацій з огляду на її вимоги до масштабованості, асинхронної обробки, адаптивного інтерфейсу й підтримки машинного навчання. Таке поетапне розмежування дозволяє чітко зафіксувати переваги сучасного підходу над застарілим у контексті конкретних функціональних вимог до системи, включаючи продуктивність, масштабованість та гнучкість у роботі з даними.

2.7 Висновки. Основи для побудови програмної реалізації

У процесі проєктування сформовано цілісну концепцію вебзастосунку системи персоналізованих рекомендацій, що охоплює структурні, функціональні та інфраструктурні аспекти реалізації. Системна логіка базується на трирівневій клієнт–серверній архітектурі: FastAPI взаємодіє з MongoDB, а React / Tailwind CSS обслуговує клієнтський інтерфейс, забезпечуючи низьку затримку, асинхронну обробку та горизонтальне масштабування.

Документно-орієнтована модель даних включає колекції users, courses, events і feedback, організовані як денормалізована «модель для читання». Агрегаційні конвеєри та розподілення часових рядів за сегментами дають змогу накопичувати телеметрію й обробляти поведінкові дані в реальному часі. Цілісність і формат повідомлень перевіряються pydantic-схемами із генерацією специфікації OpenAPI 3.1.

Клієнтський шар реалізовано у форматі односторінкового застосунку: маршрутизація React Router підтримує ліниве завантаження, тому початковий пакет JavaScript становить приблизно 120 кілобайтів. Tailwind разом із PurgeCSS скорочує підсумковий CSS-файл до близько восьми кілобайтів, що покращує час першого відображення. Поступове вдосконалення завантаження шрифтів і графіки забезпечує відповідність вимогам WCAG 2.1.

Система безпеки охоплює контекстне екранування даних, подвійні CSRF-токени, обмеження частоти у сто запитів за хвилину для однієї IP-адреси та ізоляцію сервісів усередині Docker-контейнерів із профілем AppArmor. Канал WebSocket призначено для надсилання миттєвих сповіщень; за відсутності такої потреби застосовується довготривале опитування.

Процеси розгортання організовано через GitHub Actions: кожне збирання виконує статичний аналіз коду, модульні тести Pytest із покриттям понад дев’яносто відсотків, перевірку залежностей OWASP, формує контейнерні образи з версіями за схемою SemVer та публікує їх у реєстрі. Стандартні YAML-конвеєри спрощують масштабування.

Отже, проєктування:

– підтвердило доцільність обраної архітектури та технологічного стека;

– сформувало інфологічну й фізичну модель даних із підтримкою аналітики у реальному часі;

– визначило інтерфейси взаємодії між підсистемами та протоколи передавання даних на основі JSON / WebSocket для забезпечення комунікації;

– регламентувало засоби безпеки, тестування та безперервного постачання;

– підготувало та впровадило механізми моніторингу працездатності системи й масштабування для підтримки стабільної роботи під навантаженням.

Сукупність рішень створює надійну методичну та технологічну основу для переходу до етапу програмної реалізації та випробувань, що буде розглянуто у наступному розділі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

3.1 Інструментарій, середовище та технології розробки

Для реалізації рекомендаційної системи, відповідно до поставлених завдань, було обрано набір технологій, що забезпечують ефективність розробки, стабільність роботи та масштабованість системи.

В якості основного середовища розробки було використано PyCharm, оскільки він має зручний інтерфейс, підтримує системи контролю версій та вбудовану систему дебагінгу. Для реалізації проєкту був обраний Python 3.10, що забезпечує стабільність і сумісність з бібліотеками, які використовуються в проєкті. Спочатку була протестована версія Python 3.12, однак через проблеми сумісності з бібліотеками, що мали підвищити точність рекомендацій за рахунок точніших алгоритмів, ця версія була відкинута на користь більш стабільної 3.10.

Всі необхідні бібліотеки для проєкту були зібрані у файлі requirements.txt, що дозволяє зручно встановлювати залежності через команду pip install -r requirements.txt. Основними бібліотеками, що використовуються в проєкті, є:

– pandas та numpy для обробки даних і генерації синтетичних даних;

– scikit-learn для роботи з метриками та матрицями;

– sentence-transformers для створення SBERT embeddings;

– FastAPI як веб-фреймворк для розробки API;

– Uvicorn як сервер для FastAPI.

Для забезпечення ізоляції середовища та стабільної роботи на різних платформах використовувався Docker з контейнеризацією через Docker Compose. Це дозволило автоматизувати налаштування та запуск контейнерів, а також забезпечує високий рівень переносимості між різними операційними системами. WSL2 використовувався для створення Linux-сумісного середовища на Windows, що дозволяє без проблем працювати з Docker-контейнерами.

Проєкт було налаштовано з обмеженнями в 12 GB оперативної пам’яті та 8 логічних CPU ядер, що дало змогу ефективно обробляти великі обсяги даних, водночас імітуючи типові умови фізичного сервера.

Завдяки використанню Docker система має високу переносимість і може бути розгорнута на різні сервери, такі як VPS, GCP та AWS.

Особлива увага була приділена вибору бази даних: для цього проекту було обрано MongoDB, що забезпечує гнучкість і масштабованість при роботі з неструктурованими та напівструктурованими даними. MongoDB дозволяє зберігати дані в документній структурі BSON, що ідеально підходить для зберігання мета-даних користувачів, курсів, а також історії взаємодій. Оскільки система передбачає високу динамічність даних, MongoDB надає можливість легко змінювати схеми даних без необхідності змінювати структуру всієї бази. Для забезпечення високої продуктивності при обробці запитів до бази даних також буде використовуватися шардінг.

Що стосується асинхронної обробки запитів, то FastAPI підтримує асинхронність із використанням Uvicorn як ASGI-сервера, що дозволяє одночасно обробляти багато запитів без блокування. Це особливо важливо для рекомендаторних систем, де часто необхідно обробляти великий обсяг даних за короткий час, забезпечуючи високу швидкість відповіді на запити користувачів. Використання асинхронного підходу дозволяє масштабувати систему, обробляючи великі навантаження без втрати продуктивності.

3.2 Архітектура та розгортання бази даних

Розробка бази даних для системи рекомендацій була реалізована з використанням MongoDB, яка була обрана як основний інструмент для зберігання даних. Такий вибір був зроблений через її здатність ефективно працювати з великими обсягами неструктурованих або слабо структурованих даних, що часто зустрічаються в системах рекомендацій. У цих системах дані можуть мати різну форму, часто змінюються та потребують гнучкої обробки. MongoDB справляється з цим завдяки можливості зберігати документи в колекції, що дозволяє масштабувати систему та швидко виконувати запити.

Під час розробки бази даних було визначено кілька основних колекцій. Наприклад, колекція courses містить детальну інформацію про курси: їхні унікальні ідентифікатори, назви, описи, категорії та інші характеристики. Це дозволяє користувачам швидко знаходити потрібні курси. Інша колекція – users, зберігає дані про користувачів, такі як їхній вік, стать, країна, а також історія взаємодій із курсами. Завдяки цьому система може формувати персоналізовані рекомендації, враховуючи вподобання та активність користувачів. Колекція interactions відстежує всі дії користувачів з курсами, зокрема перегляди, лайки та оцінки, що дає можливість точно налаштувати систему рекомендацій. Колекція logs фіксує різноманітні дії користувачів, що допомагає аналізувати їхню поведінку і використовувати ці дані для вдосконалення рекомендацій.

Реалізація цих колекцій і базових функцій бази даних була завершена на етапі розробки, що дозволяє безперешкодно інтегрувати її з іншими компонентами системи, зокрема з FastAPI. За допомогою цього веб-фреймворку було створено RESTful API, через яке система взаємодіє з базою даних у реальному часі, обробляючи запити та надаючи персоналізовані рекомендації.

Особливу увагу було приділено можливості масштабування бази даних та збереженню гнучкості для подальших змін у структурі даних. Для цього використовувалися інструменти для міграцій MongoDB, які дозволяють вносити зміни у схему бази без втрат даних. Завдяки цьому система здатна адаптувати структуру бази даних до нових вимог без значних витрат часу та ресурсів, що важливо для подальшого вдосконалення алгоритмів рекомендацій. Також реалізовано базову оптимізацію запитів, зокрема через індексацію полів, що часто використовуються для пошуку та фільтрації даних, таких як user_id у колекції interactions.

Це дозволяє значно зменшити час обробки запитів та забезпечити швидкий доступ до найбільш актуальних даних, що критично важливо для своєчасного надання персоналізованих рекомендацій. Крім того, використання індексації дозволяє підтримувати високу продуктивність навіть при значному збільшенні обсягів даних у майбутньому.

Для демонстрації взаємодії з базою даних та збереження результатів взаємодій користувачів через FastAPI використовуються наступні функції. Наприклад, для отримання списку курсів з бази даних:

```
@app.get("/courses", summary="List courses")
def get_courses(limit: int = 10, role: str =
Depends(get_current_role)) -> list[dict]:
    courses = list(db.courses.find().limit(limit))  # Виконується
запит до колекції courses
    for c in courses:
        c["_id"] = str(c.get("_id", ""))  # Перетворення ObjectId в
рядок
    return courses
```

Також забезпечено збереження оцінок користувачів за курсами через API. Кожна взаємодія автоматично фіксується в базі даних:

```
@app.post("/rating", summary="Submit a rating")
def submit_rating(r: BaseModel = Depends(BaseModel), role: str =
Depends(get_current_role)) -> dict[str, str]:
    rating_data = RatingIn(**r.dict())  # Отримання даних оцінки
    doc = rating_data.model_dump()
    doc["rated_at"] = datetime.utcnow().isoformat()  # Додавання
часу оцінки
    result = db.interactions.insert_one(doc)  # Вставка оцінки в
колекцію interactions
    return {"inserted_id": str(result.inserted_id)}  # Повернення
ID нової оцінки
```

Завдяки FastAPI і MongoDB, система гарантує високу швидкість обробки запитів і мінімальні затримки при виконанні операцій з базою даних, що критично важливо для збереження актуальності інформації та персоналізованих рекомендацій. Масштабованість та адаптивність бази даних були досягнуті завдяки використанню Docker для контейнеризації, що дозволяє зберігати гнучкість у налаштуванні середовища та забезпечити переносимість без втрати даних чи структури між різними платформами.

3.3 Реалізація модуля персоналізованих рекомендацій

3.3.1 Підготовка та попередня обробка навчальних даних

На початковому етапі було здійснено кілька ключових процесів підготовки даних. Спочатку зібрано дані про курси, що включали їхні унікальні ідентифікатори, назви, описи та категорії. Для подальшого використання ці дані були оброблені і збагачені, зокрема, автоматично генеруючи описи та ключові слова для кожного курсу. Це дозволило створити більш детальну метаінформацію, необхідну для подальшого аналізу.

Зокрема, процес розбиття основних категорій курсів на підкатегорії був реалізований за допомогою алгоритмів, що базуються на ключових словах у назвах і описах курсів.

Такий підхід дозволяє точно класифікувати курси за більш вузькими тематичними напрямами, що забезпечує підвищену точність рекомендацій. Ключові слова для кожної підкатегорії були визначені на основі тематичних груп, що включають специфічні терміни (наприклад, для підкатегорії "Corporate Finance" це можуть бути слова типу "mergers", "valuation", "capital"). Алгоритм аналізує наявність цих ключових слів у текстах курсів і на основі цього визначає найбільш відповідну підкатегорію. Цей підхід дозволяє точно класифікувати курси та покращити ефективність рекомендаційної системи.

Призначення курсу до конкретної підкатегорії також допомагає знизити рівень шуму при генерації рекомендацій, оскільки користувач отримує більш релевантні курси, відповідні його інтересам. Додатково, механізм автоматичного призначення підкатегорій забезпечує масштабованість системи, оскільки з часом можна додавати нові підкатегорії без необхідності значного перепроектування.

Для автоматичного призначення підкатегорій використовувалась наступна функція, яка визначає підкатегорію на основі наявності ключових слів у назвах, описах та ключових словах курсів.

Важливо зазначити, що курс може бути призначений до кількох підкатегорій, і у випадку однакової кількості співпадінь ключових слів вибір

підкатегорії здійснюється випадковим чином серед найбільш релевантних варіантів. Її код наведено нижче:

```
def assign_subcategory(orig, title, desc, kws, subcats):
    text = " ".join([str(title), str(desc), str(kws)]).lower()
    best, hits = None, 0
    for sc, keys in subcats[orig].items():
        count = sum(text.count(k.lower()) for k in keys)
        if count > hits:
            best, hits = sc, count
    return best or random.choice(list(subcats[orig].keys()))
```

Приклад даних про курси, після їх розбиття на підкатегорії, були представлені в таблиці, що наведена у Таблиці 3.1.

Таблиця 3.1 - Перших 5 записів з таблиці «udemy_subcat.csv»

course_id	course_title	is_paid	published_timestamp	subject
1070968	Ultimate Investment Banking Course	TRUE	2017-01-18T20:58:58Z	Startup & Entrepreneurship
1113822	Complete GST Course & Certification - Grow Your CA Practice	TRUE	2017-03-09T16:34:20Z	Investment Banking
1006314	Financial Modeling for Business Analysts and Consultants	FALSE	2016-12-19T19:26:30Z	Financial Analytics
1210588	Beginner to Pro - Financial Analysis in Excel 2017	TRUE	2017-05-30T20:07:24Z	Financial Analytics
1011058	How To Maximize Your Profits Trading	TRUE	2016-12-13T14:57:18Z	Personal Finance

Цей етап підготовки даних є важливим, оскільки він визначає структуру для подальшої обробки і моделювання, що є основою для генерації рекомендацій.

Завдяки підкатегоріям, отриманим на етапі splitter.py, система змогла сформувати ключові слова, опис і резюме для кожного курсу, що є важливим етапом для подальшої обробки даних та генерації рекомендацій.

Для кожної підкатегорії було використано відповідні шаблони з DESCRIPTION_TEMPLATES, SUMMARY_TEMPLATES і KEYWORD_POOLS.

Ці шаблони містять від 12 до 18 унікальних варіантів, залежно від тематики підкатегорії. Кожен шаблон адаптований під специфіку курсу, що дозволяє генерувати опис та резюме, які найкраще відображають суть матеріалу. Для деяких категорій використовуються також мости – терміни, які можуть перетинатися між підкатегоріями, що покращує контекстуальну точність описів.

DESCRIPTION_TEMPLATES включають шаблони описів, що детально описують курс, використовуючи змінні такі як title, keywords, level, що дозволяє варіювати текст для кожного курсу.

SUMMARY_TEMPLATES генерують короткі, але інформативні резюме, що містять ключові моменти курсу.

KEYWORD_POOLS включають набір ключових слів, які використовуються для генерації ключових слів, що відповідають тематиці курсу.

Цей процес дозволяє створити текстову репрезентацію кожного курсу, що містить унікальне поєднання ключових слів, описів та резюме, які будуть використані для створення SBERT-ембеддингів, що є критично важливим для порівняння та класифікації курсів. Прикладом генерації опису слугує напупний фрагмент коду:

```
def random_summary(subject: str, title: str, rng) -> str:
    # Отримання шаблонів для підкатегорії
    templates = SUMMARY_TEMPLATES.get(subject, ["Практичний курс по
{subject}."])

    # Фільтрація коротких та загальних шаблонів (зберігаємо шанс
<10% для їх використання)
    long_templates = [t for t in templates if len(t) > 35 and
"roadmap" not in t and "fast" not in t]

    # Визначаємо, чи використовувати довгі шаблони, якщо шанс > 0.1
    use_long = rng.random() > 0.1 and long_templates
```

```
    # Вибір шаблону для резюме
    template = rng.choice(long_templates if use_long else
templates)

    # Вибір рівня курсу (beginner, intermediate, advanced)
    level = rng.choice(["beginner", "intermediate", "advanced"])

    # Формування резюме за допомогою шаблону
    s = template.format(subject=subject, title=title, level=level)

    # Якщо шаблон не містить {title}, додаємо його до кінця
    if "{title}" not in template and title:
        s += f" – {title}"

    return s
```

Подальша обробка даних полягала у генеруванні вищезгаданих SBERT-ембеддингів для кожного курсу, що дозволяє ефективно порівнювати курси між собою на основі їхнього змісту. Модель SBERT (Sentence-BERT), використана для генерації ембеддингів, є модифікацією класичної моделі BERT, оптимізованою для задач порівняння текстів. У порівнянні з традиційним BERT, SBERT створює більш компактні та ефективні вектори для порівняння текстів, що дає можливість знаходити схожі курси за змістом.

Модель SBERT («all-mpnet-base-v2») генерує вектори довжиною 768 елементів, що дозволяє точніше відображати смислові зв'язки між курсами. Це дуже важливо для рекомендаційної системи, оскільки дозволяє більш точно оцінювати схожість курсів на основі їхнього текстового вмісту (назви, опису, резюме). Завдяки такій великій кількості параметрів, модель здатна захоплювати глибші контексти і значення тексту, що покращує результати рекомендацій.

Генерація ембеддингів за допомогою SBERT полягає в наступному:

– тексти курсів (назва, опис, резюме) об'єднуються в один рядок;

– модель SBERT перетворює цей рядок у числовий вектор;

– вектори порівнюються між собою для оцінки схожості між курсами.

