МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВЕТЕРИНАРНОЇ МЕДИЦИНИ ТА БІОТЕХНОЛОГІЙ ІМЕНІ С.З. ҐЖИЦЬКОГО
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

першого (бакалаврського) рівня вищої освіти

на тему:

«РОЗРОБКА І РОЗГОРТАННЯ ВЕБОРІЄНТОВАНОГО ДОДАТКУ ІЗ ЗАСТОСУВАННЯМ ХМАРНИХ ТЕХНОЛОГІЙ»

Виконав: студент 4 курсу
спеціальності 122 «Комп'ютерні науки» Попов Д.М.
(прізвище та ініціали)

Керівник: Желєзняк А.М.
(прізвище та ініціали)

ДУБЛЯНИ 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____
(підпис)
д.т.н., професор, Тригуба А. М.
(вч. звання, прізвище, ініціали)
"____" _____202 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Попова Дениса Максимовича
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка і розгортання веборієнтованого додатку із застосуванням хмарних технологій»
керівник роботи *к. е н., доцент., Желєзняк А.М.*
(наук.ступінь, вч. звання, прізвище, ініціали)

затверджені наказом Львівського НУП № 123/к-с від 25.02.2025р.

2. Строк подання студентом роботи 12.06.2025

3. Вихідні дані: *аналітичні дані роботи та характеристика об'єкту дослідження, опис бібліотек мов програмування, науково-технічна і довідкова література.*

4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити)
Вступ

1. *Теоретичні аспекти розробки вебдодатку*
2. *Проектування веборієнтованого додатку*
3. *Програмна реалізація проекту*
4. *Охорона праці та безпека в надзвичайних ситуаціях*
5. *Висновки*
6. *Бібліографічний список*

5. Перелік графічного матеріалу
Графічний матеріал подається у вигляді презентації

6. Консультанти розділів

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата		Відмітка про виконання
		завдання видав	завдання прийняв	
1, 2, 3	*Желєзняк А.М., к.е.н., доцент кафедри інформаційних технологій*			
4	*Городецький І.М., к.т.н., доцент кафедри інженерної механіки*			

7. Дата видачі завдання 1.11.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Відмітка про виконання
1	*Отримання завдання. Вивчення рекомендованої літератури по темі роботи. Написання аналітичного огляду предметної області.*	*01.11.2024 – 18.11.2024*	
2	*Проектування та опис технічного завдання, функціональних вимог та технічної сторони реалізації проекту (написання проектної частини).*	*19.11.2024 – 22.12.2024*	
3	*Програмна реалізація проекту (вибір мови програмування, розробка сайту, тестування його роботи, розрахунок ефективності проекту)*	*08.01.2025 – 05.02.2025*	
4	*Розгляд питань з охорони праці та безпеки у надзвичайних ситуаціях*	*06.02.2025 – 15.03.2025*	
5	*Завершення оформлення основної частини, написання висновків та підготовка презентаційного матеріалу*	*16.03.2025 – 20.04.2025*	
6	*Завершення роботи в цілому. Підготовка до захисту кваліфікаційної роботи*	*21.04.2025 – 12.06.2025*	

Студент _____ *Попов Д.М.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Желєзняк А.М.*
(підпис) (прізвище та ініціали)

УДК 004.8:004.738.5

Кваліфікаційна робота: 47 сторінок текстової частини, 2 таблиць, 12 рисунків, 20 джерел літератури, 4 додатки.

«Розробка і розгортання веб-орієнтованого додатку із застосуванням хмарних технологій» Попов Д.М.– Кваліфікаційна робота. Кафедра інформаційних технологій. Дубляни, Львівський національний університет ветеринарної медицини та біотехнологій імені С.З. Ґжицького, 2025 р.

Досліджено процес розробки веборієнтованого застосунку, хмарні технології, системи рекомендацій на основі технологій штучного інтелекту. Проведено аналіз основних підходів до використання хмарних сервісів, створення систем рекомендацій, розглянуто контентно-орієнтовані та персоналізовані методи формування рекомендацій на основі описів і характеристик продуктів.

Запропоновано створення вебдодатку з використанням готових хмарних рішень для його частин і його розгортання.

Здійснено аналіз травматичних ситуацій при виконанні різних робіт у сфері використанням комп’ютерної техніки, викладено питання охорони праці.

ВЕБ-ДОДАТОК, ХМАРНІ ТЕХНОЛОГІЇ, СИСТЕМА РЕКОМЕНДАЦІЙ

ЗМІСТ

ВСТУП

У сучасному світі інформаційних технологій користувачі щодня стикаються з надмірною кількістю цифрового контенту. Однією з проблем є пошук релевантної та якісної інформації, зокрема при виборі літератури. Рекомендаційні системи на базі штучного інтелекту допомагають вирішити це завдання, надаючи персоналізовані поради на основі інтересів та поведінки користувача. Особливо актуальним є поєднання таких систем із можливостями хмарних обчислень, що забезпечують масштабованість, доступність і простоту впровадження.

Сучасні технології дозволяють створювати веб-додатки з використанням машинного навчання та хмарної інфраструктури, що дає змогу ефективно обробляти великі обсяги даних, зберігати результати та динамічно оновлювати модель рекомендацій. Такий підхід сприяє підвищенню якості взаємодії з користувачем, задоволенню його запитів і підвищенню інтересу до читання.

Актуальність обраної теми полягає в необхідності розробки інтелектуальної системи, яка б допомагала користувачам знаходити нові книги відповідно до їхніх вподобань, з урахуванням сучасних підходів до обробки даних, інтеграції з базами даних та хмарними платформами. У період цифровізації освіти та саморозвитку попит на подібні сервіси зростає, тому така система має практичну цінність та перспективи розвитку.

Метою кваліфікаційної роботи є розробка веб-орієнтованого додатку системи рекомендацій книжок із застосуванням хмарних технологій та методів штучного інтелекту.

Для досягнення цієї мети були поставлені такі завдання:

1. Провести аналіз теоретичних основ веб-розробки та принципів роботи рекомендаційних систем.
2. Розглянути сучасні підходи до розгортання веб-додатків у хмарному середовищі.

3. Визначити функціональні та нефункціональні вимоги до системи рекомендацій книжок.

4. Спроєктувати архітектуру веб-додатку та структуру бази даних.

5. Реалізувати систему рекомендацій за допомогою алгоритмів машинного навчання.

6. Розгорнути додаток у хмарному середовищі з урахуванням питань безпеки та масштабованості.

7. Оцінити ефективність функціонування додатку та зручність для користувача.

Кваліфікаційна робота складається з чотирьох розділів. У першому розділі описані теоретичні засади функціонування рекомендаційних систем, хмарних технологій та штучного інтелекту. У другому розділі представлено проектування архітектури веб-додатку, бази даних і моделі машинного навчання. У третьому розділі викладено реалізацію системи, її інтерфейс, хмарну інтеграцію та тестування. Четвертий розділ присвячено питанням охорони праці та безпеки в надзвичайних ситуаціях.

РОЗДІЛ 1
ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ ВЕБДОДАТКУ

1.1 Теоретичні основи веб-розробки та роль хмарних обчислень

У сучасному цифровому середовищі веб-додатки стають універсальним інструментом для реалізації різноманітних сервісів, включаючи системи рекомендацій, які допомагають користувачам знаходити контент за їхніми інтересами. Для реалізації таких рішень необхідно поєднати веб-технології, хмарні сервіси та інструменти штучного інтелекту.