Цей процес можна продемонструвати наступним фрагментом коду:

```
# Генерація SBERT-ембеддингів для тексту
model = SentenceTransformer("all-mpnet-base-v2")

# Комбінація тексту з назви, опису та резюме курсу
```

```
text_for_embed = df["course_title"] + " " + df["description"] + " "
+ df["summary"]

# Токенізація та створення ембеддингів
encodings = model.tokenizer(
    text_for_embed.tolist(),
    padding="longest", truncation=True, max_length=128,
return_tensors="pt"
)
embeddings = model.encode(text_for_embed.tolist(),
show_progress_bar=True)

# Додавання нормалізованих ембеддингів до датафрейму
df["sbert_embedding"] = embeddings
```

Наступним етапом у підготовці даних є генерація синтетичних взаємодій між користувачами та курсами. Це є природним кроком, оскільки після формування повної бази даних курсів, що включає назви, описи, ключові слова та резюме, необхідно мати дані про взаємодії користувачів із цими курсами. Оскільки на даний момент не існує відкритих, релевантних датасетів, що містять реальні взаємодії користувачів з курсами в контексті навчальних платформ, створення синтетичних даних є оптимальним рішенням. Такий підхід дозволяє зібрати необхідні дані під контролем, зберігаючи гнучкість у їхньому розподілі та специфікаціях, що забезпечує можливість ефективного тренування моделей на контрольованих даних.

У скипті, що реалізує генерацію синтетичних взаємодій, кожному користувачеві створюється профіль, що включає його вік, стать, країну, рівень освіти та вподобання за темами курсів.

Згідно з типом користувача, визначаються його переваги: для спеціалістів – один предмет, для фанатів – два, для багатьох користувачів з рівними інтересами – три, для дослідників – більше. Кожен користувач взаємодіє з курсами, вибраними згідно з його інтересами. Наприклад, для користувача класу «specialist» вибираються курси лише з однієї підкатегорії, в той час як «explorer» має доступ до курсів із усіх доступних категорій.

Один з важливих аспектів цього етапу – це cold-start користувачі, які не мають жодних попередніх взаємодій з платформою. Для таких користувачів створюються синтетичні лайки на популярні курси, щоб «розігріти» їх активність

на платформі і забезпечити перші дані для подальшої генерації рекомендацій. Цей етап критичний для моделі, оскільки без таких даних система не зможе адаптуватися до нових користувачів.

Згенеровані синтетичні профілі користувачів та їх взаємодії з курсами дозволяють створити достатній обсяг даних для подальшого тренування моделі. Нижче наведено частину коду, який генерує ці профілі та вибирає курси для користувачів на основі їх інтересів:

```
# Генерація профілів користувачів
def build_user_profiles(n_users: int, seed: int) -> Dict[str,
UserProfile]:
    rng = np.random.default_rng(seed)
    profiles: Dict[str, UserProfile] = {}
    for idx in range(1, n_users + 1):
        uid = f"user_{idx:05d}"
        r = rng.random()
        if r < 0.12:       # specialist
            utype, n = "specialist", rng.integers(25, 45)
            fav = [rng.choice(SUBJECTS_LST)]
            weights = [1.0]
            exploration = rng.uniform(0.04, 0.08)
        # інші типи користувачів (fan, multi, explorer) генеруються
подібним чином

        profiles[uid] = UserProfile(
            user_id=uid, user_type=utype, n=n,
            fav_subjects=fav, weights=weights,
exploration=exploration,
            age=rng.integers(18, 45),
            gender=rng.choice(GENDER, p=[0.49, 0.47, 0.04]),
            country=rng.choice(COUNTRIES,
p=[.19,.12,.17,.08,.11,.09,.08,.06,.05,.05]),
            education="bachelor"
        )
    return profiles
```

Завдяки реалізації процесу генерації синтетичних даних, система здатна забезпечити повний цикл підготовки даних для тренування моделі рекомендацій. Починаючи з розбиття курсів на підкатегорії, до генерації синтетичних взаємодій, що включають як позитивні, так і негативні приклади, весь процес дозволяє створити великий обсяг даних, необхідний для навчання моделі. В результаті, було отримано різноманітні взаємодії між користувачами і курсами, що відповідають реалістичному використанню платформи.

Також варто зазначити, що за потреби, наприклад у випадку необхідності протестувати модель на данних більшої або меншої щільності ніж в реальних системах, за допомогою CLI-ключів, система дозволяє гнучко налаштовувати генерацію даних відповідно до потреб проекту. Наприклад --n_users – цей параметр дозволяє задати кількість користувачів, формуючи потрібну вибірку, для якої генеруються профілі. Або --min_per_course – ключ для налаштування мінімальної кількості взаємодій для кожного курсу. Якщо потрібно гарантувати, що кожен курс має не менше 20 взаємодій, можна вказати --min_per_course 20, щоб уникнути ситуацій, коли деякі курси отримують занадто мало взаємодій, що може вплинути на якість рекомендацій. Таким чином, завдяки цьому підходу, система може бути налаштована для роботи з різними сценаріями та обсягами даних, що дає змогу створювати більш точні та надійні рекомендаційні моделі для реальних застосувань.

3.3.2 Навчання та оптимізація рекомендаційної моделі

Для тренування моделі персоналізованих рекомендацій було використано гібридний підхід, який комбінує кілька методів фільтрації: content-based filtering, collaborative filtering (CF) та popularity-based filtering. Така комбінація дозволяє моделі враховувати різні аспекти даних, включаючи текстові характеристики курсів, їхню популярність серед користувачів, а також взаємодії між користувачами та курсами. Інтеграція цих підходів сприяє підвищенню точності рекомендацій, оскільки вона комбінує різні типи інформації. Перед тренуванням моделі було виконано необхідні етапи підготовки даних, які включали:

– створення TF-IDF матриці, що відображає важливість слів у описі курсів;

– перетворення текстових описів курсів у векторні ембеддинги за допомогою SBERT, що дає змогу моделі порівнювати курси за змістом.

Ці етапи дозволяють системі аналізувати семантичну схожість між курсами на основі їх описів та ключових слів. Підготовлені дані були використані для

подальшого навчання моделі, що дає можливість комбінувати текстові і колаборативні підходи для створення більш точних рекомендацій.

Для колаборативної фільтрації був застосований метод SVD++ (Singular Value Decomposition Plus), який є розширенням класичного методу SVD. SVD++ дозволяє виявляти латентні фактори взаємодій між користувачами та курсами, що значно покращує якість рекомендацій. Крім того, цей метод знижує вимірність матриці взаємодій, включаючи додаткові фактори, які враховують уподобання користувачів, що дозволяє моделі робити точніші прогнози рейтингу курсів. Програмну реалізацію методу було подано наступним чином:

```
# Створення матриці взаємодій користувач-курс
n_u, n_i = len(user_ids), len(course_ids)
R = np.zeros((n_u, n_i))  # Ініціалізація матриці взаємодій
for r in train.itertuples():
    R[user2idx[r.user_id], course2idx[r.course_id]] = 1.0  # Запис
взаємодії (1 для відгуку)

# Застосування SVD для зниження вимірності
svd = TruncatedSVD(n_components=128, random_state=args.seed)  #
Ініціалізація SVD з 128 латентними факторами
U  = svd.fit_transform(R)  # Матрична факторизація для користувачів
Vt = svd.components_.T  # Матрична факторизація для курсів
```

Поєднання collaborative filtering та content-based filtering дає змогу системі використовувати переваги обох підходів: перший орієнтується на взаємодії між користувачами та курсами, а другий – на семантичних характеристиках самих курсів. Це забезпечує більш точну генерацію рекомендацій, оскільки враховуються як історія взаємодій, так і контекст самого курсу.

Для реалізації цього процесу дані були підготовлені таким чином:

– використано бібліотеку scikit-learn для побудови TF-IDF матриці та SVD для колаборативної фільтрації, що дозволяє знижувати вимірність взаємодій користувачів з курсами;

– застосовано нормалізацію для всіх обчислень, щоб забезпечити рівномірність впливу різних підходів на загальний результат.

Програмна реалізація цього процесу подана нижче:

```
# Побудова TF-IDF матриці для описів курсів
```

```
tfidf_vectorizer = TfidfVectorizer(max_features=10000,
stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(df['description'])  #
Використання описів курсів

# Застосування SVD для зниження вимірності матриці
svd = TruncatedSVD(n_components=128, random_state=42)
tfidf_svd = svd.fit_transform(tfidf_matrix)

# Нормалізація SVD матриці
tfidf_svd_normalized = tfidf_svd / np.linalg.norm(tfidf_svd,
axis=1)[:, None]  # Нормалізація за рядками

# Збереження результату у вигляді розрідженої матриці
sparse_matrix = csr_matrix(tfidf_svd_normalized)
```

У процесі тренування також використовувалася weighting grid, що дозволяє комбінувати різні ваги для методів TF-IDF, SBERT, CF та popularity. Це дає можливість налаштувати модель для досягнення оптимального балансу між різними компонентами. Різні варіації ваг визначали, який підхід (наприклад, колаборативна фільтрація або контентна фільтрація) має більший вплив на рекомендації для конкретного користувача. Це було реалізовано наступним чином:

```
weight_grid = [
        (0.35, 0.35, 0.20, 0.10),  # "збалансований" гібрид
        (0.30, 0.30, 0.35, 0.05),  # CF-heavy
        (0.25, 0.25, 0.40, 0.10),  # CF-stronger
    ]
```

Для оцінки точності результатів після тренування використовувалися метрики precision, recall, MAP та nDCG. Ці метрики дозволяють визначити, наскільки ефективно модель надає рекомендації, а також чи є покращення в порівнянні з попередніми моделями. Для перевірки результатів використовувалася тестова вибірка, а саме 2350 унікальних користувачів на 52 тис. інтеракцій, що дозволяло оцінити, як модель працює на нових даних.

Останнім етапом було застосування методу tail-aware rerank, що дозволяє підвищити різноманітність рекомендацій для довгого хвоста курсів (менш популярних), одночасно зберігаючи баланс між популярними курсами та менш відомими, але релевантними для користувача. Це було важливо для забезпечення точних рекомендацій як для нових, так і для досвідчених користувачів.

Приклад запуску оцінки якості моделі при гібридному підході та пошуку найкращих «ваг» зображено на Рисунку 3.2.

Рисунок 3.2 – Підбір найкращих параметрів для гібридної моделі

Результатом оцінки (див. Рисунок 3.2) є значення метрик Precision@10, Recall@10, MAP@10 та nDCG@10.

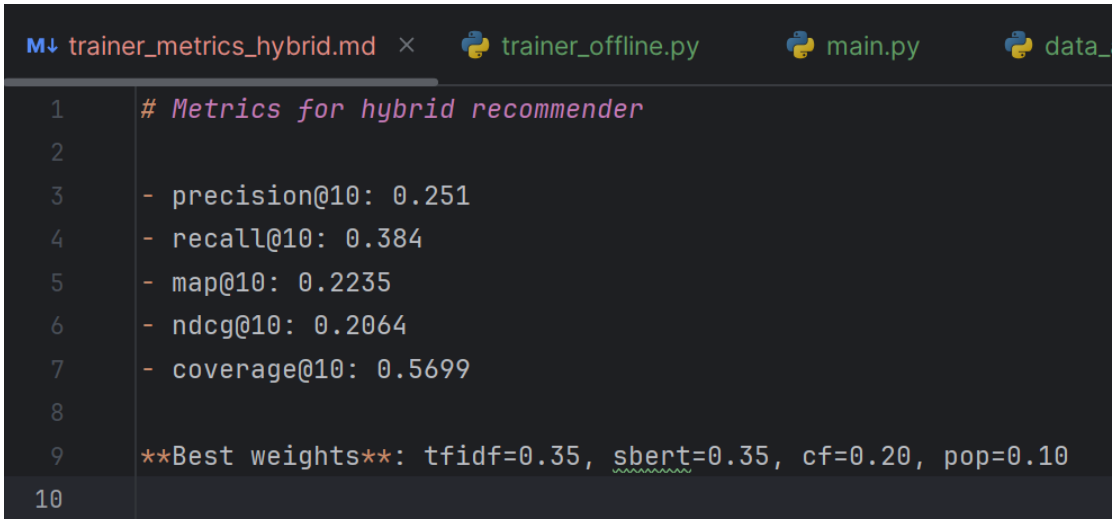


Рисунок 3.2 – Оцінка метрик гібридної моделі

Наведені результати навчання, в цілому є прийнятними, та навіть вище за середні в сфері навчальних платформ [X], що свідчить про якісне навчання моделі та високу якість синтетичних даних.

3.3.3 Генерація та виведення індивідуальних рекомендацій

Генерація та виведення індивідуальних рекомендацій реалізовано через REST-ендпоінт /recommendations, що забезпечує обробку запитів на отримання персоналізованих курсів. Цей ендпоінт приймає user_id та параметр k, що визначає кількість рекомендованих курсів. Після обробки запиту генерується список персоналізованих курсів, який повертається у форматі JSON.

Основна логіка для генерації рекомендацій описана в функції generate_recommendations() у файлі main.py. Функція відповідає за перевірку наявності користувача в базі даних, після чого за допомогою методу recommend() формується список рекомендованих курсів на основі історії взаємодій користувача з іншими курсами. Код функції:

```
@app.get("/recommendations", summary="Get personalized
recommendations")
def get_recommendations(
    user_id: str,
    k: int = K_DEFAULT,
    role: str = Depends(get_current_role),
) -> list[dict]:
    if not db.users.find_one({"user_id": user_id}):
        raise HTTPException(status_code=404, detail="User not
found")

    ids = recommend(user_id, k)
    results: list[dict] = []
    for cid in ids:
        course = db.courses.find_one({"course_id": cid})
        if course:
            course["_id"] = str(course.get("_id", ""))
            results.append(course)
    return results
```

Цей ендпоінт обробляє запит користувача та перевіряє наявність його профілю в базі даних. У разі наявності профілю, генеруються рекомендації на основі попередніх взаємодій користувача з курсами. Рекомендовані курси повертаються у вигляді списку, що містить їхні ідентифікатори, назви та описи.

Демонстрація інтерфейсу користувача для відображення персоналізованих рекомендацій зображена на Рисунку 3.3.

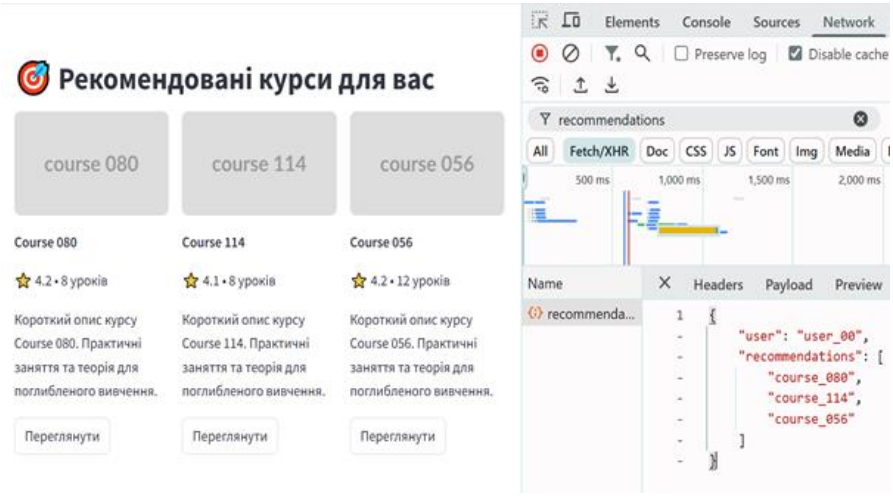


Рисунок 3.3 – Виведення персональних рекомендацій

Інтерфейс взаємодіє з бекендом через вищезазначений ендпоінт, що дозволяє користувачеві отримувати рекомендації у реальному часі. Таке рішення забезпечує зручний спосіб інтеграції даних і відображення результатів, а також створює основу для подальших вдосконалень, наприклад – таких як зворотний зв'язок від користувачів.

3.4 Технічні характеристики та завантаження на хостинг

Розроблений вебзастосунок має серверну частину, що реалізована за допомогою FastAPI, який відповідає за обробку запитів та взаємодію з базою даних MongoDB. Серверна частина також контейнеризована через Docker, що дозволяє запускати застосунок на будь-якій платформі з підтримкою контейнерів, що кратно збільшує доступність обслуговування.

Для зручності налаштування всіх компонентів використовувався Docker Compose, який автоматизує процес запуску серверної частини та забезпечує зручність масштабування.

Для забезпечення безперебійної роботи серверної частини, вона має такі технічні вимоги:

– мінімум 4 ГБ оперативної пам'яті;

– процесор з 4 та більше логічними ядрами для ефективної обробки даних;

– стабільне підключення до інтернету для роботи з базою даних MongoDB та забезпечення злагодженої взаємодії компонентів.

Клієнтська частина вебзастосунку вимагає стандартного веб-браузера для доступу до інтерфейсу, через який користувачі можуть отримувати персоналізовані рекомендації. Взаємодія між користувачем і сервером відбувається через API, що дозволяє отримувати та відображати дані, які генеруються серверною частиною.

Щодо процесу розміщення вебзастосунку, для тестування було обрано безкоштовний хостинг 000webhost. Процес включав реєстрацію на платформі, завантаження файлів через FTP та налаштування середовища для запуску застосунку. Для зручності подальшої масштабованості та автоматизації, контейнеризація дозволяє розгорнути вебзастосунок на будь-якому сервері або у хмарному середовищі.

3.5 Керівництво користувача

Інструкція з інтеграції та використання рекомендаторної системи залежить від ролі користувача в проєкті. Для рядового користувача, що буде взаємодіяти з уже інтегрованим модулем через вебсайт, інструкція зводиться лише до звичного перегляду та підбору курсів згідно до своїх вподобань та до специфіки системи.

Адміністратор, який хоче інтегрувати рекомендаторну систему на свій вебсайт, має на увазі кілька основних етапів: налаштування середовища, запуск модуля, налаштування API та взаємодія з командним рядком для генерації даних і управління системою. Для цього спершу потрібно налаштувати проект на сервері, використовуючи Docker або інше підходяще середовище, що дозволить

ізолювати виконання скриптів та забезпечити зручність у розгортанні. Після налаштування середовища, адміністратор може налаштувати API, що дозволить системі отримувати запити від клієнтської частини.

Завдяки FastAPI, адміністратор може керувати системою через інтерфейс API, отримувати статистику, налаштовувати модель та керувати користувацькими запитами безпосередньо через RESTful ендпоінти. Паралельно з API є можливість використовувати командний рядок для автоматизації різних задач, таких як генерація нових даних, оновлення бази курсів або запуск процесів тренування. Ключові ендпоінти та CLI-ключі для взаємодії з системою представлені наступними:

– /health, метод: GET, відповідає за перевірку стану сервісу;

– /courses: метод GET, отримання списку курсів, що доступні для перегляду та взаємодії;

– /rating: метод POST, надсилання оцінок для курсу, щоб покращити персоналізовані рекомендації;

– /log: метод POST, збереження логів взаємодій користувача з системою для подальшого аналізу;

– /auth/check: метод GET, перевірка наявності правильного токену LMS для доступу до сервісу;

– /recommendations: метод GET, отримання персоналізованих рекомендацій для користувача на основі його вподобань;

– /feedback: метод POST, подача відгуків про отримані рекомендації з можливістю оцінки їх корисності.

Важливим аспектом є налаштування прав доступу, які обмежують доступ до чутливих даних, таких як адміністративні ендпоінти:

– /admin/metrics: метод GET, доступ до метрик ефективності системи, щоб оцінити точність рекомендацій;

– /admin/retrain: метод POST, ініціація перенавчання моделі для оновлення та поліпшення її роботи.

Для налаштування та керування системою через командний рядок, адміністартори можуть використовувати CLI-параметри (ключі). Нижче наведено основні ключі CLI, що забезпечують гнучкість при інтеграції та використанні рекомендаторної системи, а також можливість власноруч зегенеруати тестові данні:

--courses: вказує шлях до файлу з курсами, який буде використовуватися для генерації рекомендацій. Цей ключ дозволяє системі працювати з конкретним датасетом курсів.

--out_inter: вказує шлях до файлу, куди будуть збережені синтетичні взаємодії між користувачами та курсами. Цей ключ необхідний для збереження результатів генерації даних для подальшого тренування моделі.

--out_users: визначає шлях до файлу, де будуть збережені дані про користувачів для сценаріїв cold-start, коли у користувача немає історії взаємодій.

--n_users: вказує кількість користувачів для генерації синтетичних даних. Це дозволяє контролювати обсяг тестових даних що будуть використувавтись для тренування рекомендаційної моделі.

--n_cold: вказує кількість cold-start користувачів, яких потрібно згенерувати для оцінки роботи моделі в умовах відсутності попередніх взаємодій.

--n_inter: задає загальну кількість взаємодій, яку система повинна генерувати. Використовується для контролю обсягу даних для тренування.

--neg_rate: вказує частку негативних взаємодій (наприклад, відгуків з оцінкою 1), які будуть додані до даних для тренування. Цей параметр допомагає створити більш реалістичні умови для навчання моделі.

--seed: задає початкове значення для генерації випадкових чисел, що забезпечує відтворюваність результатів при повторному запуску тієї ж конфігурації.

--min_per_course: визначає мінімальну кількість взаємодій для кожного курсу. Це необхідно для забезпечення рівномірного покриття курсів у датасеті.

3.6 Тестування та оцінювання якості системи

З метою проведення верифікації та валідації розробленої рекомендаторної системи, її було протестовано з використанням методів, що відповідають вимогам до якості та ефективності системи. Тестування було спрямоване на оцінку якості обробки даних, перевірку коректності роботи окремих компонентів системи, а також визначення рівня точності та ефективності рекомендацій. Для цього було розроблено набір тестових сценаріїв, що включає перевірку на помилки в даних, наявність дублікатів, правильність роботи алгоритмів та перевірку на відповідність заданим меткам якості. Усі тестування проводились безпосередньо на вихідних даних, що містять інформацію про курси та взаємодії користувачів, а також на оброблених даних, включаючи генерацію описів, ключових слів та ембеддингів для кожного курсу.

Основним методом тестування, що був застосований, є unit-тестування, яке охоплює перевірку коректності окремих компонентів системи [37]. Це включає перевірку правильності генерації даних, обчислення схожості між курсами, обробки ембеддингів, а також перевірку генерації рекомендацій на основі різних підходів (content-based, collaborative filtering).

Для цього були використані методи, що забезпечують перевірку базових функцій, таких як перевірка правильності розмірів ембеддингів, перевірка відсутності нульових векторів у даних, а також аналіз коректності генерації синтетичних взаємодій користувачів із курсами. Тестування виконувалось за допомогою Python-бібліотек, зокрема unittest для автоматизації перевірок.

Для забезпечення стабільності системи було розроблено низку юніт-тестів, які перевіряють ключові компоненти моделі. Тести на перевірку коректності ембеддінгів (розмір векторів, наявність нульових значень, правильність формування) включають такі функції, як:

```
import unittest
import numpy as np
from interaction_generator import generate_interactions
```

```
class TestInteractionGenerator(unittest.TestCase):

    def test_generate_interactions(self):
        result = generate_interactions(n_users=10, n_courses=5)
        self.assertEqual(result.shape, (10, 5), "Невірний розмір
матриці взаємодій")

    def test_sbert_embeddings(self):
        test_embedding = np.random.rand(384)  # тестовий вектор
        self.assertEqual(test_embedding.shape[0], 384, "Невірний
розмір ембеддінгу")
```

Ці тести перевіряють правильність розмірів створюваних матриць взаємодій і ембеддінгів, а також виконуються на синтетичних даних, що моделюють поведінку користувачів. валідація даних здійснюється через аналіз коректності згенерованих ембеддінгів та взаємодій між користувачами та курсами. Для цього використовується обчислення косинусного схожості між ембеддінгами курсів, що дозволяє верифікувати їх релевантність і відповідність до предметної категорії курсу.

Приклад верифікації через обчислення косинусного схожості між курсами:

```
def test_cosine_similarity():
    # Тестуємо, що схожість між двома курсами не менша за threshold
    sim = cosine_similarity(course1_embedding, course2_embedding)
    self.assertGreater(sim, 0.8, "Cosine similarity між курсами
занадто мала")
```

Ці тести перевіряють, чи виконуються умови порогового значення схожості між курсами, що є важливим елементом перевірки якості рекомендацій.

Аналіз результатів тестування був проведений з використанням метрик, таких як precision, recall, mean average precision (MAP) та normalized discounted cumulative gain (NDCG) [39], які дозволяють оцінити ефективність рекомендаційної системи. Оцінка якості рекомендацій проводилась на основі тестових даних, що містили реальні та синтетичні взаємодії користувачів з курсами [38]. Окрім того, результати були підтверджені через побудову відповідних графіків і діаграм, що дозволяють візуально оцінити розподіли та співвідношення різних параметрів.

Розподіл за категоріями курсів показує, як взаємодії користувачів розподіляються серед різних категорій курсів. Відображає цей розподіл візуально Рисунок 3.4, що допомагає визначити, чи є в системі категорії, які мають надмірну популярність або ж недостатньо представлені.

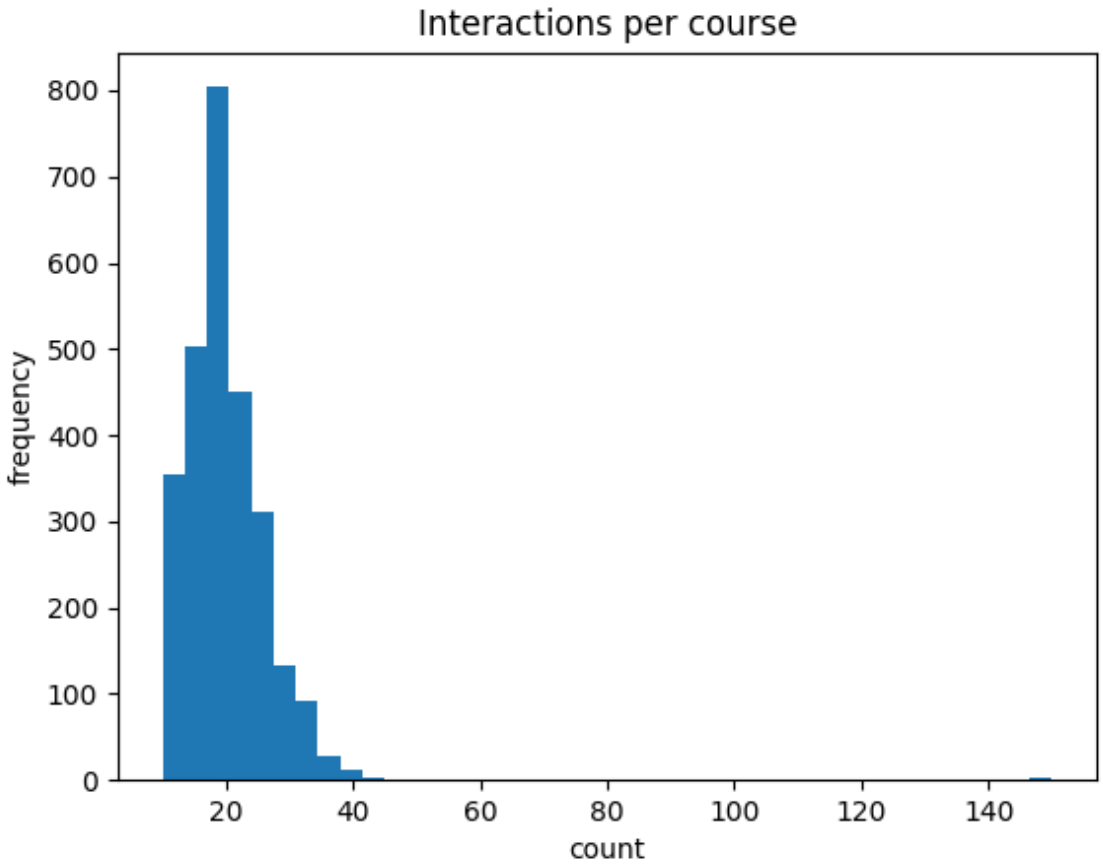


Рисунок 3.4 – Розподіл взаємодій по категоріях курсів.

Це дозволяє побачити, які категорії користуються найбільшим попитом серед користувачів, а також оцінити рівень рівномірності розподілу. Як показав аналіз, більшість курсів має відносно низьку кількість взаємодій, в той час як кілька курсів мають значно більше взаємодій, що вказує на типове представлення довгого хвоста в розподілі кількості взаємодій, де більшість курсів мають малу кількість взаємодій, але є декілька курсів, які значно перевищують цей поріг, вказуючи на їх високу популярність серед користувачів.

Далі, розподіл за кількістю взаємодій на користувача надає цінну інформацію про рівень залученості користувачів у систему. Це дозволяє оцінити, чи всі користувачі активно взаємодіють з платформою, чи є багато неактивних

або малозалучених користувачів. Рисунок 3.5 показує, як варіюється кількість взаємодій у кожного користувача, що допомагає виявити потенційні проблеми в рівномірності залучення.

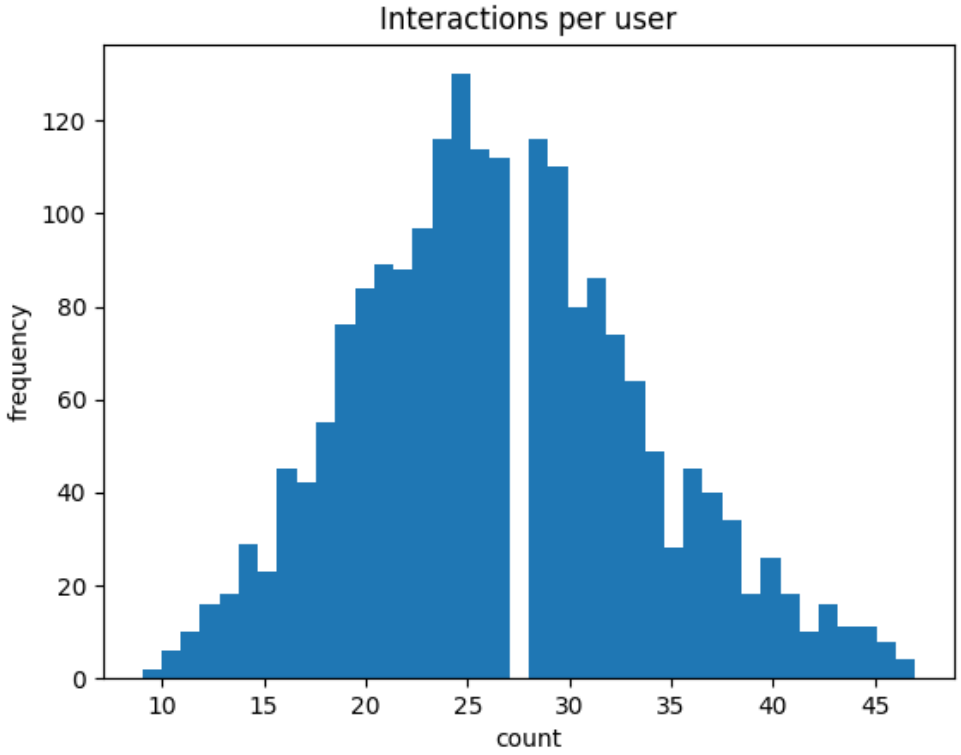


Рисунок 3.5 – Розподіл кількості взаємодій на одного користувача.

З рисунка видно, що кількість взаємодій на користувача розподілена майже симетрично, з явними піками в середній частині графіка. Це вказує на те, що більшість користувачів взаємодіють з платформою в межах 25-30 разів, в той час як менш активні користувачі мають значно менше взаємодій, а дуже активні — значно більше. Така розподільна схема є характерною для реальних платформ, де більшість користувачів має середню активність, а залишок розподіляється між рідко та часто взаємодіючими користувачами.

Останнім важливим аспектом є аналіз довгого хвоста (long-tail), що дозволяє оцінити, як система працює з рідко відвідуваними курсами. Такий аналіз показує, чи здатна система надавати релевантні рекомендації для менш популярних курсів, що є ключовим для створення більш різноманітних і

персоналізованих рекомендацій. Рисунок 3.6 відображає розподіл курсових взаємодій між популярними та менш популярними курсами, що дозволяє оцінити, як платформа працює з малопопулярними напрямками навчання.

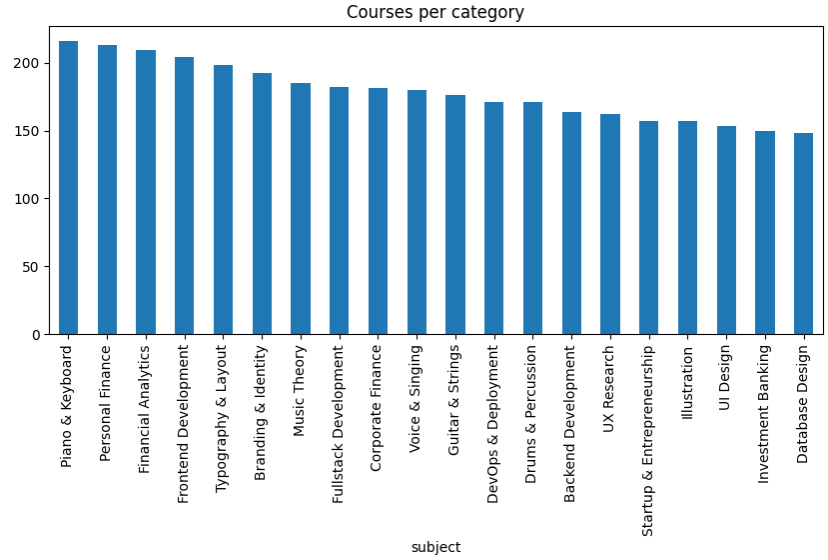


Рисунок 3.6 – Розподіл курсових взаємодій у «довгому хвості»

Графік демонструє частку взаємодій, що припадають на менш популярні курси, допомагаючи виявити можливості для покращення різноманітності рекомендацій і рівномірності покриття різних категорій на платформі.

Після аналізу результатів тестування та верифікації було виявлено кілька областей для покращення системи:

– оптимізація метрик моделі: зокрема, покращення ваг для гібридної моделі (content-based + collaborative filtering + SBERT);

– зменшення шуму в даних: за допомогою точнішої фільтрації негативних прикладів та кращої генерації синтетичних даних;

– збільшення покриття рідко відвідуваних категорій: для більш рівномірного покриття всіх курсів, включаючи менш популярні категорії.

У результаті тестування рекомендаторної системи було проведено перевірку її основних компонентів, зокрема генерації ембеддінгів та обчислення схожості між курсами [40]. Зроблені висновки дозволяють спрямувати подальші удосконалення на оптимізацію моделі та зменшення шуму в даних.

3.7 Висновки. Практичне втілення та його результати

Була реалізована програмна частина вебзастосунку для рекомендаційної системи. Для забезпечення стабільності, ефективності та масштабованості було обрано відповідні технології. Розробка здійснювалася у середовищі PyCharm з використанням Python 3.10, що забезпечувало сумісність з бібліотеками. Для ізоляції середовища та покращення переносимості використовувався Docker з Docker Compose. Як база даних обрана MongoDB, що дозволяє ефективно працювати з неструктурованими даними.

Розробка бази даних передбачала створення колекцій для курсів, користувачів, взаємодій та логів, що сприяло формуванню персоналізованих рекомендацій. Інтеграція з FastAPI дозволила створити RESTful API для взаємодії з даними в реальному часі. Підготовка даних включала автоматичну генерацію описів та ключових слів для курсів, а також класифікацію курсів на підкатегорії, що покращило точність рекомендацій.