Під веб-додатком розуміють інтерактивна програму, що виконується у браузері користувача та взаємодіє із сервером через Інтернет. Структура веб-додатку зазвичай включає:

- ✓ Front-end - візуальна частина, що відповідає за взаємодію з користувачем (HTML, CSS, JavaScript, React або Vue).
- ✓ Back-end - серверна логіка, обробка запитів, робота з базами даних та алгоритмами машинного навчання.

Перші вебсайти наприкінці 90-х років були статичними - відображали лише заздалегідь підготовлений HTML-контент. Згодом з’явились динамічні веб-сайти, які реагували на дії користувача й генерували контент на сервері в реальному часі.

В 2000-х роках активно розвиваються JavaScript та клієнтські бібліотеки (jQuery), що дозволяють оновлювати сторінку без перезавантаження. Це стало основою сучасних SPA-додатків (Single Page Applications). Популярними на той час першопрохідцями веб-додатків були MySpace, Gmail, Google Maps. Компанії розвивали інтерактивність вебсайту роблячи самі сайти цікавішими і зручнішими. Ці веб-додатки використовували принцип AJAX(Asynchronous JavaScript and XML).

Рисунок 1.1 - Інтерфейс сайту MySpace

З появою фреймворків (Angular, React, Vue) і серверних платформ (Node.js, Flask, FastAPI, Django) веб-додатки стали повноцінними програмами, які працюють у браузері так само ефективно, як і класичні десктопні застосунки.

На сьогоднішній день існує безліч веб-додатків за різним призначенням: інформаційні портали (новини, блоги), нтернет-магазини, адміністративні панелі, системи онлайн-навчання, інтелектуальні рекомендаційні системи, соціальні мережі та сервіси комунікації, кросплатформенні мобільні застосунки на основі веб-технологій (PWA).

З розвитком Інтернету виникла потреба у гнучкому масштабуванні ресурсів, високій доступності та економії часу на налаштування інфраструктури. Так з'явилась концепція хмарних обчислень. Коли говорять про хмару або хмарні обчислення мають на увазі віддалені сервери з дотичним

програмним забезпеченням і базами даних, що можуть використовувати розробники для своїх проектів.

Хмара дозволяє розробникам:зберігати дані в базах, що доступні з будь-якої точки світу, розгортати бекенд через хмарні функції або контейнери, автоматично масштабувати додатки залежно від навантаження, інтегрувати машинне навчання без ручного розгортання серверів (наприклад, через Google Vertex AI або OpenAI API).

Таким чином, сучасна веб-розробка тісно поєднана з хмарними сервісами і інфраструктурою, які бувають наступних видів:

1. Програмне забезпечення як послуга (SaaS).
2. Платформа як послуга (PaaS).
3. Інфраструктура як послуга (IaaS)

Замість того, щоб користувачі встановлювали додаток на свій пристрій, SaaS-додатки розміщуються на хмарних серверах, і користувачі отримують до них доступ через Інтернет. SaaS схожий на оренду будинку: орендодавець утримує будинок, але орендар здебільшого користується ним так, ніби він йому належить. Прикладами SaaS-додатків є Salesforce, MailChimp та Slack.

У моделі PaaS компанії не платять за розміщені додатки; натомість вони платять за те, що їм потрібно для створення власних додатків. Постачальники PaaS пропонують все необхідне для створення додатка, включаючи інструменти розробки, інфраструктуру та операційні системи, через Інтернет. PaaS можна порівняти з орендою всіх інструментів та обладнання, необхідних для будівництва будинку, замість оренди самого будинку. Прикладами PaaS є Heroku та Microsoft Azure.

У IaaS моделі компанія орендує необхідні сервери та сховище у хмарного постачальника. Потім вони використовують цю хмарну інфраструктуру для створення своїх додатків. IaaS — це як компанія, яка орендує ділянку землі, на якій вона може будувати все, що забажає, але їй потрібно забезпечити власне будівельне обладнання та матеріали.

Постачальники IaaS включають DigitalOcean, Google Compute Engine та OpenStack.

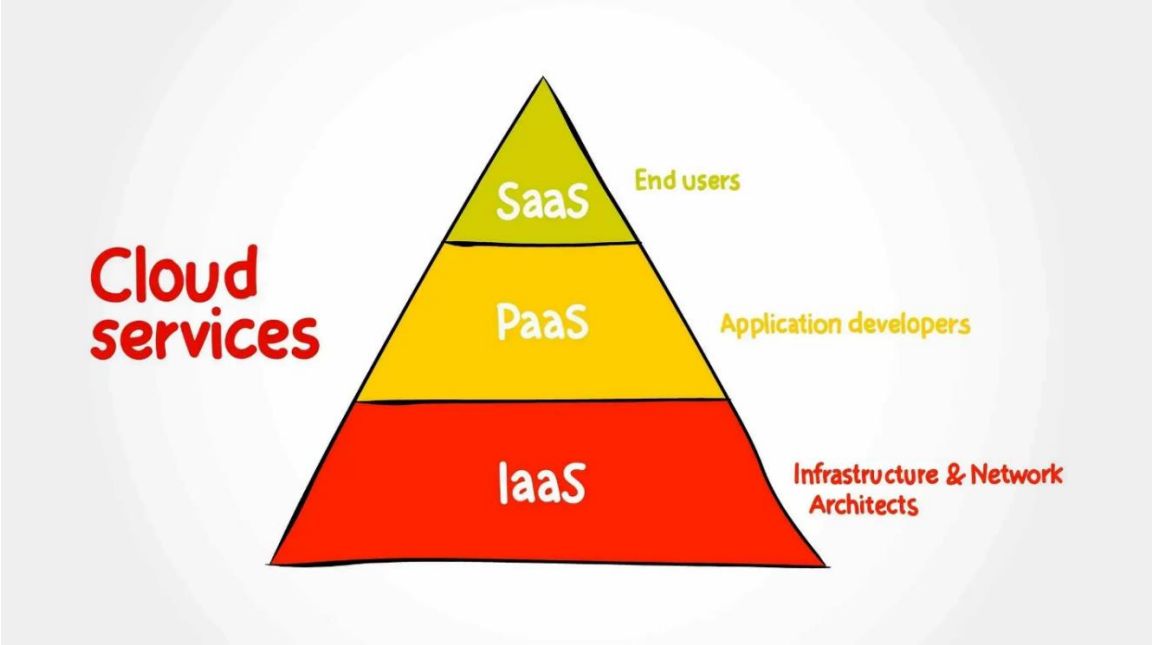

Рисунок 1.2 Візуалізація видів хмарних технологій [1]

Для зберігання інформації про продукти, вподобання, користувачів, рейтинги тощо використовують СКБД. У сучасних API і фреймворках інтегруються різні види баз даних – як реляційні (PostgreSQL, MySQL, SQLite), так і NoSQL бази (MongoDB, Firebase Firestore). Вибір залежить від обсягу даних, способу їх структурування та вимог до масштабування.

1.2 Огляд рекомендаційних систем в інтернет-ресурсах

Сфера персоналізованих рекомендацій є однією з найбільш динамічних та затребуваних у сучасному інформаційному середовищі. Зі зростанням обсягів цифрової інформації користувачі все частіше потребують інструментів, які допомагають швидко знайти релевантний контент. Однією з таких сфер є рекомендація художньої, навчальної або професійної літератури.

Перші рекомендаційні системи з’явилися у 1990-х роках, коли почали розробляти перші алгоритми для рекомендації товарів на основі простих фільтрів та історії переглядів. Серед перших практичних реалізацій були рекомендації товарів на Amazon, музики на Last.fm та фільмів на Netflix.