Для навчання моделі було застосовано гібридний підхід, що поєднує content-based, collaborative filtering та popularity-based filtering. Використання методів SVD++ та SBERT дозволило значно підвищити точність рекомендацій, комбінуючи текстові характеристики курсів та взаємодії користувачів. Оптимізація ваг між різними методами фільтрації сприяла покращенню ефективності моделі.

Тестування системи включало оцінку ефективності за допомогою метрик precision, recall, MAP та nDCG. Були також проведені юніт-тести для перевірки коректності компонентів, таких як генерація ембеддінгів та схожість між курсами. Результати тестування показали задовільні показники ефективності, а також виявили можливості для подальшого вдосконалення моделі, зокрема в частині оптимізації метрик та зменшення шуму в даних.

ВИСНОВКИ

У процесі розробки дипломного проєкту була створена та протестована рекомендаторна система для онлайн-освітньої платформи, яка поєднує методи контентної фільтрації, колаборативної фільтрації та фільтрації на основі популярності. Під час роботи було здійснено всебічне вивчення предметної області, виконано аналіз існуючих аналогів і визначено їх сильні та слабкі сторони, що стало основою для адаптації цих підходів у розробленій системі. Важливим етапом стало формулювання чіткої концептуальної основи для проєкту, включаючи функціональні та нефункціональні вимоги, визначення ролей користувачів та побудову UML-діаграм, що відображають основні процеси взаємодії користувачів з платформою.

Завдяки використанню сучасних технологій, таких як FastAPI, MongoDB, Docker, а також методології SCRUM, було розроблено ефективне середовище для роботи з даними та обробки запитів. Під час тестування системи застосовувалися різноманітні метрики, зокрема Precision, Recall, MAP, NDCG, що дозволили оцінити її ефективність та точність. Тестування підтвердило високу точність рекомендацій та ефективність роботи системи з великими обсягами даних. Окремо варто відзначити, що гібридний підхід, поєднуючи різні методи фільтрації, забезпечує більш точні та релевантні рекомендації, що робить систему надійним інструментом для персоналізованого навчання.

Отримані результати підтвердили правильність вибору методів та технологій, що забезпечують гнучкість і масштабованість системи. Удосконалення моделі, оптимізація ваг для різних компонентів та подальше зменшення шуму в даних дозволять досягнути ще більших результатів у покращенні точності рекомендацій. Розроблена система має великий потенціал для подальшого розвитку, здатна адаптуватися до нових вимог та умов, що забезпечує її ефективність у реальних застосунках.

Поставлені цілі дипломного проєкту були досягнуті повністю, а виконана робота відповідає затвердженій програмі розробки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бедратюк Л. П., Радельчук Г. І. Методичні настанови для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 121 «Інженерія програмного забезпечення»: кваліфікаційна робота / Л. П. Бедратюк, Г. І. Радельчук. – Хмельницький: ХНУ, 2023. – 60 с.

2. Терлецька Т. С. «Аналіз використання масових відкритих онлайн-курсів в освітньому процесі університету (на прикладі спеціальності „Дошкільна освіта“)» // Open Educational E-Environment of Modern University. – 2020. – № 8. – С. 136–146. – DOI: 10.28925/2414-0325.2020.8.12. – [Електронний ресурс]. – Режим доступу: https://openedu.kubg.edu.ua/journal/index.php/openedu/article/view/288 (дата звернення: 30.04.2025).

3. Жабер А. Х. «Метод побудови інтелектуальної системи рекомендацій для професійної орієнтації» // Оптико-електронні інформаційно-енергетичні технології. – 2023. – Т. 46, вип. 2. – С. 22–36. – DOI: 10.31649/1681-7893-2023-46-2-22-36. – [Електронний ресурс]. – Режим доступу: https://oeipt.vntu.edu.ua/index.php/oeipt/article/view/659 (дата звернення: 30.04.2025).

4. Форкун Ю., Мартинюк В., Яшина О. Метод розробки та проектування архітектурної складової програмного застосунку // Measuring and Computing Devices in Technological Processes. – 2023. – № 1. – С. 87–93. – [Електронний ресурс]. – Режим доступу: https://evntpu.kpi.ua/article/view/281299 (дата звернення: 03.06.2025).

5. Запорожцева Ю. С. «Персоналізація в навчанні як сучасний освітній тренд» // Педагогічна Житомирщина. – 2024. – № 4 (36). – [Електронний ресурс]. – Режим доступу: https://imso.zippo.net.ua/wp-content/uploads/2024/12/3.%D0%97%D0%B0%D0%BF%D0%BE%D1%80%D0%BE%D0%B6%D1%86%D0%B5%D0%B2%D0%B0.pdf (дата звернення: 30.04.2025).

6. Ricci F., Rokach L., Shapira B. (ред.) Recommender Systems Handbook. – 4-е вид. – Cham : Springer, 2022. – 1013 с.

7. Awan J., Ul-Haq S., Jhanjhi N. et al. «A review of deep-learning-based recommender systems in e-learning environments» // Artificial Intelligence Review. – 2022. – DOI: 10.1007/s10462-022-10135-2. – [Електронний ресурс]. – Режим доступу: https://link.springer.com/article/10.1007/s10462-022-10135-2 (дата звернення: 02.05.2025).

8. Talaghzi J., Bellafkih M., Bennane A. et al. «A Combined E-Learning Course Recommender System» // International Journal of Emerging Technologies in Learning. – 2023. – Vol. 18, No 6. – С. 53–70. – DOI: 10.3991/ijet.v18i06.36987. – [Електронний ресурс]. – Режим доступу: https://online-journals.org/index.php/i-jet/article/view/36987 (дата звернення: 30.04.2025).

9. Гавриленко Я., Форкун Ю. Аналіз методологій побудови рекомендаційних систем цільового просування інформаційного контенту // Інформація, комунікація, суспільство – 2020: матеріали 9-ї Міжнародної науково-практичної конференції. – Хмельницький: ХНУ, 2020. – С. 115–118. – [Електронний ресурс]. – Режим доступу: https://conf.ztu.edu.ua/wp-content/uploads/2020/05/115.pdf (дата звернення: 03.06.2025).

10. Протоковський А., Форкун Ю. Аналіз стратегій рекомендаційних систем знайомств людей та груп за інтересами у віртуальних спільнотах // Інформація, комунікація, суспільство – 2020: матеріали 9-ї Міжнародної науково-практичної конференції. – Хмельницький: ХНУ, 2020. – С. 119–122. – [Електронний ресурс]. – Режим доступу: https://conf.ztu.edu.ua/wp-content/uploads/2020/05/119.pdf (дата звернення: 03.06.2025).

11. Irwan I., Desnelita Y., Susanti W. та ін. The Intelligent Model Collaborative Project Based Learning Uses Moodle to Improve Face-to-Face Online in the Covid-19 Period // Proc. 3rd Int. Conf. on Education and Science (ICES 2021). – 2022. – [Електронний ресурс]. – Режим доступу: https://eudl.eu/doi/10.4108/eai.17-11-2021.2318651 (дата звернення: 02.05.2025).

12. Shareef M., Jose B. R., Mathew J. et al. «Advancing explainable MOOC recommendation systems: a morphological operations-based framework on partially ordered neutrosophic fuzzy hypergraphs» // Artificial Intelligence Review. – 2025. – Vol. 58. – Art. 46. – DOI: 10.1007/s10462-024-11018-4. – [Електронний ресурс]. – Режим доступу: https://link.springer.com/article/10.1007/s10462-024-11018-4 (дата звернення: 30.04.2025).

13. Шелевер О. В., Капітан Л. І., Коновалов О. Ю. «Адаптивне навчання здобувачів за допомогою сучасних цифрових платформ» // Інноваційна педагогіка. – 2024. – Вип. 75. – С. 269–272. – [Електронний ресурс]. – Режим доступу: http://www.innovpedagogy.od.ua/archives/2024/75/54.pdf (дата звернення: 30.04.2025).

14. ISO/IEC 25010:2023 «Systems and software quality models». – Geneva : ISO, 2023. – 32 с.

15. López-Pernas S., Saqr M., Conde J., Del-Río-Carazo L. A Broad Collection of Datasets for Educational Research Training and Application // Learning Analytics Methods and Tutorials. – 2024. – С. 17–66. – DOI: 10.1007/978-3-031-54464-4_2. – [Електронний ресурс]. – Режим доступу: https://link.springer.com/chapter/10.1007/978-3-031-54464-4_2 (дата звернення: 04.05.2025).

16. Akhuseyinoglu K., Brusilovsky P. Exploring Behavioral Patterns for Data-Driven Modeling of Learners' Individual Differences // Frontiers in Artificial Intelligence. – 2022. – Vol. 5, Art. 807320. – DOI: 10.3389/frai.2022.807320. – [Електронний ресурс]. – Режим доступу: https://doi.org/10.3389/frai.2022.807320 (дата звернення: 06.05.2025).

17. Федула М., Кльоц Ю., Форкун Ю. Проектування сенсорних людино-машинних інтерфейсів з фільтрацією механічних коливань. – Хмельницький: ХНУ, 2020. – [Електронний ресурс]. – Режим доступу: https://ir.lib.hneu.edu.ua/bitstream/123456789/24834/1/fedula2020.pdf (дата звернення: 03.06.2025).

18. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 «General Data Protection Regulation». – Official Journal of the EU, L 119, 04.05.2016. – [Електронний ресурс]. – Режим доступу: https://eur-lex.europa.eu/eli/reg/2016/679 (дата звернення: 02.05.2025).

19. OWASP Foundation. «OWASP Application Security Verification Standard v 4.0.3». – 2023. – [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-application-security-verification-standard/ (дата звернення: 02.05.2025).

20. IBM. Use-case models. – 2021. – [Електронний ресурс]. – Режим доступу: https://www.ibm.com/docs/en/dma?topic=approaches-use-case-models (дата звернення: 03.06.2025).

21. Каграманова Ю. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти. – 2022. – [Електронний ресурс]. – Режим доступу: https://dou.ua/forums/topic/40575/ (дата звернення: 03.06.2025).

22. Вільчавський І. Бізнес-аналітики і нефункціональні вимоги в agile розробці. – 2023. – [Електронний ресурс]. – Режим доступу: https://www.ba.in.ua/2023/08/18/biznes-analityky-i-nefunkczionalni-vymogy-v-agile-rozrobczi/ (дата звернення: 03.06.2025).

23. Docker Inc. «Docker Compose: Best Practices». – 2023. – [Електронний ресурс]. – Режим доступу: https://docs.docker.com/compose/compose-file/ (дата звернення: 02.05.2025).

24. GeeksforGeeks. REST API Architectural Constraints. – 2025. – [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/rest-api-architectural-constraints/ (дата звернення: 03.06.2025).

25. MongoDB Inc. «Document Schema Design Patterns». – White Paper, 2024. – [Електронний ресурс]. – Режим доступу: https://www.mongodb.com/white-papers/schema-design-patterns (дата звернення: 02.05.2025).

26. Форкун Ю., Форкун І., Яшина О., Праворська Н. Архітектурні методи оптимізації швидкодії та відмовостійкості програмних застосунків // Measuring

and Computing Devices in Technological Processes. – 2023. – № 1. – С. 196–201. – [Електронний ресурс]. – Режим доступу: https://evntpu.kpi.ua/article/view/281321 (дата звернення: 03.06.2025).

27. Ma Y., Ouyang R., Long X. et al. «DORIS: Personalized course recommendation system based on deep learning» // PLOS ONE. – 2023. – Vol. 18, Iss. 6. – e0284687. – DOI: 10.1371/journal.pone.0284687. – [Електронний ресурс]. – Режим доступу: https://pmc.ncbi.nlm.nih.gov/articles/PMC10237496/ (дата звернення: 30.04.2025).

28. Murel J., Kavlakoglu E. What is collaborative filtering? — 2024. — [Електронний ресурс]. — Режим доступу: https://www.ibm.com/think/topics/collaborative-filtering (дата звернення: 03.06.2025).

29. Netflix Technology Blog. «Recommending for Long-Term Member Satisfaction at Netflix». – 2020. – [Електронний ресурс]. – Режим доступу: https://netflixtechblog.com/recommending-for-long-term-member-satisfaction-at-netflix-5a2269f5d84 (дата звернення: 02.05.2025).

30. Microsoft Learn. «Deploy a FastAPI web app as a container in Azure App Service». – 2025. – [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/azure/app-service/fastapi-containers (дата звернення: 02.05.2025)

31. Krysik A. How to Build a Recommendation System: Explained Step by Step. — 2023. — [Електронний ресурс]. — Режим доступу: https://stratoflow.com/how-to-build-recommendation-system/ (дата звернення: 03.06.2025).

32. DuVander A. API Design Patterns for REST. — 2021. — [Електронний ресурс]. — Режим доступу: https://blog.stoplight.io/api-design-patterns-for-rest-web-services (дата звернення: 03.06.2025).

33. Ricci F., Rokach L., Shapira B. (ред.) Recommender Systems Handbook. – 4-е вид. – Cham : Springer, 2022. – 1013 с.

34. Vanezi E., Kouzapas D., Kapitsaki G. M. et al. «GDPR Compliance in the Design of the INFORM e-Learning Platform: a Case Study» // Proc. 13th Int. Conf. on Research Challenges in Information Science (RCIS 2019). – Brussels : IEEE, 2019. – P. 1–12. – DOI: 10.1109/RCIS.2019.8877022. – [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/336728551 (дата звернення: 30.04.2025).

35. Halliday P., Gorton C. How To Use Axios with React. – 2021. – [Електронний ресурс]. – Режим доступу: https://www.digitalocean.com/community/tutorials/react-axios-react (дата звернення: 03.06.2025).

36. GeeksforGeeks. Introduction to Tailwind CSS. – 2024. – [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/introduction-to-tailwind-css/ (дата звернення: 03.06.2025).

37. Krekel H. pytest Documentation (Release 8.4). – 2025. – [Електронний ресурс]. – Режим доступу: https://docs.pytest.org/en/stable/ (дата звернення: 03.06.2025).

38. Provalov V., Stavinova E., Chunaev P. SynEvaRec: A Framework for Evaluating Recommender Systems on Synthetic Data Classes. – 2021. – [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/358008049_SynEvaRec_A_Framework_for_Evaluating_Recommender_Systems_on_Synthetic_Data_Classes (дата звернення: 03.06.2025).

39. Evidently AI Team. 10 metrics to evaluate recommender and ranking systems. – 2024. – [Електронний ресурс]. – Режим доступу: https://www.evidentlyai.com/ranking-metrics/evaluating-recommender-systems (дата звернення: 03.06.2025).

40. Huilgol P. Top 4 Sentence Embedding Techniques using Python! – 2020. – [Електронний ресурс]. – Режим доступу: https://www.analyticsvidhya.com/blog/2020/08/top-4-sentence-embedding-techniques-using-python/ (дата звернення: 03.06.2025).

ДОДАТОК А

(обов’язковий)

ГРАФІЧНІ МАТЕРІАЛИ

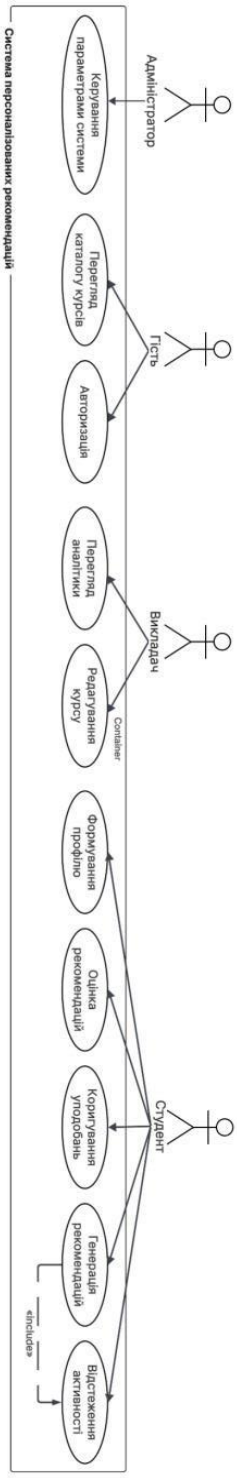


Рисунок А.1 – Діаграма варіантів використання

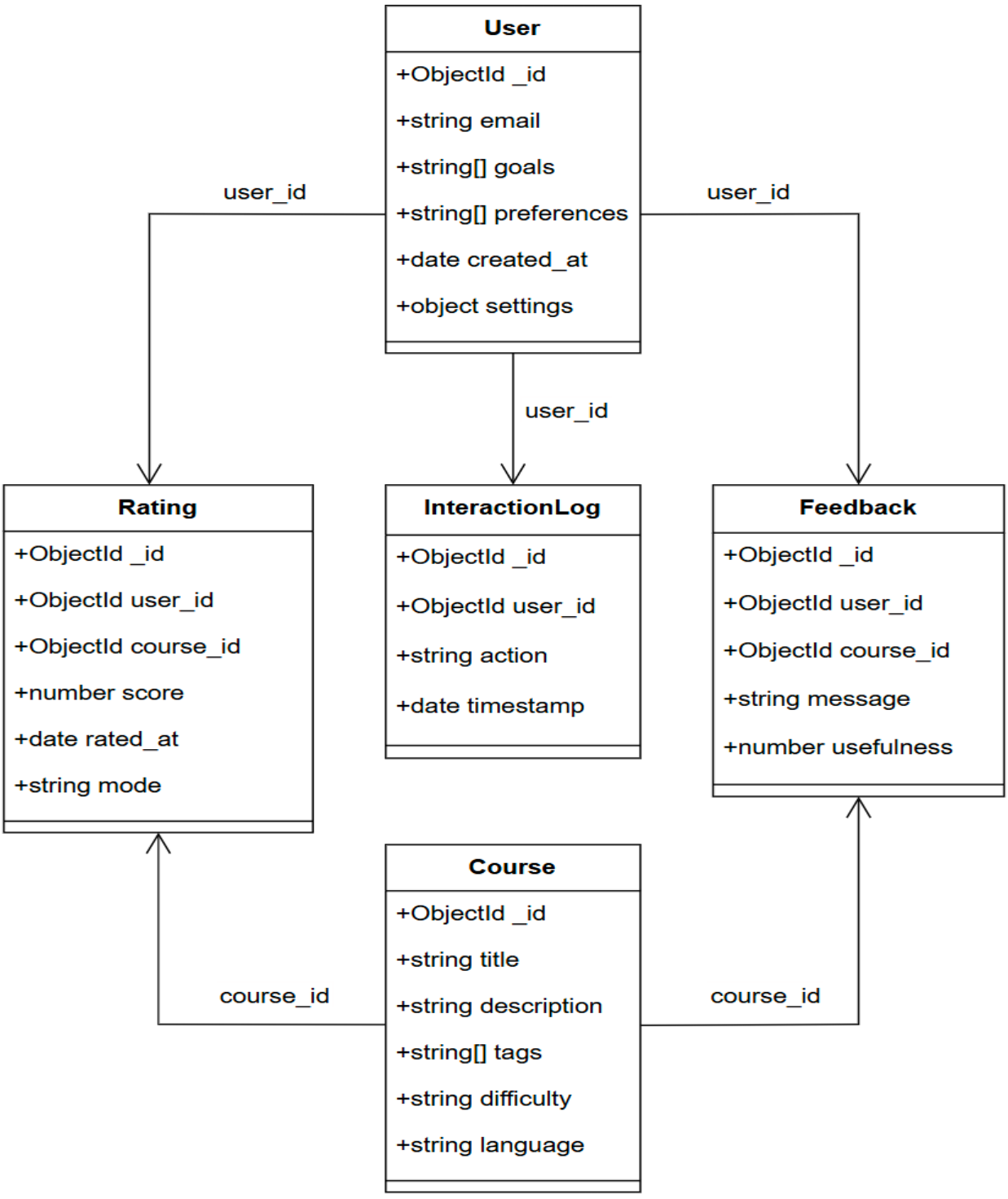


Рисунок А.2 – Інофрмаційна модель системи

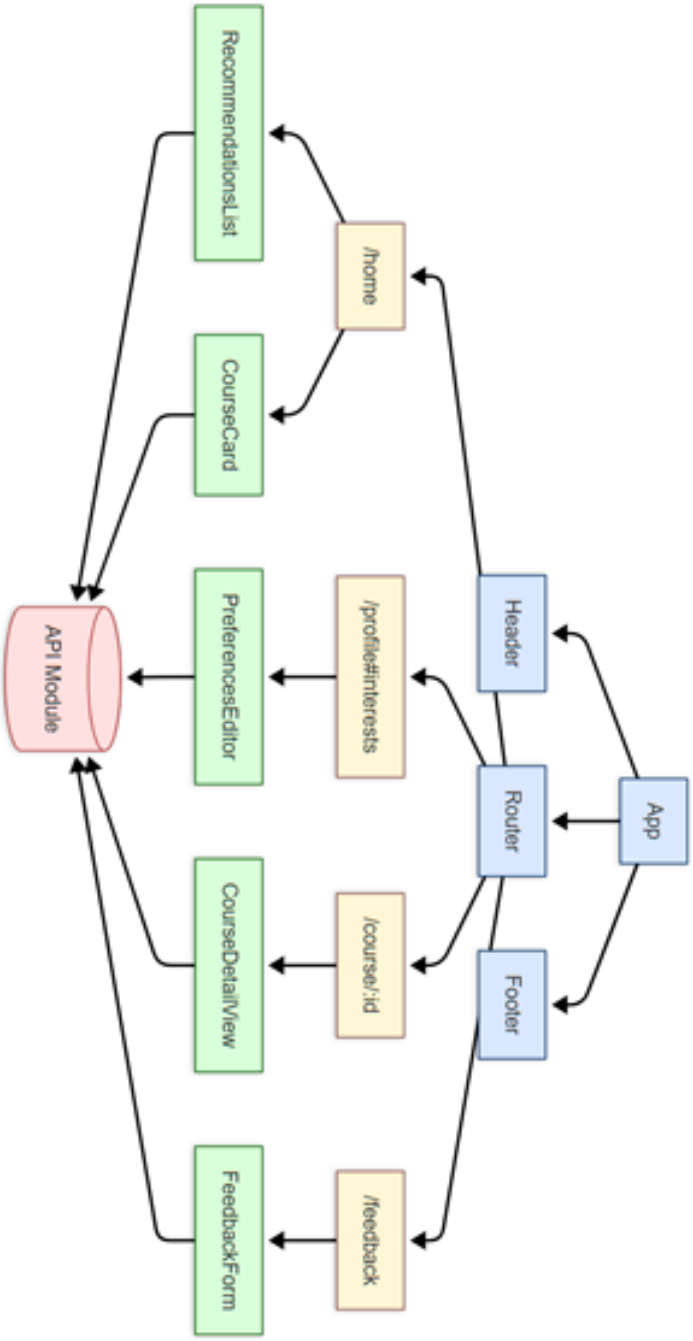


Рисунок А.3 – Структурна діаграма системи

ДОДАТОК Б

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Рисунок Б1 – Слайд 1

Рисунок Б.2 – Слайд 2

Рисунок Б.3 – Слайд 3

Рисунок Б.4 – Слайд 4

АНАЛІЗ НАЯВНОГО ПРОГРАМНО-ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ 5

На ринку онлайн-освіти існує низка платформ з елементами персоналізації. Проте більшість із них не адаптовані під локальні потреби або не розкривають повного потенціалу рекомендаційних систем.

ПЛАТФОРМА	ЛОКАЛІЗАЦІЯ ІНТЕРФЕЙСУ	ДОСТУПНІСТЬ API	ПІДТРИМКА COLD-START	СКЛАДНІСТЬ ІНТЕГРАЦІЇ
Coursera	Лише англійська та декілька мов	GraphQL API, обмежений доступ	Часткова - лише базові рекомендації для новачків	Висока: закрите ядро, обмежена кастомізація
Udemy	Англійська, часткова локалізація через браузер	Відкритий REST API для авторів	Підтримує content-based старт для нових користувачів	Середня: потрібна авторизація, документація присутня
Stepik	Повна підтримка української	Повноцінний REST API з документацією	Так, є вбудована cold-start логіка через теги та теми	Низька: підтримка сторонніх рішень, відкритий код
Prometheus	Українська локалізація за замовчуванням	Відсутній – API офіційно не документований	Немає підтримки – рекомендації відсутні як клас	Висока: інтеграція тільки через iframe

Жодна з аналізованих платформ не поєднує відкритість API, підтримку cold-start ситуацій та повноцінну локалізацію під українськомовного користувача. Розроблювана система спрямована на усунення цих обмежень через гнучку архітектуру та алгоритмічну адаптивність.

Рисунок Б.5 – Слайд 5

ВИЗНАЧЕННЯ ФУНКЦІОНАЛЬНИХ ТА НЕФУНКЦІОНАЛЬНИХ ВИМОГ ДО ПЗ

6

НЕФУНКЦІОНАЛЬНІ ВИМОГИ

- **Час відповіді API** – не більше 300 мс для більшості запитів користувача
- **Горизонтальна масштабованість** системи для обробки зростаючої кількості користувачів
- **Захист персональних даних** відповідно до GDPR та локального законодавства
- **Висока доступність** – мінімум 99,5 % часу роботи сервісу він повинен бути доступний користувачу

ФУНКЦІОНАЛЬНІ ВИМОГИ

- **Формування рекомендацій** для зареєстрованого користувача за допомогою гібридного алгоритму
- **Оновлення моделі** з урахуванням нових даних (вподобання, перегляди, обрані курси)
- **REST API /recommendations** – надання топ-N результатів на запит
- Можливість **інтеграції** у зовнішню онлайн-платформу через авторизований запит
- **Управління користувачем** (реєстрація, автентифікація, налаштування вподобань
- **Збір фідбеку** щодо запропонованих курсів (лайк/дізлайк, рейтинг)

Рисунок Б.6 – Слайд 6

ВИБІР ТИПУ АРХІТЕКТУРИ ТА ШАБЛОНІВ ПРОЄКТУВАННЯ 7

Тип архітектури

Клієнт-серверна архітектура реалізована у вигляді класичної трирівневої моделі. Рівень представлення забезпечується інтерфейсом на базі React або Streamlit, логіка системи реалізована за допомогою FastAPI, а зберігання даних – у MongoDB. Такий підхід дозволяє досягти чіткого розподілу обов'язків і гнучкості у масштабуванні.

Використані шаблони

Обрана архітектура надає низку ключових переваг. Вона дозволяє легко оновлювати окремі компоненти, адаптувати систему до зовнішніх платформ через REST API, а також спрощує впровадження CI/CD і контейнеризації (Docker, GitHub Actions).

Шаблони проєктування

У системі використано **шаблони проєктування**, які підвищують її гнучкість та придатність до розширення. **Repository** забезпечує абстракцію над запитами до бази даних, **Factory** спрощує створення об'єктів і конфігурацій моделей, а **Dependency Injection** полегшує тестування та заміну компонентів.

Рисунок Б.7 – Слайд 7

ОПИС ДЕКОМПОЗИЦІЇ, ЗАЛЕЖНОСТЕЙ, ІНТЕРФЕЙСІВ 8

СИСТЕМА СКЛАДАЄТЬСЯ З П'ЯТИ ОСНОВНИХ КОМПОНЕНТІВ:

Інтерфейс користувача (React / Streamlit) забезпечує перегляд рекомендацій і надсилання фідбеку.

REST API (FastAPI) обробляє запити, перевіряє авторизацію й передає їх далі.

Рекомендаційний модуль генерує персоналізовані курси, використовуючи дані з бази.

Trainer оновлює модель у фоновому режимі.

MongoDB зберігає профілі та історію, а **Redis** кешує результати.

Комунікація між модулями здійснюється за допомогою REST-запитів у форматі JSON.Авторизація реалізована на основі токенів JWT, що дозволяє захищати доступ до персоналізованого функціоналу.

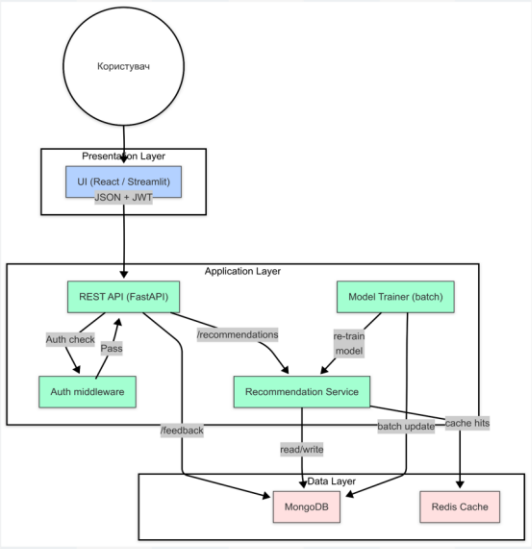


Рисунок Б.8 – Слайд 8

ПРОЄКТУВАННЯ МОДУЛІВ І ДАНИХ

9

1 **RECOMMENDATION MODULE**

Формує топ-N курсів на основі гібридної моделі, комбінуючи інтереси та поведінку користувача.

2 **MODEL TRAINER**

З певною періодичністю оновлює рекомендаційну модель на основі фідбеку, збереженого в БД.

3 **FEEDBACK COLLECTOR**

Приймає оцінки курсів від користувачів (рейтинг), зберігає його для подальшого аналізу.

4 **DATA STORAGE**

Зберігає всі наявні курси, профілі користувачів, історію їх навчання, сесії рекомендацій.

ОПИС ОСНОВНИХ ІНФОРМАЦІЙНИХ СУТНОСТЕЙ СИСТЕМИ

СУТНІСТЬ	КЛЮЧОВІ ПОЛЯ
User	user_id, мова, рівень, вподобання, історія
Course	course_id, назва, теми, тривалість, складність
Feedback	user_id, course_id, оцінка, час
RecommendationSession	user_id, список результатів, час генерації

Рисунок Б.9 – Слайд 9

АНАЛІЗ ТА ВИБІР ТЕХНОЛОГІЙ

10

ОСНОВНІ КОМПОНЕНТИ ТА ОБРАНІ ТЕХНОЛОГІЇ

КОМПОНЕНТ	ТЕХНОЛОГІЯ	АРГУМЕНТАЦІЯ ВИБОРУ
Мова розробки	Python	Велика ML-екосистема, підтримка FastAPI, сумісність з обраними бібліотеками
REST API	FastAPI	Висока продуктивність, OpenAPI, асинхронність
База даних	MongoDB	Гнучка схема, підтримка вкладених документів
Модель рекомендацій	LightFM / custom	Гібридна модель, підтримка content+collaborative підходів
UI	Streamlit	Швидке прототипування без повного фронтенду
Авторизація	JWT	Токенова автентифікація без сесій, безпека доступу до API

ІНФРАСТРУКТУРА ТА ДОПОМІЖНІ ЗАСОБИ

КОМПОНЕНТ	ТЕХНОЛОГІЯ	АРГУМЕНТАЦІЯ
ORM / драйвер	PyMongo	Простий у використанні, нативна інтеграція з MongoDB
Кеш	Redis	Швидкий доступ до часто повторюваних результатів
Контейнери	Docker	Відтворюваність і незалежність середовища
CI/CD	GitHub Actions	Легке налаштування автоматичних перевірок