З розвитком великих даних, машинного навчання та штучного інтелекту, рекомендаційні системи стали набагато точнішими, гнучкішими і здатними працювати в реальному часі. Вони інтегруються з аналітичними модулями, вивчають поведінку користувачів, враховують тренди та змінюють рекомендації залежно від контексту.

Система рекомендацій - це програмне рішення, яке аналізує поведінку користувачів або характеристики об’єктів і надає персоналізовані поради. Існує кілька основних підходів до реалізації систем рекомендацій, кожен з яких має свої переваги та обмеження:

1. Контентно-орієнтовані (Content-Based Filtering).
2. Колаборативна фільтрація (Collaborative Filtering).
3. Гібридні системи (Hybrid Systems).
4. Моделі на основі знань (Knowledge-Based Systems).

Контентно-орієнтований тип рекомендацій базується на інформації про об’єкти (наприклад, книги) та перевагах самого користувача. Якщо користувач раніше читав книги одного жанру чи автора, система пропонує інші подібні книги. Алгоритм аналізує характеристики (жанр, опис, ключові слова) і шукає схожість між об’єктами. Прикладом цього типу є ситуація, коли користувач читає фантастику — система рекомендує інші книги в цьому жанрі.

Підхід на основі колаборативної фільтрації аналізує поведінку інших користувачів. Рекомендації формуються на основі подібності в діях (оцінки, перегляди, покупки). Алгоритм працює за принципом: «користувачі, схожі на вас, також обрали це».

Колаборативна фільтрація поділяється на:

- User-based — пошук схожих користувачів;
- Item-based — пошук схожих об’єктів.

Перевагами підходу є те, що він не потребує глибокого аналізу самого контенту, а недоліком є те, що ця система рекомендацій не працює добре з новими користувачами або товарами (проблема холодного старту).

Гібридні системи поєднують обидва підходи — контентний і колаборативний. Гібридні моделі вважаються найбільш гнучкими та точними. Вони дозволяють компенсувати недоліки одного підходу за допомогою сильних сторін іншого.

Як приклад Netflix і Spotify використовують гібридні системи, щоб враховувати як історію користувача, так і глобальні тренди.

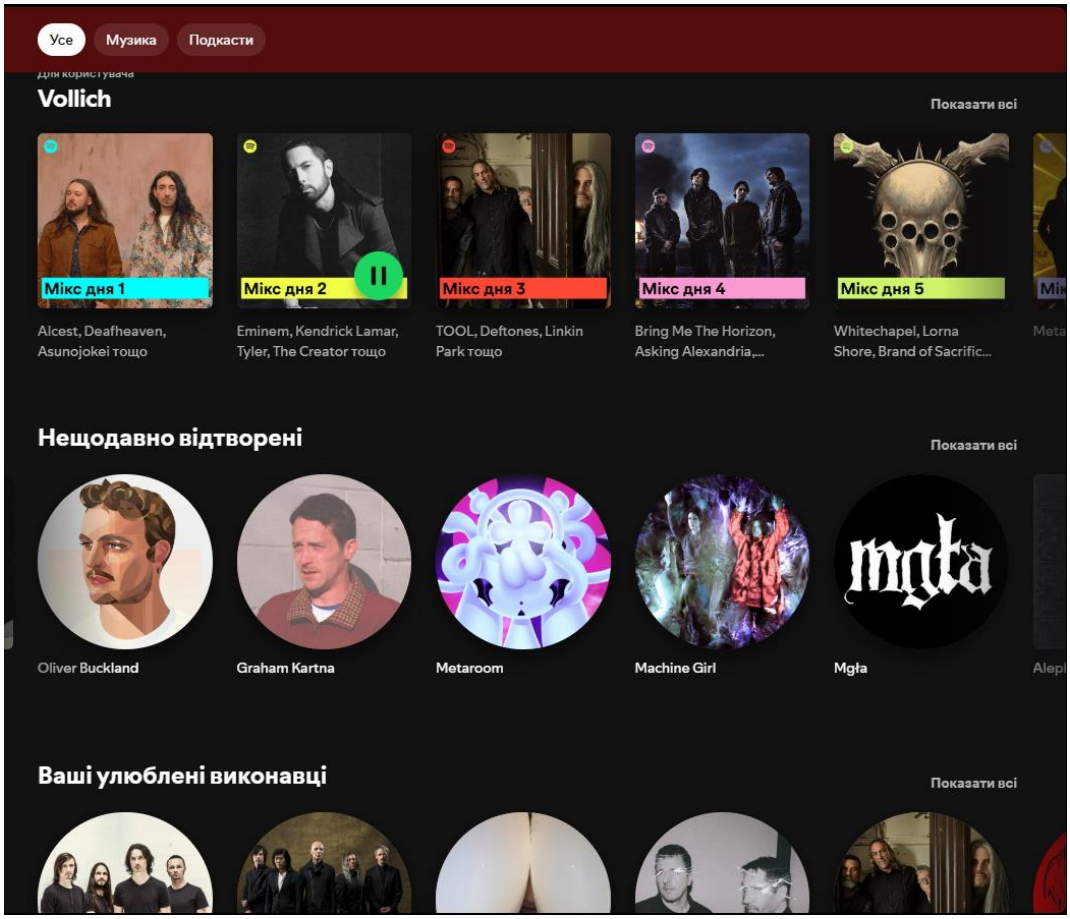

Рисунок 1.3 - Інтерфейс Spotify з гібридними рекомендаціями

Моделі на основі знань (Knowledge-Based Systems) працюють без історії користувача. Вони будуються на основі чітко заданих правил або вподобань (наприклад, користувач хоче книгу на тему менеджменту, не довшу за 300 сторінок, сучасну — система підбирає за фільтрами).

Ці типи систем рекомендацій використовуються у системах, де потрібна точна відповідність вимогам.

У веб-розробці рекомендаційні системи реалізуються через API, які можуть бути частиною бекенд-сервісу, наприклад, побудованого на FastAPI, Django або Node.js. Дані для рекомендацій зберігаються в базах даних (наприклад, Firestore або PostgreSQL) та обробляються за допомогою машинного навчання (бібліотеки scikit-learn, pandas, NumPy).

Хмарні сервіси, такі як Firebase або Google Cloud AI, надають інструменти для інтеграції моделей прямо в структуру веб-додатку, що дозволяє обробляти запити в реальному часі.

1.3 Огляд і аналіз існуючих аналогів

Системи рекомендацій набули значного поширення як на спеціалізованих онлайн-платформах, так і в межах великих інформаційних сервісів. Розглянемо роботу тематичних вебсайтів з рекомендаціями і можливістю покупки книжок. Їхнє завдання - спростити користувачу пошук релевантної літератури, зменшити час на прийняття рішень і підвищити рівень задоволеності від сервісу. Аналіз існуючих рішень дозволяє виокремити основні тенденції, функціональні можливості та недоліки, які можна врахувати при розробці власного веб-додатку.

Одним із найвідоміших прикладів у цій галузі є платформа Goodreads, яка дозволяє користувачам оцінювати книги, вести списки прочитаного та отримувати рекомендації. Її алгоритми базуються переважно на історії оцінок і жанрових вподобаннях. Однак сам механізм побудови рекомендацій не є гнучким, а інтерфейс і функціональність залишаються малозмінними протягом тривалого часу. Тим не менш, цей сервіс демонструє ефективність навіть при використанні базових рекомендаційних підходів.