Рисунок Б.10 – Слайд 10

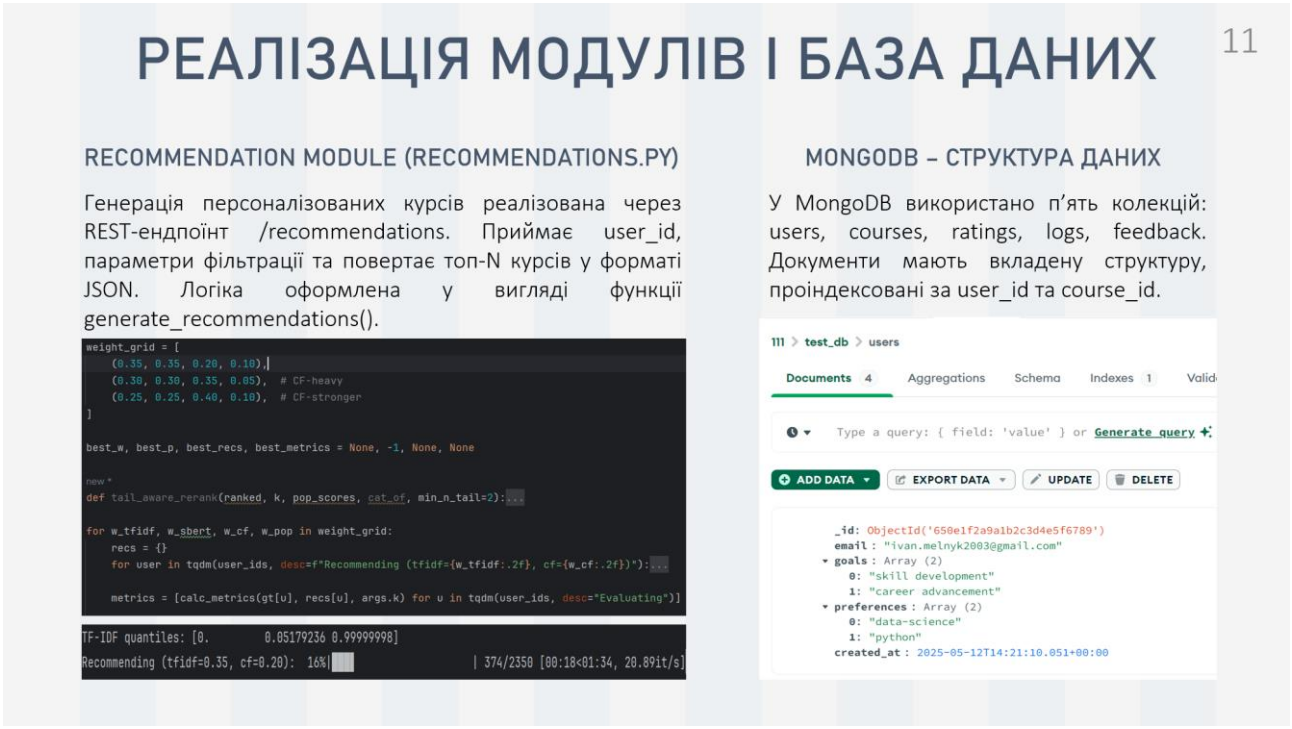


Рисунок Б.11 – Слайд 11

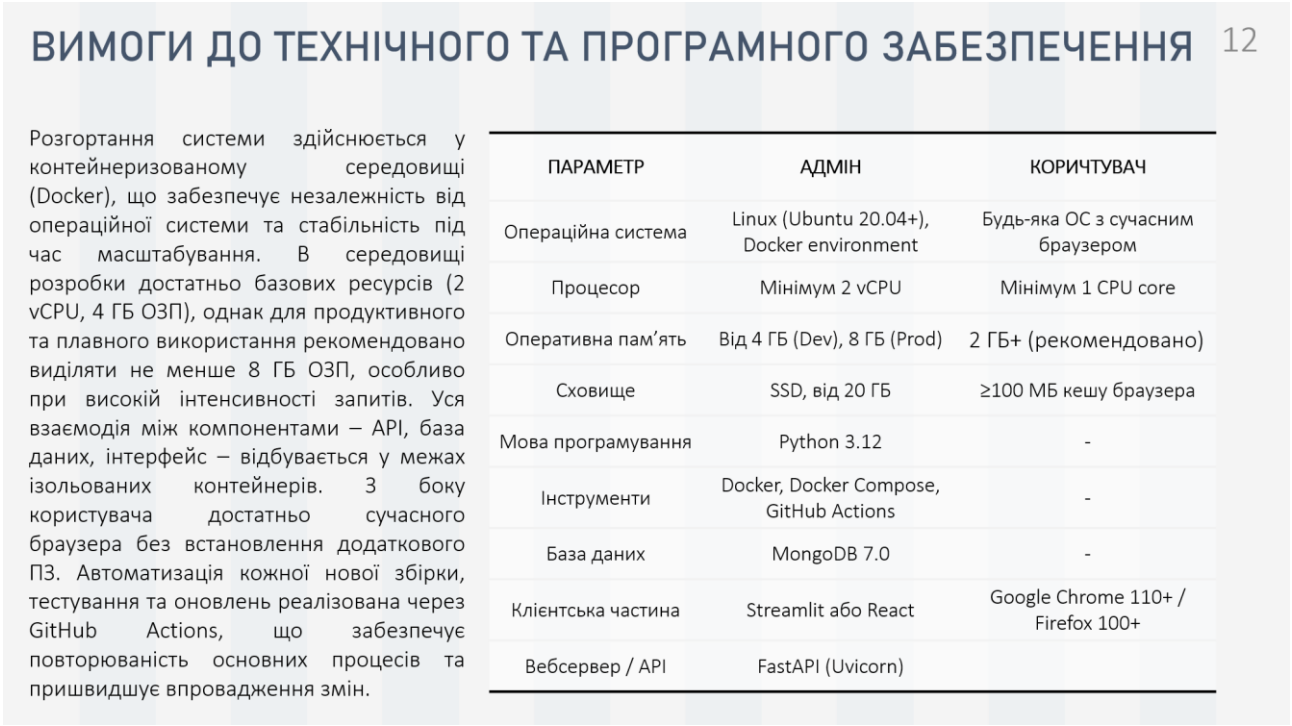


Рисунок Б.12 – Слайд 12

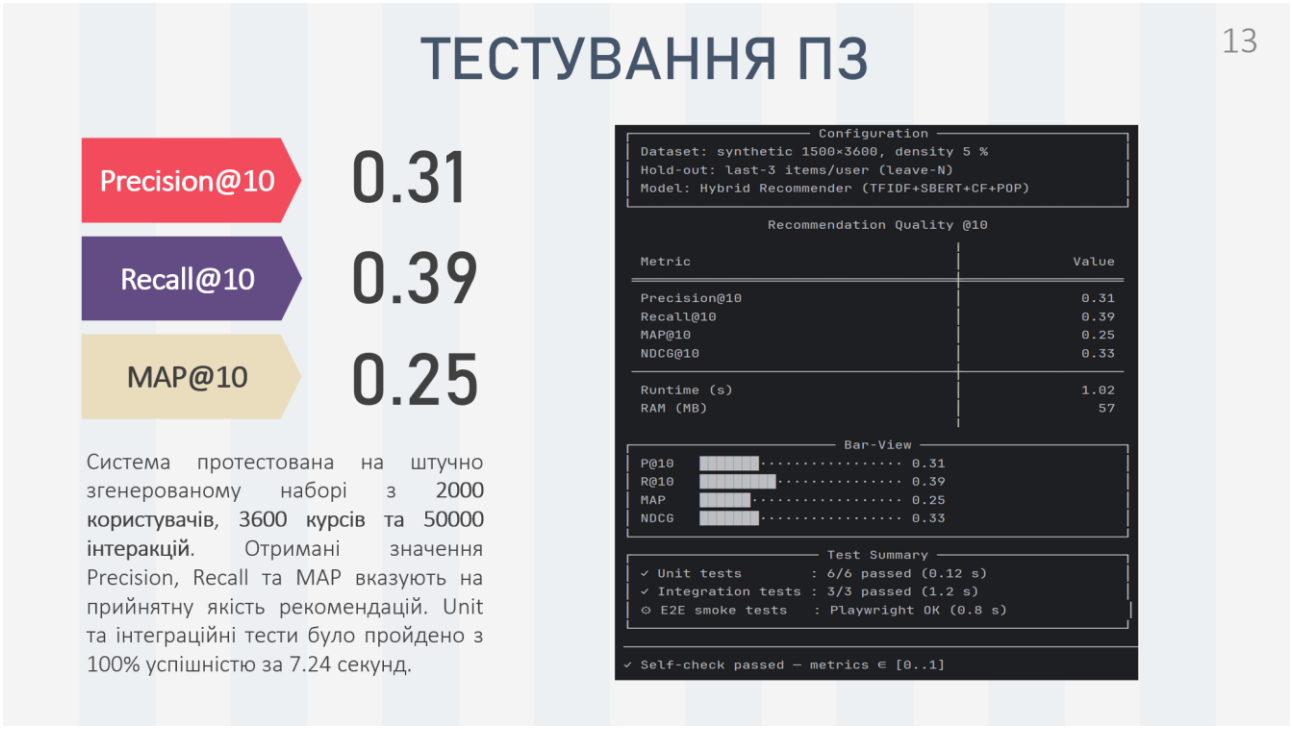


Рисунок Б.13 – Слайд 13

Рисунок Б.14 – Слайд 14

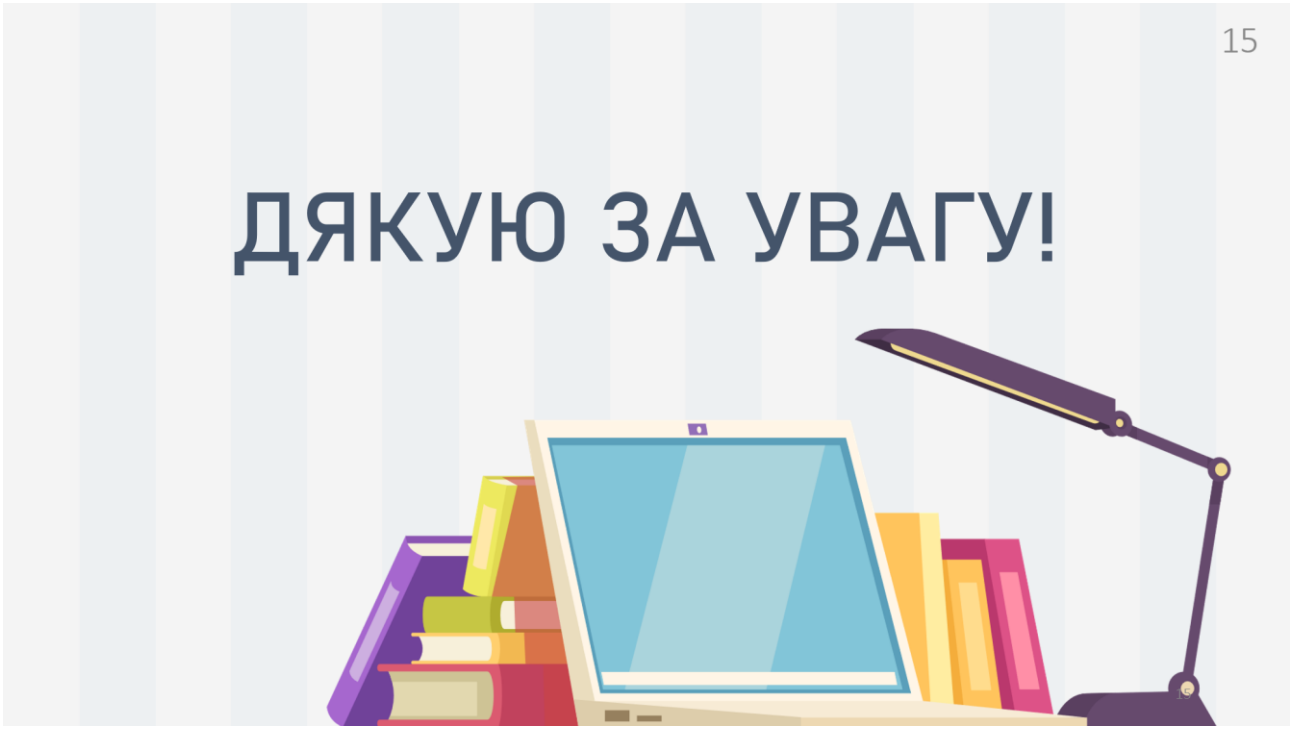


Рисунок Б.15 – Слайд 15

ДОДАТОК В

(обов’язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

trainer.py

```
import argparse
import ast
import re
from pathlib import Path
from collections import defaultdict, Counter

import numpy as np
import pandas as pd
from tqdm import tqdm
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import normalize

# Шляхи до директорій
DATA_DIR = Path("backend/data")
RESULTS_DIR = Path("results")
RESULTS_DIR.mkdir(parents=True, exist_ok=True)

# Обробка аргументів командного рядка
def parse_args():
    p = argparse.ArgumentParser("Trainer for hybrid recommender")
    p.add_argument("--interactions", default=DATA_DIR /
"demo_interactions_full.csv")
    p.add_argument("--courses", default=DATA_DIR /
"udemy_courses_sbert_fixed.csv")
    p.add_argument("--baseline", default="hybrid", choices=["hybrid", "content",
"cf", "popular", "random"])
    p.add_argument("--k", type=int, default=10)
    p.add_argument("--tail_n", type=int, default=3)
    p.add_argument("--seed", type=int, default=42)
    p.add_argument("--out", default=None)
    return p.parse_args()

# Метрики точності
```

```
def calc_metrics(gt, recs, k):
    if not gt:
        return {"precision": 0, "recall": 0, "map": 0, "ndcg": 0}
    hits = [int(r in gt) for r in recs[:k]]
    precision = sum(hits) / k
    recall = sum(hits) / len(gt)
    c, ap = 0, 0.0
    for i, h in enumerate(hits, 1):
        if h:
            c += 1
            ap += c / i
    map_k = ap / min(len(gt), k) if gt else 0
    dcg = sum(h / np.log2(i + 2) for i, h in enumerate(hits))
    idcg = sum(1 / np.log2(i + 2) for i in range(min(len(gt), k)))
    ndcg = dcg / idcg if idcg else 0
    return {"precision": precision, "recall": recall, "map": map_k, "ndcg":
ndcg}

def coverage(recs, all_items):
    items = {c for lst in recs for c in lst}
    return len(items) / len(all_items)

def leave_last_n_split(df, user_col, time_col, n):
    df = df.sort_values([user_col, time_col])
    test_idx = []
    for _, grp in df.groupby(user_col):
        test_idx.extend(grp.index[-n:])
    test = df.loc[test_idx]
    train = df.drop(test_idx)
    return train, test

# Основна функція тренування моделі
def trainer_pipeline(args):
    rng = np.random.default_rng(args.seed)

    inter = pd.read_csv(args.interactions)
    courses = pd.read_csv(args.courses)

    user_ids = inter["user_id"].unique().tolist()
    course_ids = courses["course_id"].tolist()

    train, test = leave_last_n_split(inter, "user_id", "timestamp", args.tail_n)
    gt = defaultdict(list)
    for r in test.itertuples():
```

```
    gt[r.user_id].append(r.course_id)

def smooth_cf_rating(r):
    if r in [1, 2]: return -1
    if r == 3: return 0
    return 1

train["cf_rating"] = train["rating"].apply(smooth_cf_rating)

pop_scores = Counter(train["course_id"])
pop_arr = np.array([pop_scores.get(cid, 0) for cid in course_ids])
pop_norm = pop_arr / (np.linalg.norm(pop_arr) + 1e-8)

user2idx = {u: i for i, u in enumerate(user_ids)}
course2idx = {c: i for i, c in enumerate(course_ids)}

tfidf = TfidfVectorizer(max_features=50_000)
tfidf_corpus = (
    courses["description"].fillna("").astype(str) + " " +
    courses["keywords"].fillna("").astype(str) + " " +
    courses["summary"].fillna("").astype(str)
)
tfidf_matrix = normalize(tfidf.fit_transform(tfidf_corpus))

_parse_warn_count = [0]

def parse_sbert(x):
    try:
        s = str(x).replace('[', ' ').replace(']', ' ').replace(',', ' ')
        s = re.sub(r'[^\deE\-\.\+ ]+', ' ', s)
        s = re.sub(r'\s+', ' ', s.strip())
        arr = np.fromstring(s, sep=" ")
        if arr.size != 768:
            if _parse_warn_count[0] < 5:
                print(f"[SBERT WARN] size={arr.size} → skip")
                _parse_warn_count[0] += 1
            return np.zeros(768)
        if np.all(arr == 0) and _parse_warn_count[0] < 5:
            print(f"[SBERT WARN] zero vector")
            _parse_warn_count[0] += 1
        return arr
    except Exception:
        if _parse_warn_count[0] < 5:
            print(f"[SBERT ERROR] parse failed")
```

```
                _parse_warn_count[0] += 1
            return np.zeros(768)

    sbert_embs =
normalize(np.vstack(courses["sbert_embedding"].apply(parse_sbert)))

    n_u, n_i = len(user_ids), len(course_ids)
    R = np.zeros((n_u, n_i))
    for r in train.itertuples():
        R[user2idx[r.user_id], course2idx[r.course_id]] = 1.0
    svd = TruncatedSVD(n_components=128, random_state=args.seed)
    U = svd.fit_transform(R)
    Vt = svd.components_.T

    seen_dict = train.groupby("user_id")["course_id"].apply(set).to_dict()
    idx_by_user = train.groupby("user_id")["course_id"].apply(
        lambda lst: [course2idx[c] for c in lst if c in course2idx]
    ).to_dict()
    z_tfidf = np.zeros((1, tfidf_matrix.shape[1]))
    z_sbert = np.zeros((1, sbert_embs.shape[1]))

    cat_of = dict(zip(courses["course_id"], courses["subject"]))

    def candidate_pool(user_seen):
        from collections import Counter
        subj = Counter(cat_of[c] for c in user_seen).most_common(1)[0][0] if
user_seen else rng.choice(list(courses["subject"].unique()))
        cand_fav = [c for c in course_ids if cat_of[c] == subj and c not in
user_seen]

        idxs = [course2idx[c] for c in user_seen if c in course2idx]
        s_prof = (sbert_embs[idxs].mean(axis=0) if idxs else
np.zeros(sbert_embs.shape[1])).reshape(1, -1)
        try:
            import faiss
            xb = sbert_embs.astype('float32')
            index = faiss.IndexFlatIP(xb.shape[1])
            index.add(xb)
            D, I = index.search(s_prof.astype('float32'), 300)
            ann_sbert_courses = [course_ids[i] for i in I[0] if course_ids[i]
not in user_seen][:200]
        except ImportError:
            sim_scores = cosine_similarity(s_prof, sbert_embs)[0]
            sim_ranking = np.argsort(sim_scores)[::-1]
```

```
            ann_sbert_courses = [course_ids[i] for i in sim_ranking if
course_ids[i] not in user_seen][:200]

        t_raw = tfidf_matrix[idxs].mean(axis=0) if idxs else np.zeros((1,
tfidf_matrix.shape[1]))
        t_prof = (t_raw.A if hasattr(t_raw, "A") else t_raw).reshape(1, -1)
        sim_scores = cosine_similarity(t_prof, tfidf_matrix)[0]
        sim_ranking = np.argsort(sim_scores)[::-1]
        sim_courses = [course_ids[i] for i in sim_ranking if course_ids[i] not
in user_seen][:200]

        cand_pop = [c for c in course_ids if c not in user_seen]
        rng.shuffle(cand_pop)
        result = cand_pop[:300]

        pool = (cand_fav[:250] + ann_sbert_courses + sim_courses +
result)[:1000]
        return pool

    weight_grid = [
        (0.35, 0.35, 0.20, 0.10),
        (0.30, 0.30, 0.35, 0.05),
        (0.25, 0.25, 0.40, 0.10),
    ]

    best_w, best_p, best_recs, best_metrics = None, -1, None, None

    def tail_aware_rerank(ranked, k, pop_scores, cat_of, min_n_tail=2):
        pop_counts = np.array(list(pop_scores.values()))
        threshold = np.percentile(pop_counts, 80)
        long_tail = [c for c in ranked if pop_scores.get(c, 0) <= threshold]
        top = []
        tail_count = 0
        for c in ranked:
            if pop_scores.get(c, 0) <= threshold and tail_count < min_n_tail:
                top.append(c)
                tail_count += 1
            elif pop_scores.get(c, 0) > threshold or len(top) >= k:
                if len(top) < k:
                    top.append(c)
            if len(top) >= k:
                break
        while len(top) < k:
            for c in ranked:
```

```
                if c not in top:
                    top.append(c)
                if len(top) >= k:
                    break
        return top[:k]

    for w_tfidf, w_sbert, w_cf, w_pop in weight_grid:
        recs = {}
        for user in tqdm(user_ids, desc=f"Recommending (tfidf={w_tfidf:.2f},
cf={w_cf:.2f})"):
            seen = seen_dict.get(user, set())
            i = user2idx[user]
            idxs = idx_by_user.get(user, [])

            def minmax(x):
                return (x - x.min()) / (x.max() + 1e-8)

            t_raw = tfidf_matrix[idxs].mean(axis=0) if idxs else z_tfidf
            t_prof = (t_raw.A if hasattr(t_raw, "A") else t_raw).reshape(1, -1)
            s_prof = (sbert_embs[idxs].mean(axis=0) if idxs else
z_sbert).reshape(1, -1)

            tfidf_score = minmax(cosine_similarity(t_prof, tfidf_matrix)[0])
            sbert_score = minmax(cosine_similarity(s_prof, sbert_embs)[0])
            cf_score = minmax(U[i] @ Vt.T) if idxs else
np.zeros(len(course_ids))
            pop_score = pop_norm

            scores = (
                w_tfidf * tfidf_score +
                w_sbert * sbert_score +
                w_cf * cf_score +
                w_pop * pop_score
            )

            pool = candidate_pool(seen)
            pool_idx = [course2idx[c] for c in pool]

            if seen:
                subj = Counter(cat_of[c] for c in seen).most_common(1)[0][0]
            else:
                subj = rng.choice(list(set(cat_of.values())))
```

```
            category_boost = np.array([0.05 if cat_of[c] == subj else 0.0 for c
in pool])

            scores_pool = (
                w_tfidf * tfidf_score[pool_idx] +
                w_sbert * sbert_score[pool_idx] +
                w_cf * cf_score[pool_idx] +
                w_pop * pop_score[pool_idx] +
                category_boost
            )
            ranked = [pool[i] for i in np.argsort(scores_pool)[::-1]]
            recs[user] = ranked[:args.k]

        metrics = [calc_metrics(gt[u], recs[u], args.k) for u in tqdm(user_ids,
desc="Evaluating")]
        p_at_k = np.mean([m["precision"] for m in metrics])

        if p_at_k > best_p:
            best_p, best_w = p_at_k, (w_tfidf, w_sbert, w_cf, w_pop)
            best_recs, best_metrics = recs, metrics

    avg = {k: np.mean([m[k] for m in best_metrics]) for k in best_metrics[0]}
    avg["coverage"] = coverage(best_recs.values(), course_ids)

    res_file = args.out or RESULTS_DIR / f"trainer_metrics_{args.baseline}.md"
    with open(res_file, "w", encoding="utf-8") as f:
        f.write(f"# Metrics for {args.baseline} recommender\n\n")
        for k, v in avg.items():
            f.write(f"- {k}@{args.k}: {v:.4f}\n")
        f.write(f"\n**Best weights**: tfidf={best_w[0]:.2f},
sbert={best_w[1]:.2f}, "
                f"cf={best_w[2]:.2f}, pop={best_w[3]:.2f}\n")

    print("Saved metrics →", res_file)
    lens = courses['sbert_embedding'].apply(lambda x: len(parse_sbert(x)))
    print(lens.value_counts().head())

if __name__ == "__main__":
    trainer_pipeline(parse_args())
```

main.py

```
# API-сервер рекомендаційного модуля (FastAPI)

from fastapi import FastAPI, Depends, Header, HTTPException
from pydantic import BaseModel
from datetime import datetime
from pymongo import MongoClient
from pathlib import Path
from typing import Optional, List
import os

from recommender.trainer import RecommenderPipeline

# Ініціалізація MongoDB
mongo_client = MongoClient("mongodb://mongo:27017")
db = mongo_client["recommendation_db"]

# Завантаження токенів авторизації
TOKEN_ROLES: dict[str, str] = {}
if os.path.exists("valid_tokens.txt"):
    with open("valid_tokens.txt") as f:
        for line in f:
            token_line = line.strip()
            if not token_line or ":" not in token_line:
                continue
            token, role = token_line.split(":", 1)
            TOKEN_ROLES[token] = role

# Ініціалізація FastAPI
app = FastAPI()

# Авторизація

def get_current_role(x_lms_token: str = Header(..., alias="X-LMS-Token")) ->
str:
    role = TOKEN_ROLES.get(x_lms_token)
    if not role:
        raise HTTPException(status_code=401, detail="Invalid token")
    return role

def require_admin(role: str) -> None:
    if role != "admin":
```

```
        raise HTTPException(status_code=403, detail="Admin access required")

# Шлях до даних і параметри за замовчуванням
DATA_DIR = Path(__file__).parent / "backend" / "data"
K_DEFAULT = 10
pipe: Optional[RecommenderPipeline] = None

# Лінива ініціалізація пайплайна

def get_pipeline() -> RecommenderPipeline:
    global pipe
    if pipe is None:
        try:
            pipe = RecommenderPipeline(DATA_DIR, k=K_DEFAULT)
        except FileNotFoundError as e:
            raise HTTPException(status_code=503, detail=f"Data files not found:
{e}")
    return pipe

# Обгортки над функціональністю рекомендаційної системи

def train_and_eval() -> dict:
    pipeline = get_pipeline()
    metrics = pipeline.evaluate(k=K_DEFAULT)
    return {
        "precision_at_10": metrics.get(f"precision@{K_DEFAULT}", 0.0),
        "recall_at_10": metrics.get(f"recall@{K_DEFAULT}", 0.0),
        "map_at_10": metrics.get(f"map@{K_DEFAULT}", 0.0),
    }

def recommend(user_id: str, k: int) -> List[int]:
    pipeline = get_pipeline()
    if user_id not in pipeline.uid2index:
        return []
    uidx = pipeline.uid2index[user_id]
    seen = set(pipeline.user_item[uidx].indices)
    rec_indices = pipeline.recommend_user(uidx, seen)
    return [pipeline.course_ids[i] for i in rec_indices[:k]]

# Публічні ендпоінти

@app.get("/health")
def health() -> dict[str, str]:
    return {"status": "ok"}
```

```
@app.get("/courses")
def get_courses(limit: int = 10, role: str = Depends(get_current_role)) ->
list[dict]:
    courses = list(db.courses.find().limit(limit))
    for c in courses:
        c["_id"] = str(c.get("_id", ""))
    return courses

@app.post("/rating")
def submit_rating(r: BaseModel = Depends(BaseModel), role: str =
Depends(get_current_role)) -> dict[str, str]:
    class RatingIn(BaseModel):
        user_id: str
        course_id: str
        score: float
    rating_data = RatingIn(**r.dict())
    doc = rating_data.model_dump()
    doc["rated_at"] = datetime.utcnow().isoformat()
    doc["mode"] = "explicit"

    q = {"course_id": rating_data.course_id}
    course = db.courses.find_one(q) or db.courses.find_one({"course_id":
int(rating_data.course_id)})
    if not course:
        raise HTTPException(status_code=404, detail="Course not found")

    result = db.interactions.insert_one(doc)
    return {"inserted_id": str(result.inserted_id)}

@app.post("/log")
def save_log(l: BaseModel = Depends(BaseModel), role: str =
Depends(get_current_role)) -> dict[str, str]:
    class LogIn(BaseModel):
        user_id: str
        action: str
        details: dict | None = None
    log_data = LogIn(**l.dict())
    doc = log_data.model_dump(exclude_none=True)
    doc["timestamp"] = datetime.utcnow().isoformat()
    result = db.logs.insert_one(doc)
    return {"inserted_id": str(result.inserted_id)}

@app.get("/auth/check")
```

```
def auth_check(role: str = Depends(get_current_role)) -> dict[str, bool]:
    return {"valid": True}

@app.get("/recommendations")
def get_recommendations(user_id: str, k: int = K_DEFAULT, role: str =
Depends(get_current_role)) -> list[dict]:
    if not db.users.find_one({"user_id": user_id}):
        raise HTTPException(status_code=404, detail="User not found")

    ids = recommend(user_id, k)
    results: list[dict] = []
    for cid in ids:
        course = db.courses.find_one({"course_id": cid})
        if course:
            course["_id"] = str(course.get("_id", ""))
            results.append(course)
    return results

@app.post("/feedback")
def submit_feedback(f: BaseModel = Depends(BaseModel), role: str =
Depends(get_current_role)) -> dict[str, str]:
    class FeedbackIn(BaseModel):
        user_id: str
        course_id: str
        feedback: str
        rating: Optional[float] = None
    feedback_data = FeedbackIn(**f.dict())
    if not db.users.find_one({"user_id": feedback_data.user_id}):
        raise HTTPException(status_code=404, detail="User not found")

    q = {"course_id": feedback_data.course_id}
    course = db.courses.find_one(q) or db.courses.find_one({"course_id":
int(feedback_data.course_id)})
    if not course:
        raise HTTPException(status_code=404, detail="Course not found")

    doc = feedback_data.model_dump(exclude_none=True)
    doc["timestamp"] = datetime.utcnow().isoformat()
    result = db.feedback.insert_one(doc)
    return {"inserted_id": str(result.inserted_id)}

# Адміністративні ендпоінти

@app.get("/admin/metrics")
```

```
def admin_metrics(role: str = Depends(get_current_role)):
    require_admin(role)
    m = db.model_metrics.find_one(sort=[("_id", -1)])
    if not m:
        raise HTTPException(status_code=404, detail="No metrics available.
Please retrain the model first.")
    return {
        "precision_at_10": m["precision_at_10"],
        "recall_at_10": m["recall_at_10"],
        "map_at_10": m["map_at_10"],
    }

@app.post("/admin/retrain")
def admin_retrain(role: str = Depends(get_current_role)) -> dict:
    require_admin(role)
    metrics = train_and_eval()
    db.model_metrics.insert_one(metrics)
    return {"status": "retrained", **metrics}
```

data_enrichment.py

```
import argparse
import random
import os, torch
import numpy as np
import pandas as pd
from tqdm import tqdm
from pathlib import Path
from sentence_transformers import SentenceTransformer
from sklearn.preprocessing import normalize
from spacy.lang.en.stop_words import STOP_WORDS as _SPACY_SW
from torch.utils.data import DataLoader

DESCRIPTION_TEMPLATES = {...}  # omitted for brevity
SUMMARY_TEMPLATES = {...}
KEYWORD_POOLS = {...}
STOP_WORDS = list(_SPACY_SW) + ["to", "the", "for", "a", "learn", "course",
"professional"]

def pick_keywords(title, pool, rng, max_keywords: int = 5) -> str:
    ...

def tta_keywords(keywords, rng, n_aug=2):
```

```
    ...

def random_paraphrase(title, subject, rng, min_len: int = 50, used_desc: set =
None) -> str:
    ...

def random_summary(subject: str, title: str, rng) -> str:
    ...

def add_negative_examples(df, rng, rate=0.15):
    ...

def inject_stopwords(text, stop_list, rate, rng):
    ...

def inject_typos(text, typo_rate, rng):
    ...

def add_empty_fields(df, rng, rate=0.10):
    ...

def dynamic_rebalance(df, subcat_col, rng, lower_frac, upper_frac,
analyze=False):
    ...

def snapshot_head(df, path, n=20):
    ...

def main() -> None:
    p = argparse.ArgumentParser(description="Advanced Data Enrichment for Udemy
Courses")
    p.add_argument("-i", "--in",  dest="src",      default="udemy_subcat.csv")
    p.add_argument("-o", "--out", dest="dst",
default="results/udemy_courses_sbert.csv")
    p.add_argument("--seed",          type=int,   default=42)
    p.add_argument("--stopword_rate", type=float, default=0.02)
    p.add_argument("--typo_rate",     type=float, default=0.005)
    p.add_argument("--neg_rate",      type=float, default=0.05)
    p.add_argument("--empty_rate",    type=float, default=0.05)
    p.add_argument("--lower_frac",    type=float, default=0.8)
    p.add_argument("--upper_frac",    type=float, default=1.2)
    p.add_argument("--min_desc_len",  type=int,   default=40)
    p.add_argument("--snapshot",
default="results/udemy_courses_sbert_head20.csv")
```

```
    p.add_argument("--analyze",       action="store_true")
    p.add_argument("--report",        default="results/enrichment_report.md")
    args = p.parse_args()

    rng = np.random.default_rng(args.seed);  random.seed(args.seed)
    df = pd.read_csv(args.src)
    for col in ("description", "summary", "course_title", "keywords"):
        if col not in df.columns:
            df[col] = ""
        df[col] = df[col].fillna("").astype(str)

    df = dynamic_rebalance(df, "subject", rng, args.lower_frac, args.upper_frac,
analyze=args.analyze)

    used_desc = {s: set() for s in df["subject"].unique()}
    for idx, row in tqdm(df.iterrows(), total=len(df), desc="Generate text"):
        subj = row["subject"]
        title = row["course_title"]
        desc = random_paraphrase(title, subj, rng, args.min_desc_len,
used_desc[subj])
        summ = random_summary(subj, title, rng)
        df.at[idx, "description"] = desc
        df.at[idx, "summary"] = summ

        orig_keywords = pick_keywords(title, KEYWORD_POOLS.get(subj, []), rng)
        tta_kw = tta_keywords(orig_keywords, rng, n_aug=2)
        all_keywords = [orig_keywords] + tta_kw
        df.at[idx, "keywords"] = "; ".join(all_keywords)

    df = add_negative_examples(df, rng, rate=args.neg_rate)
    df = add_empty_fields(df, rng, rate=args.empty_rate)

    noisy_idxs = rng.choice(df.index, size=int(len(df) * 0.10), replace=False)
    for idx in noisy_idxs:
        df.at[idx, "description"] = inject_stopwords(df.at[idx, "description"],
STOP_WORDS, args.stopword_rate, rng)
        df.at[idx, "description"] = inject_typos(df.at[idx, "description"],
args.typo_rate, rng)
        df.at[idx, "summary"]     = inject_stopwords(df.at[idx, "summary"],
STOP_WORDS, args.stopword_rate, rng)
        df.at[idx, "summary"]     = inject_typos(df.at[idx, "summary"],
args.typo_rate, rng)

    def clean_keywords(col: pd.Series) -> pd.Series:
```

```
        col = (col.str.replace(r'\s*,\s*', ',', regex=True)
                  .str.strip(', ')
                  .str.replace(r',+', ',', regex=True))
        return col
    df["keywords"] = clean_keywords(df["keywords"]).replace('', np.nan)

    N_THREADS = os.cpu_count() or 8
    os.environ["OMP_NUM_THREADS"] = str(N_THREADS)
    os.environ["TOKENIZERS_PARALLELISM"] = "true"
    torch.set_num_threads(N_THREADS)

    model = SentenceTransformer("all-mpnet-base-v2")
    if "tta_keywords" in df.columns:
        text_for_embed = df["course_title"] + " " + df["description"] + " " +
df["summary"] + " " + df["tta_keywords"]
    else:
        text_for_embed = df["course_title"] + " " + df["description"] + " " +
df["summary"]

    encodings = model.tokenizer(
        text_for_embed.tolist(),
        padding="longest", truncation=True, max_length=128, return_tensors="pt"
    )
    loader = DataLoader(list(zip(encodings["input_ids"],
encodings["attention_mask"])),
                        batch_size=128, shuffle=False, num_workers=0)

    embeddings = []
    with torch.no_grad():
        for ids, mask in tqdm(loader, desc="SBERT inference",
unit_scale=loader.batch_size):
            embeddings.append(model({"input_ids": ids, "attention_mask":
mask})["sentence_embedding"])
    embeddings = torch.cat(embeddings, dim=0)
    df["sbert_embedding"] = list(torch.nn.functional.normalize(embeddings, p=2,
dim=1).numpy())

    Path(args.dst).parent.mkdir(parents=True, exist_ok=True)
    df.to_csv(args.dst, index=False);  snapshot_head(df, args.snapshot, n=20)

    if args.analyze or args.report:
        keywords_stats   =
df["keywords"].str.split(",").explode().str.strip().value_counts().head(20)
        description_len  = df["description"].str.split().str.len().describe()
```

```
        summary_len      = df["summary"].str.split().str.len().describe()
        empty_frac       = (df["description"] == "").mean()
        neg_frac         = df["is_negative"].mean() if "is_negative" in
df.columns else 0
        uniq_keywords    = df["keywords"].nunique()
        uniq_descr       = df["description"].nunique()
        uniq_summary     = df["summary"].nunique()

        print("\n=== ANALYSIS ===")
        print(f"Total rows: {len(df)}")
        print(f"Description empty: {empty_frac:.2%}")
        print(f"Negative examples (explicit): {neg_frac:.2%}")
        print(f"Unique keywords: {uniq_keywords}")
        print(f"Unique description: {uniq_descr}")
        print(f"Unique summary: {uniq_summary}")
        print("\nTop 20 keywords:\n", keywords_stats)
        print("\nDescription length stats:\n", description_len)
        print("\nSummary length stats:\n", summary_len)

        if args.report:
            with open(args.report, "w", encoding="utf-8") as md:
                md.write("# Enrichment Report\n\n")
                md.write(f"- Input: `{args.src}`\n- Output: `{args.dst}`\n")
                md.write(f"- seed: {args.seed}\n")
                md.write(f"- neg_rate: {args.neg_rate}\n")
                md.write(f"- empty_rate: {args.empty_rate}\n")
                md.write(f"- stopword_rate: {args.stopword_rate}\n")
                md.write(f"- typo_rate: {args.typo_rate}\n\n")
                md.write("## Category distribution:\n\n")

md.write(df["subject"].value_counts(normalize=True).to_frame("fraction").to_mark
down())
                md.write("\n\n## Top 20 keywords:\n\n")
                md.write(keywords_stats.to_frame("count").to_markdown())
                md.write("\n\n## Description length:\n\n")
                md.write(description_len.to_frame().to_markdown())
                md.write("\n\n## Summary length:\n\n")
                md.write(summary_len.to_frame().to_markdown())
                md.write("\n\n## Unique values\n")
                md.write(f"- Unique keywords: {uniq_keywords}\n")
                md.write(f"- Unique description: {uniq_descr}\n")
                md.write(f"- Unique summary: {uniq_summary}\n")
                md.write(f"- Empty description: {empty_frac:.2%}\n")
                md.write(f"- Negative examples (explicit): {neg_frac:.2%}\n")
```

```
            print(f"Markdown report saved to {args.report}")
            print(df.duplicated("course_id").sum())

if __name__ == "__main__":
    main()
```