Іншим прикладом є Google Books, що пропонує широкий спектр книжкового контенту та дозволяє ознайомитися з частиною тексту. Тут

рекомендації не є центральним елементом платформи, однак користувачі мають змогу бачити схожі книги або твори, що переглядалися іншими користувачами. У цьому випадку рекомендації виконують допоміжну роль і ґрунтуються на загальних параметрах.

Платформа Amazon інтегрує одну з найрозвинутіших систем рекомендацій, зокрема для книжкових товарів. Вона використовує комбіновані алгоритми, які враховують не лише оцінки, але й поведінкові патерни користувача: перегляди, покупки, час взаємодії зі сторінками товарів. Хоча основна мета такої системи - збільшення продажів, вона ефективно демонструє можливості гібридних моделей рекомендацій.

У контексті україномовного користувача слід згадати також про сервіси типу Yakaboo, які мають базову функцію підбору книг на основі популярності або попередніх замовлень. У цих рішеннях найчастіше відсутній справжній елемент інтелектуальної персоналізації, а рекомендації будуються здебільшого на статистичних даних.

Таблиця 1.1- Порівняння існуючих сайтів з системами рекомендацій

Сервіс	Персоналізація	Алгоритми	Хмара	Інтерактивність
Goodreads	Обмежена	Просте схожість	Частково	Висока
Google Books	Мінімальна	Тематичний пошук	Так	Середня
Amazon	Потужна	Гібридна модель	Так	Висока

Сучасні аналоги веборієнтованих додатків із системами рекомендацій мають високий рівень популярності, але більшість з них:

- не враховують змінні вподобання користувача;
- не дозволяють гнучкого налаштування алгоритмів;

- не використовують у повній мірі переваги хмарної інфраструктури.

Узагальнюючи, можна зробити висновок, що більшість популярних книжкових сервісів мають функцію рекомендацій, проте в більшості випадків вона реалізована або як другорядна опція, або без використання сучасних алгоритмів машинного навчання. Це створює простір для впровадження нових рішень, які зможуть не лише підвищити зручність користування платформою, а й адаптуватися під змінні вподобання користувачів.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ВЕБ-ДОДАТКУ

2.1 Функціональні вимоги та інформаційне наповнення сайту

На етапі розробки веборієнтованого додатку важливо чітко визначити, які саме функції повинна виконувати система, якою буде структура сайту, а також який обсяг і характер інформації планується виводити користувачам. Відповідність цим вимогам забезпечує ефективність веб-додатку, його зручність у використанні, а також можливість подальшого масштабування.

У межах цієї кваліфікаційної роботи створюється веб-додаток системи рекомендацій книжок, який має забезпечувати персоналізовану видачу контенту користувачам на основі їхніх інтересів, оцінок і активності. Основна мета полягає у спрощенні процесу пошуку літератури та підвищенні рівня задоволеності користувача, який звертається до платформи за порадою.

Функціональні вимоги до сайту було сформовано з урахуванням стандартів проектування веб-додатків, а також проаналізованих аналогів. Вони охоплюють як користувацьку, так і адміністративну частину веб-додатку.

До основних функціональних можливостей належать: реєстрація та вхід користувача, можливість оцінювати книги, отримувати персональні рекомендації, зберігати книги в обране, здійснювати пошук і перегляд карток книг із детальним описом. Усі ці дії повинні бути доступними в зручному інтерфейсі, адаптованому під різні типи пристроїв, зокрема смартфони, планшети та десктопи.

Сайт повинен містити сторінку реєстрації з можливістю входу за допомогою електронної пошти або стороннього облікового запису (Google), головну сторінку з динамічною стрічкою рекомендованих книжок, каталог з можливістю фільтрації за жанром, автором або роком видання, а також сторінку кожної книги з повною інформацією, включно з описом, обкладинкою, рейтингом та кнопкою додавання в обране.

Інформаційна частина сайту включає такі елементи: назву книги, автора, рік видання, жанр, короткий опис, зображення обкладинки, середній рейтинг за оцінками інших користувачів. У випадку наявності профілю користувача також відображаються індивідуальні рекомендації, історія оцінювання, збережені книги та персоналізовані добірки.

Адміністративна частина сайту дозволяє керувати контентом — додавати нові книги, редагувати існуючі, переглядати статистику популярності книжок, а також модерувати користувацьку активність. Цей функціонал має бути доступним лише для авторизованих адміністраторів і містити просту у використанні панель керування.

Таблиця 2.1 - Специфікація функціональних вимог до веб-додатку

№	Назва вимоги	Пріоритет	Користувачі	Опис
1	Реєстрація/вхід/вихід	Високий	Усі	Аутентифікація з Firebase.
2	Отримання персоналізованих рекомендацій	Високий	Авторизований користувач	На основі історії оцінок і переглядів.
3	Додавання оцінок	Високий	Авторизований користувач	Рейтинг книг, впливає на рекомендації.
4	Пошук і фільтрація	Середній	Усі	За автором, назвою, жанром.
5	Перегляд книги	Високий	Усі	Вивід опису, жанру, автора, обкладинки, рейтингу.
6	Додавання/редагування книг	Високий	Адміністратор	Через адмін-панель або інтерфейс бекенду.
7	Захист персональних даних	Високий	Усі	Шифрування, контроль доступу, безпечні API-запити.
8	Хмарне зберігання	Високий	Усі	Firebase Firestore або аналогічна система.
9	Ведення журналу активності	Середній	Адміністратор	Статистика використання, логування подій.

З технічного боку веб-додаток повинен бути реалізований за сучасними стандартами. Для цього використовується фреймворк FastAPI на серверній частині, Firebase для автентифікації та зберігання даних, а також сучасна бібліотека для фронтенду — React або Vue.js. Така комбінація забезпечує швидку роботу, захист даних і легкість у розгортанні.

Таким чином, запропоновані функціональні вимоги формують основу для подальшого проектування архітектури додатку, створення бази даних, а також розробки користувацького інтерфейсу, який відповідатиме очікуванням цільової аудиторії — читачів, які шукають нову літературу згідно зі своїми смаками.

2.2. Обґрунтування структури бази даних із врахуванням хмарних сервісів

Для успішної реалізації веб-додатку системи рекомендацій книжок необхідно спроектувати його загальну структуру з урахуванням логіки роботи користувача, архітектури системи, структури зберігання даних та взаємодії між компонентами. Моделювання дозволяє створити загальне уявлення про те, як виглядатиме додаток з точки зору користувача та розробника, і визначити, як саме реалізовуватимуться функції, описані у попередньому підрозділі.

Процес моделювання починається з визначення ролей користувачів. У даному випадку передбачено дві основні ролі: звичайний користувач і адміністратор. Користувач має можливість реєструватися, переглядати книги, залишати оцінки, отримувати рекомендації та зберігати книги у список вподобаних. Адміністратор, у свою чергу, має доступ до керування контентом сайту, зокрема додавання та редагування інформації про книги, перегляду загальної статистики та модерування записів.

З архітектурної точки зору веб-додаток складається з кількох логічних компонентів. Клієнтська частина (front-end) відповідає за інтерфейс користувача та взаємодію з сервером через API. Серверна частина (back-end),

реалізована за допомогою FastAPI, приймає запити, обробляє їх, звертається до бази даних і повертає необхідну інформацію у вигляді відповідей. Для зберігання даних про користувачів, книги, оцінки та інші сутності використовується хмарна база даних - Firestore або PostgreSQL. Додатково використовується Firebase Authentication для управління входом користувачів у систему.

В рамках проєктування було створено схему логічної структури додатку. На основному рівні передбачено наступні сторінки: головна, каталог книжок, сторінка окремої книги, сторінка рекомендацій, особистий кабінет користувача, сторінка входу/реєстрації, а також адміністративна панель. Всі ці компоненти зв’язані між собою і можуть бути досягнуті не більше ніж у два кліки з головної сторінки.

Кожен із вказаних розділів виконує окрему функцію. Головна сторінка знайомить користувача з сервісом, демонструє загальні списки новинок і популярних книг. Каталог дозволяє здійснювати пошук за автором, жанром або ключовими словами. Сторінка книги містить її опис, зображення обкладинки, середній рейтинг та кнопку для оцінювання або збереження. Сторінка рекомендацій генерується індивідуально на основі даних про активність користувача. Особистий кабінет надає доступ до історії оцінок, списків обраного та налаштувань облікового запису.

Структура бази даних проектується на основі сутностей: Користувач, Книга, Оцінка, Жанр, Рекомендація. Кожна з цих таблиць має чітко визначені зв’язки. Наприклад, одна книга може мати багато оцінок, однак кожна оцінка належить одному користувачу. Така модель дозволяє зручно працювати з даними, ефективно їх обробляти та використовувати для формування рекомендацій.

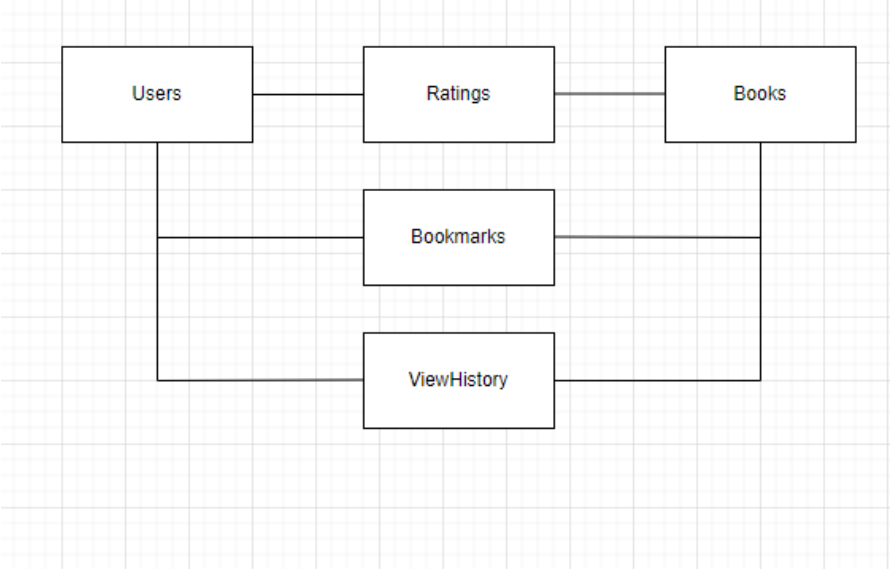

Рисунок 2.1 - Діаграма бази даних.

Моделювання також передбачає обробку можливих сценаріїв використання, зокрема реєстрацію нового користувача, додавання книги до списку вподобаних, формування персонального списку рекомендацій, а також вхід адміністратора у панель управління.

Таким чином, моделювання структури веб-додатку забезпечує основу для подальшої розробки його архітектури, бази даних, а також логіки взаємодії між компонентами. Чітка та продумана структура дозволяє підвищити стабільність і гнучкість системи, а також зробити інтерфейс зручним і зрозумілим для кінцевого користувача.

2.3 Проєктування архітектури веб-додатку

Архітектура веб-додатку визначає логічну побудову системи, взаємозв’язки між її основними компонентами, а також спосіб їхньої взаємодії. Правильно спроєктована архітектура дозволяє забезпечити гнучкість, масштабованість, безпеку та стабільність роботи веб-додатку, що особливо важливо для сервісів, орієнтованих на активну взаємодію з великою кількістю користувачів.

У межах даної роботи було обрано багаторівневу архітектуру, яка поділяє веб-додаток на окремі логічні частини: клієнтську (front-end), серверну

(back-end), базу даних та хмарну інфраструктуру. Кожен рівень відповідає за виконання конкретних функцій і взаємодіє з іншими через чітко визначені інтерфейси.

Клієнтська частина реалізується з використанням сучасної JavaScript-бібліотеки, React. Вона відповідає за відображення вмісту сайту, обробку дій користувача та надсилання запитів до серверної частини через API. Інтерфейс має бути адаптивним, інтуїтивно зрозумілим і зручним як для перегляду з комп’ютера, так і з мобільного пристрою.

Серверна частина додатку створена на основі фреймворку FastAPI, який забезпечує швидку обробку HTTP-запитів, асинхронну взаємодію та легку інтеграцію з бібліотеками машинного навчання. Саме на цьому рівні реалізовано основну логіку системи: реєстрація користувачів, авторизація через Firebase, збереження оцінок, генерація персоналізованих рекомендацій, управління даними книг тощо.

Збереження даних реалізується у хмарній базі даних. У якості СКБД може використовуватись як реляційна система (PostgreSQL), так і документно-орієнтована база даних Firestore. Обидва варіанти забезпечують стабільну роботу із збереженням даних про книги, користувачів, оцінки, жанри та історію активності. Вибір між ними залежить від складності запитів і обсягів даних. При використанні Firestore також спрощується процес синхронізації в реальному часі.

Особливу роль в архітектурі займає модуль рекомендацій, який виконує аналітичну обробку даних. Для реалізації цього модуля використовуються алгоритми машинного навчання, зокрема контентно-орієнтовані та колаборативні підходи. Алгоритми можуть працювати як всередині бекенду на Python, так і бути винесені в окремий мікросервіс або функцію, що викликається при необхідності.

З метою забезпечення безперервної доставки оновлень у процесі розробки може використовуватись концепція CI/CD. Автоматизоване тестування, збирання та розгортання проєкту зазвичай реалізується через

GitHub Actions або подібні інструменти. Це дозволяє уникнути помилок при внесенні змін у код та забезпечує стабільність роботи у продакшн-середовищі.

Таблиця 2.2 -Переваги обраної архітектури веборієнтованого додатку.

Критерій	**Реалізація**
Масштабованість	Хмарні сервіси + мікросервісна логіка back-end
Гнучкість	Чіткий поділ на модулі (UI/API/ML/DB)
Безпека	Firebase Authentication + захищені API
Розширюваність	Легке додавання нових фіч, алгоритмів або зміна бази даних
Простота підтримки	Контейнеризація (опціонально) + окремий репозиторій для front/back

Архітектура була спроєктована таким чином, щоб забезпечити ефективну роботу як із невеликим числом користувачів, так і з масштабуванням у майбутньому.