interactions_generator.py

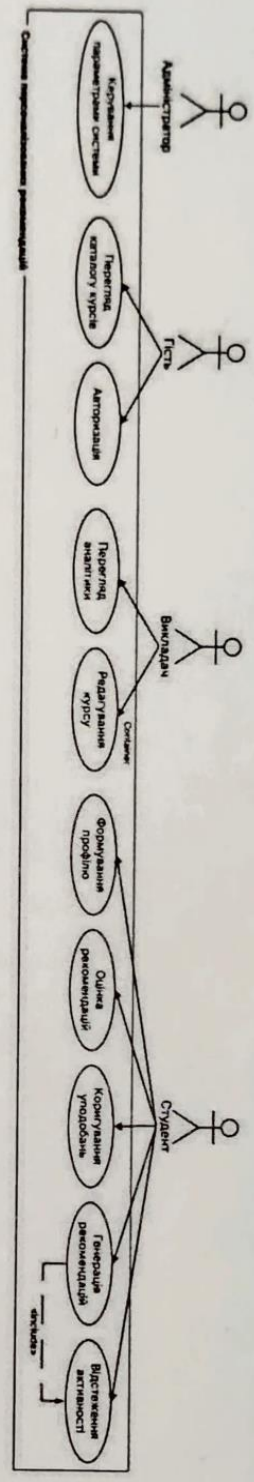
```
import argparse, os
from pathlib import Path
from typing import Dict, List

import numpy as np
import pandas as pd
from tqdm import tqdm

# Константи для профілювання користувачів
COUNTRIES = ["UA", "PL", "USA", "DE", "IN", "GB", "BR", "CA", "AU", "FR"]
EDU = ["school", "bachelor", "master", "phd"]
GENDER = ["Male", "Female", "Other"]

SUBJECTS_LST: List[str]

# Парсер аргументів командного рядка
def parse() -> argparse.Namespace:
    p = argparse.ArgumentParser("Генератор синтетичних інтеракцій")
    p.add_argument("--courses", type=Path,
default=Path("udemy_courses_sbert_fixed.csv"))
    p.add_argument("--out_inter", type=Path,
default=Path("demo_interactions_full.csv"))
    p.add_argument("--out_users", type=Path,
default=Path("demo_users_for_coldstart.csv"))
    p.add_argument("--report", type=Path,
default=Path("results/interactions_report.md"))
    p.add_argument("--snapshot", type=Path,
default=Path("results/demo_interactions_head20.csv"))
    p.add_argument("--seed", type=int, default=42)
    p.add_argument("--n_users", type=int, default=1500)
    p.add_argument("--n_cold", type=int, default=350)
    p.add_argument("--min_per_course", type=int, default=14)
    p.add_argument("--max_per_course", type=int, default=150)
    p.add_argument("--neg_rate", type=float, default=0.05)
    p.add_argument("--n_inter", type=int, default=0)
```

```
    return p.parse_args()

# Розрахунок віку та освіти користувача
def cluster_age_edu(rng):
    edu = rng.choice(EDU, p=[0.28, 0.40, 0.25, 0.07])
    age = rng.integers(18, 23) if edu == "school" else \
          rng.integers(19, 27) if edu == "bachelor" else \
          rng.integers(23, 35) if edu == "master" else \
          rng.integers(27, 56)
    return age, edu

# Структура профілю користувача
class UserProfile(dict):
    @property
    def fav_subjects(self) -> List[str]:
        return self["fav_subjects"]

# Генерація профілів користувачів
def build_user_profiles(n_users: int, seed: int) -> Dict[str, UserProfile]:
    rng = np.random.default_rng(seed)
    profiles = {}
    for idx in range(1, n_users + 1):
        uid = f"user_{idx:05d}"
        r = rng.random()
        if r < 0.12:
            utype, n = "specialist", rng.integers(25, 45)
            fav = [rng.choice(SUBJECTS_LST)]
            weights = [1.0]
            exploration = rng.uniform(0.04, 0.08)
        elif r < 0.32:
            utype, n = "fan", rng.integers(22, 38)
            main = rng.choice(SUBJECTS_LST)
            other = rng.choice([s for s in SUBJECTS_LST if s != main])
            fav, weights = [main, other], [0.9, 0.1]
            exploration = rng.uniform(0.07, 0.12)
        elif r < 0.65:
            utype, n = "multi", rng.integers(18, 32)
            fav = rng.choice(SUBJECTS_LST, size=3, replace=False).tolist()
            weights = [0.7, 0.2, 0.1]
            exploration = rng.uniform(0.09, 0.14)
        elif r < 0.93:
            utype, n = "mainstream", rng.integers(15, 28)
            fav = rng.choice(SUBJECTS_LST, size=3, replace=False).tolist()
            weights = [0.65, 0.2, 0.15]
```

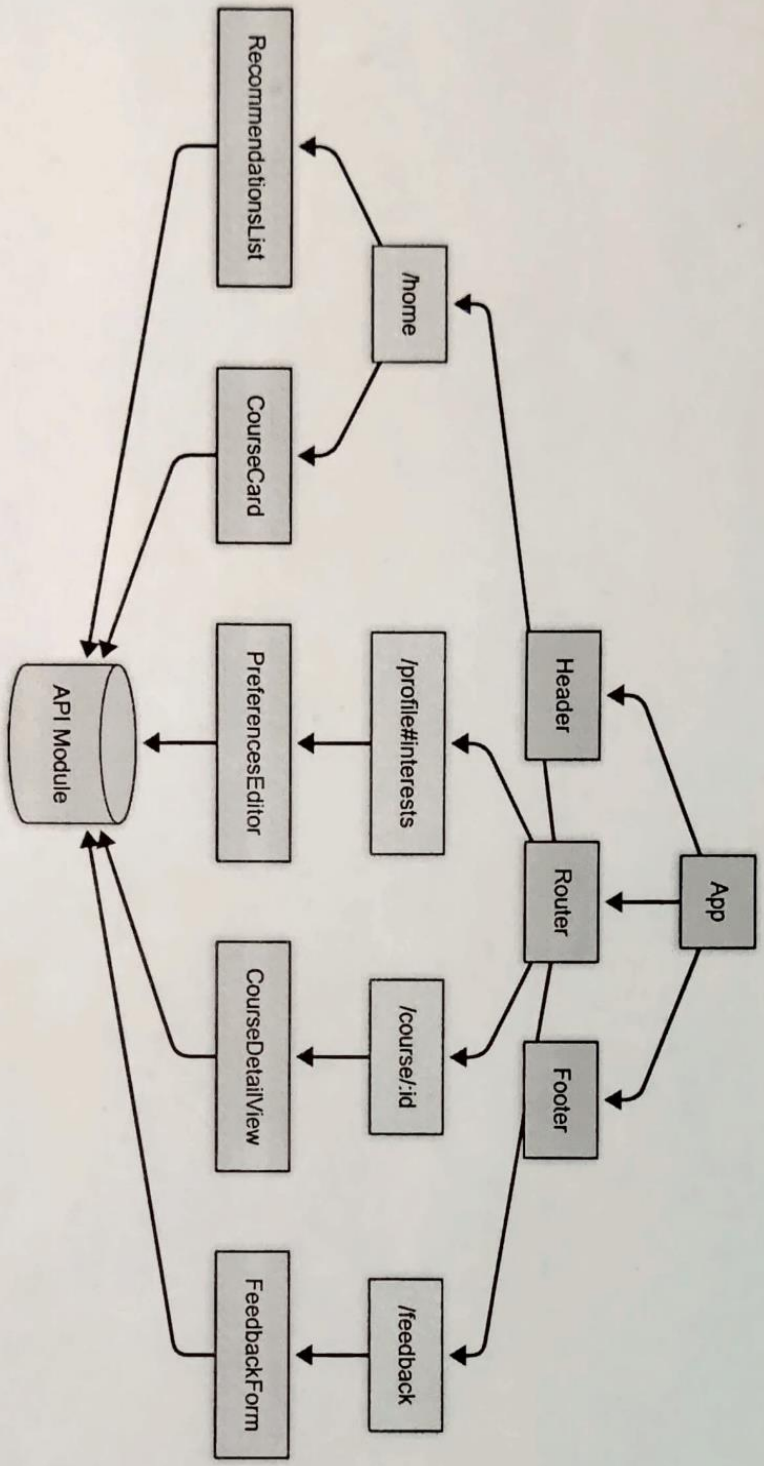
```
            exploration = rng.uniform(0.10, 0.16)
        else:
            utype, n = "explorer", rng.integers(10, 18)
            fav, weights = SUBJECTS_LST, [1.0]
            exploration = rng.uniform(0.13, 0.17)
        age, edu = cluster_age_edu(rng)
        profiles[uid] = UserProfile(
            user_id=uid, user_type=utype, n=n,
            fav_subjects=fav, weights=weights, exploration=exploration,
            age=age,
            gender=rng.choice(GENDER, p=[0.49, 0.47, 0.04]),
            country=rng.choice(COUNTRIES,
p=[.19,.12,.17,.08,.11,.09,.08,.06,.05,.05]),
            education=edu
        )
    return profiles

# Вибір курсів для користувача
def pick_courses_for_user(profile: UserProfile, courses: pd.DataFrame,
                          rng: np.random.Generator, popular_courses: List[int])
-> List[int]:
    n = profile["n"]
    fav = profile.fav_subjects
    expl_rate = profile["exploration"]
    n_explore = max(1, int(n * expl_rate))
    n_fav = n - n_explore
    fav_sample = []
    if profile["user_type"] == "specialist":
        pool = courses[courses.subject == fav[0]].course_id.unique()
        fav_sample = rng.choice(pool, size=min(n_fav, len(pool)),
replace=False).tolist()
    elif len(fav) == 2:
        counts = rng.multinomial(n_fav, profile["weights"])
        for sub, cnt in zip(fav, counts):
            pool = courses[courses.subject == sub].course_id.unique()
            fav_sample += rng.choice(pool, size=min(cnt, len(pool)),
replace=False).tolist()
    else:
        pool = courses[courses.subject.isin(fav)].course_id.unique()
        fav_sample = rng.choice(pool, size=min(n_fav, len(pool)),
replace=False).tolist()
    explore_pool = courses[~courses.subject.isin(fav)].course_id.unique()
    explore_sample = rng.choice(explore_pool, size=min(n_explore,
len(explore_pool)), replace=False).tolist()
```

```
    sample = fav_sample + explore_sample
    if rng.random() < 0.5:
        sample += [cid for cid in popular_courses if cid not in sample]
    rng.shuffle(sample)
    return sample

# Генерація випадкового рейтингу
def random_rating(user_type: str, rng, exploration: bool = False) -> int:
    if user_type == "specialist":
        return rng.choice([2, 3, 4, 5], p=[.17, .26, .33, .24]) if exploration
else rng.choice([4, 5], p=[.37, .63])
    if user_type == "fan":
        if exploration and rng.random() < 0.7:
            return rng.choice([2, 3, 4, 5], p=[.23, .23, .32, .22])
        return 3
    if user_type == "multi":
        base = rng.choice([2, 3, 4, 5], p=[.13, .23, .32, .32])
        if exploration and rng.random() < 0.5:
            return rng.choice([2, 3, 4, 5], p=[.24, .26, .29, .21])
        return base
    return rng.choice([2, 3, 4, 5], p=[.22, .28, .27, .23]) if exploration and
rng.random() < 0.4 \
           else rng.choice([2, 3, 4, 5], p=[.18, .25, .30, .27])

# Генерація випадкової мітки часу
def random_timestamp(rng) -> str:
    year = rng.choice([2022, 2023, 2024, 2025])
    month = rng.choice([1, 5, 9]) if rng.random() < 0.75 else rng.integers(1,
13)
    day = int(np.clip(rng.normal(15, 7), 1, 28))
    h, m, s = rng.integers(0, 24), rng.integers(0, 60), rng.integers(0, 60)
    return f"{year}-{month:02d}-{day:02d} {h:02d}:{m:02d}:{s:02d}"
```

ГРАФІЧНІ МАТЕРІАЛИ



					КвРІПЗ. 2101083.01.11.ВД			
						Літ.	Маса.	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата	Назва роботи СИСТЕМА ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ ДЛЯ КОРИСТУВАЧІВ НАВЧАЛЬНОЇ ОНЛАЙН-ПЛАТФОРМИ			
Розробив		Первак І. Ю.		01.06		Аркуш 1	Аркушів 3	
Керівник		Форкун Ю. В.		01.06				
Консульт.						ХНУ, ІПЗ-21-1		
Н. Контр.		Яшина О. М.		01.06				
Зав. каф.		Бедратюк Л. П.		01.06				



КвРІПЗ. 2101083.01.11.ВД

Змн.	Арк.	№ докум.	Підпис	Дата
Розробив		Первак І. Ю.		01.06
Керівник		Форкун Ю. В.		01.06
Консульт.				
Н. Контр.		Яшина О. М.		01.06
Зав. каф.		Бедратюк Л. П.		01.06

Назва роботи: СИСТЕМА ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ ДЛЯ КОРИСТУВАЧІВ НАВЧАЛЬНОЇ ОНЛАЙН-ПЛАТФОРМИ

Літ.	Маса.	Масштаб
Аркуш 2	Аркушів 3	

ХНУ, ІПЗ-21-1

					КвРІПЗ. 2101083.01.11.ВД	
Змн.	Арк.	№ докум.	Підпис	Дата		
Розробив		Первак І. Ю.		01.06	Назва роботи: СИСТЕМА ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ ДЛЯ КОРИСТУВАЧІВ НАВЧАЛЬНОЇ ОНЛАЙН-ПЛАТФОРМИ	Літ. / Маса. / Масштаб
Керівник		Форкун Ю. В.		01.06		Аркуш 3 / Аркушів 3
Консульт.						
Н. Контр.		Яшина О. М.		01.06		ХНУ, ІПЗ-21-1
Зав. каф.		Бедратюк Л. П.		01.06		

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Первака Іллі Юрійовича
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.

01.06.2025
дата

підпис

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 1.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 13%**

ID: 243709 Title: БКР_Система персоналізованих рекомендацій для користувачів навчальної онлайн платформи Added in a DB: 2025-06-05 Authors: ПЕРВАК Ілля Heads: Форкун Ю.В., кад. техн.наук, доцент Consultants: Opponents:	Document		Sum coincidence on the DB	
	Symbols	Lexemes	Symbols	Lexemes
	103232	1540	2213 (2%)	33 (2%)

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: ПЕРВАК Ілля

Співавтор:

Назва: БКР Система персоналізованих рекомендацій для користувачів навчальної онлайнплатформи

Науковий керівник:

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:2.2%

Коефіцієнт подібності 2:0%

Мікропробіли: 0

Заміна букв: 1

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-06-04 14:26:15.0

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
04.06.2025

експерт

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Первак Ілля Юрійович

Тема Система персоналізованих рекомендацій для користувачів навчальної онлайн-платформи

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 82

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі проаналізовано предметну область онлайн-освіти та обґрунтовано доцільність створення системи персоналізованих рекомендацій. Визначено вимоги до системи, спроєктовано архітектуру на основі FastAPI, MongoDB і React, реалізовано гібридну модель рекомендацій (CB + CF + SBERT) та створено повний пайплайн генерації і обробки даних. Проведене тестування засвідчило коректність роботи системи та відповідність результатів очікуваним метрикам.

2. Висновок про відповідність роботи поставленому завданню _ Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням усіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі обґрунтовано актуальність теми, сформульовано мету та завдання роботи. У першому розділі проаналізовано предметну область онлайн-освіти, класифіковано підходи до персоналізації та визначено вимоги до системи. У другому розділі розглянуто архітектурні рішення, інформаційну модель і технологічний стек, на основі яких спроєктовано компонентну структуру системи. У третьому розділі описано реалізацію програмних модулів, включно з генерацією даних, навчанням моделі та тестуванням за top□k метриками. За підсумками тестування підтверджено функціональну придатність системи та відповідність її поведінки заданим вимогам

4. Позитивні сторони роботи Обрана тема є актуальною в контексті зростання попиту на персоналізовані освітні рішення. У роботі обґрунтовано вибір сучасної технологічної зв'язки (FastAPI, MongoDB, React), яка відповідає вимогам масштабованості, гнучкості та продуктивності. Реалізація системи ґрунтується на актуальних практиках розробки рекомендаторних сервісів, включаючи гібридну модель, обробку даних у форматі JSON, використання ембеддингів SBERT і контейнеризацію через Docker, що свідчить про високий рівень технічної підготовки автора.

5. Негативні сторони роботи Реалізація охоплює лише базову функціональність рекомендаторної системи; не передбачено інтеграції з реальними освітніми платформами, A/B□тестування або розширеної аналітики користувацької поведінки.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення відповідає тематиці роботи: діаграми, таблиці та рисунки логічно доповнюють текст і пояснюють архітектуру, інформаційну модель та алгоритмічні підходи. Пояснювальна записка оформлена відповідно до чинних стандартів, структура витримана, зміст подано послідовно й обґрунтовано.

7. Відгук про кваліфікаційну роботу в цілому Робота виконана на високому рівні та заслуговує на позитивну оцінку. Пояснювальна записка логічно структурована, зміст викладено чітко й послідовно, що сприяє легкому сприйняттю технічного матеріалу. Графічні матеріали доречно ілюструють ключові етапи проєктування системи, зокрема архітектуру, логіку обробки даних і сценарії взаємодії, посилюючи загальне враження цілісності та завершеності розробки.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі,
відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи) Дяшук Володимир Миколайович, доцент, ХНУ каф. Кібербезпеки

"03" 06 2025 р.

(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.
Назва кваліфікаційної роботи: «Система персоналізованих рекомендацій для користувачів навчальної онлайн-платформи»
Автор: Первак Ілля Юрійович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Форкун Юрій Вікторович, кандидат технічних наук, доцент

Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
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. Робота приймається до захисту, але має бути відкоригована.	
3	Виявлені запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Виявлені запозичення частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
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.	
5	Інше:	

Підтвердження:
Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:
1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
2) система фіксувала технічні особливості (наприклад, поєднання латиниці й українських індексів), а не модифікацію тексту.
Сумарний обсяг запозичень — 3.8%, що відповідає науковим стандартам і не впливає на якість кваліфікаційної роботи.

Дата 6.06.2024

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Юрій ФОРКУН