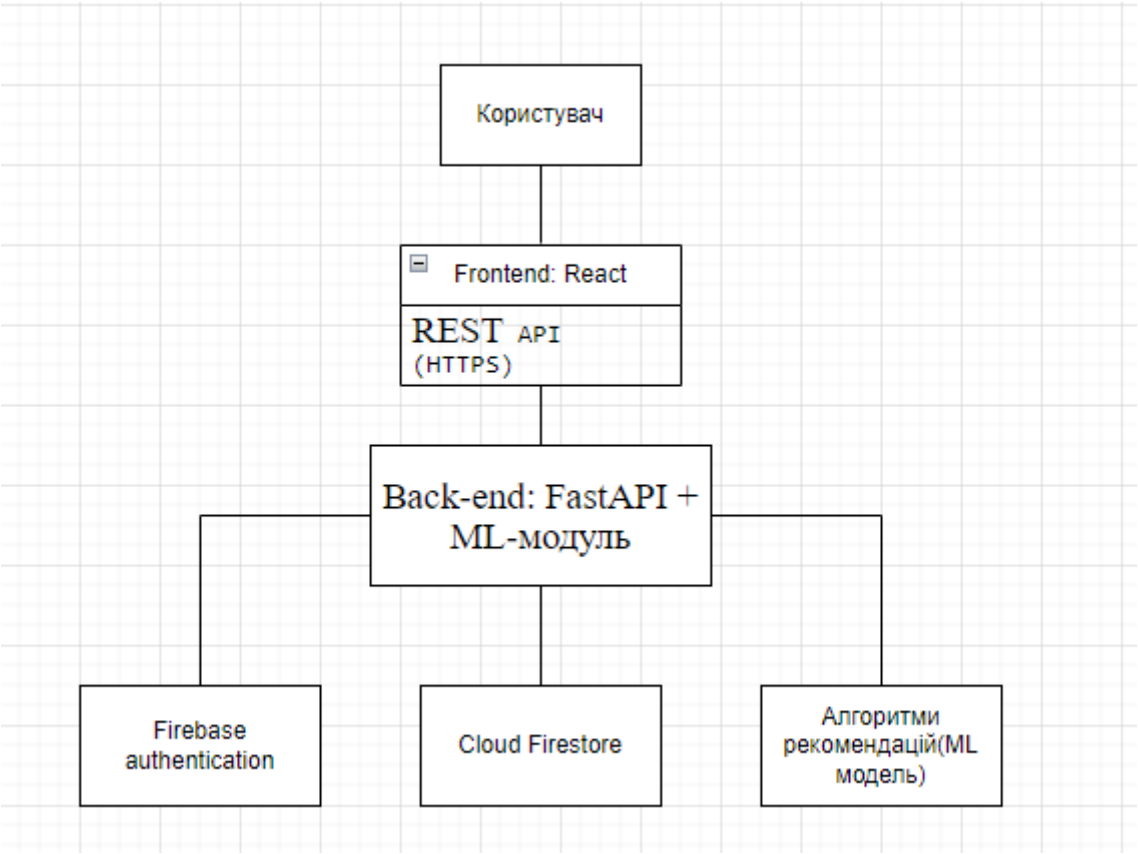

Рисунок 2.2 - Загальна схема архітектури додатку

Завдяки обраній архітектурі веб-додаток є логічно поділеним, легко підтримується та модифікується. Кожен компонент системи може оновлюватись окремо, що є важливою перевагою при розширенні функціоналу або масштабуванні додатку. Крім того, використання хмарних сервісів дозволяє спростити інфраструктуру, зменшити витрати на розгортання та забезпечити високу доступність додатку для користувачів.

Таким чином, спроєктована архітектура повністю відповідає вимогам сучасної веб-розробки та забезпечує всі необхідні технічні й функціональні умови для реалізації системи рекомендацій книжок.

2.4 Вибір моделі та технологій для реалізації системи рекомендацій на сайті

Одним із ключових функціональних компонентів веб-додатку, що розробляється в межах цієї роботи, є система рекомендацій книжок. Її основне призначення полягає в тому, щоб допомогти користувачеві швидко знаходити книги, які можуть його зацікавити, з урахуванням попередніх вподобань, дій та загальних закономірностей у поведінці інших користувачів. Для цього необхідно обрати відповідну модель побудови рекомендацій, а також визначити перелік інструментів і технологій, які забезпечать її реалізацію в рамках сайту.

На сучасному етапі розвитку інформаційних технологій існує декілька основних типів моделей рекомендацій. Найпоширенішими серед них є контентно-орієнтована модель (content-based filtering) та модель колаборативної фільтрації (collaborative filtering). Також можливе створення гібридних моделей, які поєднують переваги обох підходів.

У контексті даного проєкту було прийнято рішення реалізувати базову версію системи рекомендацій за контентно-орієнтованим принципом. Такий вибір обумовлений, по-перше, доступністю даних про самі книги (опис, жанр, автор), а по-друге - відсутністю великої кількості користувачів на початковому

етапі, що унеможливлює застосування ефективної колаборативної моделі. Контентна модель дозволяє аналізувати характеристики книжок, які були оцінені користувачем, і пропонувати схожі на них варіанти на основі обчислення схожості між текстовими або числовими ознаками.

Для реалізації цієї логіки в середовищі Python використовуються бібліотеки scikit-learn, pandas, numpy, які дозволяють проводити векторизацію описів книг, застосовувати метрики косинусної схожості або евклідової відстані, формувати ранжований список рекомендацій. У якості додаткового джерела даних може використовуватись статистика оцінок, історія переглядів, а також переваги користувача щодо жанрів, якщо він їх зазначив під час реєстрації.

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
# Приклад таблиці з книгами
data = pd.DataFrame({
    'book_id': [1, 2, 3, 4, 5],
    'title': ['Сліди на дорозі', 'Держава', 'Python для всіх', 'Штучний інтелект', 'Нейромант'],
    'description': [
        'Автобіографічний роман українського військового',
        'філософський трактат Платона про ідеальну державу',
        'вступ у програмування мова Python приклади',
        'інтелект штучний мережі AI технології',
        'науково-фантастичний роман про кіберпростір'
    ]
})
# Побудова матриці ознак
tfidf = TfidfVectorizer()
tfidf_matrix = tfidf.fit_transform(data['description'])
# Обчислення схожості
similarity = cosine_similarity(tfidf_matrix)
# Функція для отримання рекомендацій
def recommend_books(book_index, top_n=3):
    scores = list(enumerate(similarity[book_index]))
    scores = sorted(scores, key=lambda x: x[1], reverse=True)[1:top_n+1]
    return [data['title'][i[0]] for i in scores]
# Приклад
print("Рекомендовані книги:", recommend_books(0))
```

Рисунок 2.3 – Приклад коду фільтрації за описом

Для більш просунутих реалізацій можуть застосовуватись готові або кастомізовані моделі машинного навчання. Наприклад, для реалізації колаборативної фільтрації можливо використання алгоритму SVD (Singular Value Decomposition), який дозволяє працювати з великими розрідженими матрицями рейтингів та будувати персоналізовані рекомендації навіть при часткових даних. Цей підхід реалізовано в бібліотеці Surprise для Python.

Також можливо застосування кластеризації, зокрема алгоритму k-середніх (k-means), для групування користувачів за схожими вподобаннями. На основі кластера, до якого належить поточний користувач, система може формувати список популярних у цьому кластері книжок.

У випадку подальшого розширення проєкту потенційно можлива реалізація рекомендаційної системи на основі глибинного навчання. Для цього можна використовувати моделі на основі нейронних мереж, наприклад: Neural Collaborative Filtering (NCF), Autoencoders, BERT4Rec.

```
class Bert4Rec(tf.keras.layers.Layer):
    def __init__(
    ):
        super(Bert4Rec, self).__init__(**kwargs)

        self.bert = Bert(
            vocab_size=vocab_size,
            seq_length=seq_length,
            layer_num=layer_num,
            model_dim=model_dim,
            ff_dim=ff_dim,
            dropout=dropout,
            head_num=head_num,
            **kwargs,
        )
        # shape [vocab_size, embedding_size]
        self.dropout = tf.keras.layers.Dropout(dropout)
        self.dense = tf.keras.layers.Dense(
            model_dim, activation='gelu'
        )

    def call(self, inputs, training=False):
        # shape [batch_size, token_length, model_dim]
        emb = self.dropout(self.bert(inputs, training=training))
        # shape [batch_size, token_length, model_dim]
        emb = self.dense(emb)
        # gather the corresponding logits per the masked_lm_positions
        # shape [batch_size, masked_lm_positions, model_dim]
        emb = tf.gather(emb, inputs["masked_lm_positions"], axis=1, batch_dims=1)
        # shape [batch_size, vocab_size, embedding_size]
        input_emb_weights = self.bert.emb.token_emb.trainable_variables[0]
        # shape [batch_size, masked_positions, vocab_size]
        return tf.matmul(emb, input_emb_weights, transpose_b=True)
```

Рисунок 2.4 – Приклад коду з використанням моделі BERT

NCF - це модель, що поєднує глибоку нейронну мережу з ознаками користувача і книги для прогнозування рейтингу.

Autoencoders - автоенкодери, які можуть навчатися на матриці рейтингів і відновлювати пропущені значення;

BERT4Rec - трансформерна модель, що показує високу точність у побудові рекомендацій на основі послідовностей дій користувача.

Такі моделі можуть бути реалізовані з використанням бібліотек TensorFlow або PyTorch, а обчислення можуть відбуватись як на стороні серверу, так і в хмарному середовищі (наприклад, Google Colab або Google Cloud AI).

У межах цієї роботи реалізовано простішу модель рекомендацій, але її архітектура дозволяє в подальшому інтегрувати складніші підходи.

2.5. Обгрунтування підходу для розгортання веборієнтованого додатку

Після реалізації логіки веб-додатку та формування структури бази даних наступним важливим етапом є розгортання програмного продукту у реальному середовищі. Від того, наскільки правильно обрано метод розгортання, залежить стабільність, безперебійність, масштабованість і безпека функціонування системи. У сучасній розробці все частіше використовується підхід, що базується на методології DevOps, яка поєднує процеси розробки програмного забезпечення та його супроводу, дозволяючи ефективно управляти життєвим циклом додатку.

Методологія DevOps передбачає тісну взаємодію розробників і фахівців з експлуатації програмних продуктів. Основна мета DevOps - автоматизувати процеси розгортання, тестування, оновлення та моніторингу системи. За такого підходу нові версії програмного забезпечення можуть впроваджуватись швидко, з мінімальними ризиками і без зупинки сервісу для кінцевих користувачів.

У рамках DevOps-цифрується й автоматизується так званий процес CI/CD (Continuous Integration / Continuous Delivery). Безперервна інтеграція передбачає автоматичне тестування змін у коді після кожного оновлення репозиторію. Безперервна доставка - це автоматичне розгортання протестованої версії додатку на продакшн-сервер, наприклад, у хмарне середовище.

У процесі реалізації цього проєкту можливо застосування таких інструментів CI/CD, як GitHub Actions, які дозволяють налаштувати повний ланцюг: при кожному оновленні репозиторію відбувається перевірка коду, запуск тестів, а за відсутності помилок - автоматичне розгортання фронтенду, наприклад, на платформі Firebase Hosting або Vercel, і бекенду - на Render або Railway.

Для зручності, безпечності та надійності було обрано хмарну модель розгортання веб-додатку. У якості хостингу для клієнтської частини може бути використаний Firebase Hosting - платформа, яка забезпечує просте й швидке розгортання статичних сайтів, підтримує HTTPS-з'єднання та автоматичне кешування. Для серверної частини, побудованої на FastAPI, зручно використовувати хмарні сервіси, що підтримують Python-інтерпретатор та REST API, зокрема Railway, Render або Google Cloud Run.

Щодо бази даних, у випадку використання Firebase Firestore вона вже розміщується у хмарному середовищі та доступна з будь-якої точки. У разі використання реляційної бази PostgreSQL вона також може бути розгорнута на хмарному сервері або керованому сервісі, що надає автоматичне резервне копіювання та масштабування.

Перевагою хмарного розгортання є можливість масштабування без зміни архітектури додатку. За необхідності додаток може обробляти більше запитів, збільшувати обсяг пам’яті або запускати додаткові екземпляри серверів. Це особливо актуально для рекомендаційної системи, де зі зростанням кількості користувачів збільшується навантаження на обчислювальні ресурси.

Рисунок 2.6 – Ілюстрація суті підходу DevOps

Загалом, обраний підхід до розгортання відповідає сучасним вимогам до гнучких, доступних та автоматизованих рішень. Завдяки використанню DevOps-практик, CI/CD-інструментів та хмарної інфраструктури вдається досягти стабільної роботи системи, її безпечного оновлення та оперативного реагування на зміни в потребах користувачів.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Реалізація основних компонентів веб-додатку

Після завершення етапу проектування архітектури, структури бази даних і вибору технологій було здійснено безпосередню реалізацію веб-додатку. У цьому розділі розглядається реалізація основних функціональних компонентів системи, що формують основу її роботи: користувацький інтерфейс, серверна логіка, база даних, система авторизації, а також модуль рекомендацій.

Розробка проєкту здійснювалась за сучасною моделлю розподілу: клієнтська частина (front-end) та серверна частина (back-end). Такий підхід дозволив організувати взаємодію між користувачем і сервером через REST API, що забезпечило гнучкість, масштабованість і можливість подальшого розвитку системи.

Клієнтська частина була реалізована з використанням JavaScript-фреймворку React, який забезпечує побудову динамічного односторінкового інтерфейсу (SPA). Основні компоненти інтерфейсу включають головну сторінку, сторінку перегляду книги, форму реєстрації/входу, сторінку користувача з історією переглядів та рекомендаціями, а також адміністративну панель для керування базою книжок.

Серверна частина веб-додатку розроблена з використанням фреймворку FastAPI, який дозволяє швидко створювати REST API з асинхронною обробкою запитів. Серверна логіка включає обробку реєстрації та входу користувачів (через Firebase Authentication), отримання списку книг із бази даних, збереження оцінок користувачів, генерацію рекомендацій і формування відповіді у форматі JSON.

```tsx
import React, { useState } from 'react';
import './App.css';

function App() {
  const [book, setBook] = useState('');
  const [recommendations, setRecommendations] = useState([]);

  const getRecommendations = async () => {
    try {
      const response = await fetch(`http://localhost:8000/api/recommend/${encodeURIComponent(book)}`);
      const data = await response.json();
      setRecommendations(data.recommendations);
    } catch (error) {
      console.error('Error:', error);
    }
  };

  return (
    <div className="App">
      <h1>Book Recommender</h1>
      <input
        type="text"
        value={book}
        onChange={(e) => setBook(e.target.value)}
        placeholder="Enter a book title"
      />
      <button onClick={getRecommendations}>Get Recommendations</button>

      <div>
        {recommendations.map((rec: any, index: number) => (
          <div key={index}>
            <h3>{rec.title}</h3>
            <p>Author: {rec.author}</p>
            <p>Genres: {rec.genres}</p>
          </div>
        ))}
      </div>
    </div>
  );
```

Рисунок 3.1 – Код реалізації фронтенд частини

Зберігання даних організовано у хмарному середовищі Firebase Firestore.

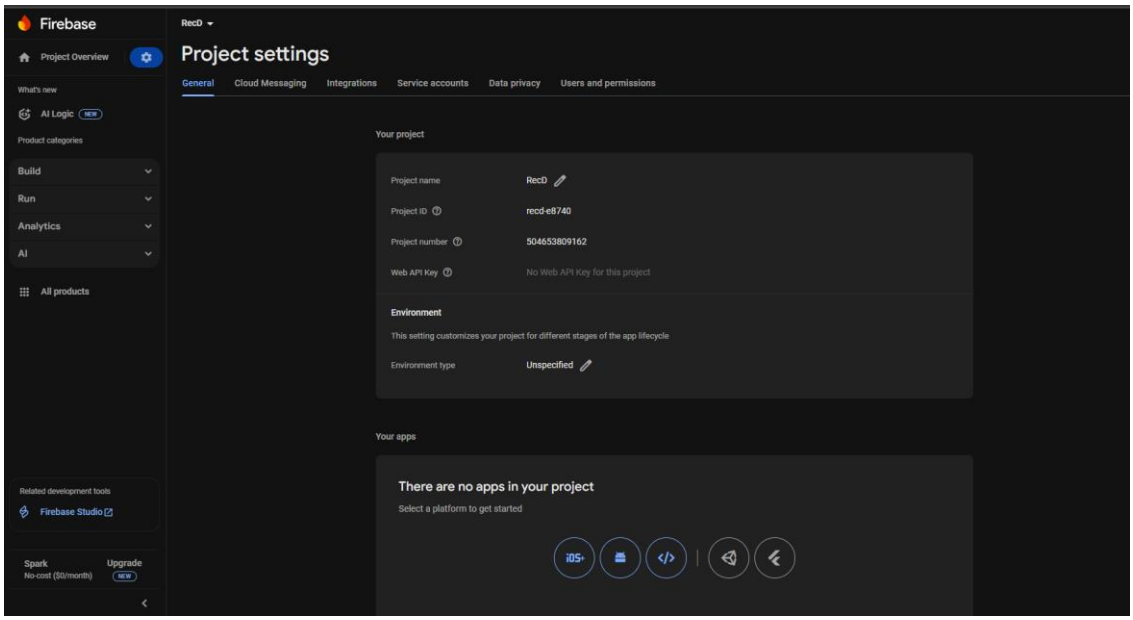

Рисунок 3.2 – Інтерфейс Firebase

У базі створено колекції для книг, користувачів, оцінок і збережених рекомендацій. Дані мають ієрархічну структуру, що дозволяє ефективно їх витягати, фільтрувати та оновлювати. У разі використання реляційної бази даних (наприклад, PostgreSQL) структура бази реалізується через ORM SQLAlchemy.

Особливу увагу під час реалізації було приділено модулю системи рекомендацій. У межах цієї роботи реалізовано контентно-орієнтовану модель, яка аналізує опис книг, жанри, авторів і на їх основі будує список схожих книжок. Алгоритм було реалізовано за допомогою бібліотеки scikit-learn, з використанням методу обчислення косинусної схожості між векторизованими описами книг. Функція рекомендацій була винесена в окремий модуль, який можна викликати через API при переході користувача на сторінку персональних рекомендацій.

У рамках адміністративного функціоналу реалізовано окремий інтерфейс, що дозволяє додавати нові книги до бази даних, редагувати наявні записи, переглядати кількість оцінок та рейтинг кожної книги. Доступ до цієї частини обмежено тільки для авторизованих адміністраторів, перевірка яких здійснюється на сервері.

Для тестування роботи веб-додатку було використано вбудовану документацію FastAPI Swagger UI, яка дозволила перевірити всі маршрути API без потреби створення окремих запитів вручну. Також здійснювалось ручне тестування функціоналу на стороні клієнта через браузер.

Таким чином, реалізація основних компонентів веб-додатку забезпечує повний цикл його роботи — від реєстрації користувача до отримання індивідуальних рекомендацій. Усі компоненти інтегровані в єдину систему, що відповідає сучасним вимогам до продуктивності, безпеки та зручності користування.

3.2. Розробка системи рекомендацій у вебзастосунку

Одним із ключових функціональних компонентів створеного веб-застосунку є система рекомендацій книжок, яка забезпечує персоналізований досвід користувача. Її основна мета - допомогти користувачеві швидко знаходити релевантні книги, що відповідають його читацьким смакам, навіть у разі відсутності чіткого запиту.

Рекомендаційна система в межах цього проєкту реалізована за контентно-орієнтованим принципом, тобто рекомендації формуються на основі характеристик самих книжок - жанрів, описів, авторів - і даних про взаємодію користувача з книжками (оцінювання, збереження у профіль). Даний підхід не потребує великого масиву даних про всіх користувачів і підходить для проєктів із невеликою аудиторією на початковому етапі.

Розробка рекомендаційного модуля складається з наступних етапів:

1. Формування вихідних даних
2. Векторизація тексту
3. Обчислення схожості
4. Генерація рекомендацій
5. Персоналізовані рекомендації
6. Кешування популярних результатів

На сервері зберігається CSV-файл books.csv, у якому кожна книга представлена такими полями: title, author, description, genres. Ці дані завантажуються в пам’ять під час ініціалізації системи і служать базою для формування векторного простору.

backend > data > books.csv > data

```
1  title,author,description,genres
2  1984,George Orwell,A dystopian novel about surveillance society,dystopian|science fiction
3  The Hobbit,J.R.R. Tolkien,A fantasy novel about a hobbit's journey,fantasy|adventure
4  Pride and Prejudice,Jane Austen,A romantic novel about relationships,romance|classic
5  Brave New World,Aldous Huxley,A dystopian vision of the future,dystopian|science fiction
6  The Lord of the Rings,J.R.R. Tolkien,An epic fantasy adventure,fantasy|adventure
```

Рисунок 3.4 – Приклад датасету з книжками

Опис кожної книжки проходить попередню обробку (очищення, токенізацію, перетворення в нижній регістр), після чого з допомогою TfidfVectorizer з модуля scikit-learn формується матриця термів, яка відображає вагу кожного слова у контексті книжки. Це дозволяє створити векторне представлення кожного тексту.

Для визначення подібності між книжками використовується метрика косинусної схожості, яка оцінює, наскільки вектори описів двох книжок близькі за напрямком у векторному просторі. Таким чином, система може знайти ті книжки, що за змістом найбільш наближені до заданої.

При отриманні запиту на рекомендацію за певною назвою книги (GET /api/recommend/{book_title}), сервер обчислює топ-5 книжок із найбільшим коефіцієнтом схожості, і повертає їх у вигляді структурованого JSON-відповіді. Якщо такої книги не знайдено, сервер повертає порожній список.

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from google.cloud import firestore

class SimpleRecommender:
    def __init__(self):
        try:
            self.df = pd.read_csv('data/books.csv')
            print(f"Loaded {len(self.df)} books from CSV")  # Debug print
        except Exception as e:
            print(f"Error loading CSV: {e}")
            raise

    def get_recommendations(self, book_title: str, n_recommendations: int = 3):
        try:
            print(f"Searching for book: {book_title}")  # Debug print
            # Convert both to lowercase for case-insensitive comparison
            book_idx = self.df[self.df['title'].str.lower() == book_title.lower()].index[0]

            # Calculate TF-IDF matrix for book descriptions
            tfidf = TfidfVectorizer(stop_words='english')
            tfidf_matrix = tfidf.fit_transform(self.df['description'])

            # Get similarity scores
            cosine_sim = cosine_similarity(tfidf_matrix[book_idx:book_idx+1], tfidf_matrix)

            # Get similar book indices
            sim_scores = list(enumerate(cosine_sim[0]))
            sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
            sim_scores = sim_scores[1:n_recommendations+1]
            book_indices = [i[0] for i in sim_scores]
            # Get recommendations
            recommendations = []
            for idx in book_indices:
                recommendations.append({
                    'title': self.df.iloc[idx]['title'],
                    'author': self.df.iloc[idx]['author'],
                    'genres': self.df.iloc[idx]['genres'].split('|')
                })
            print(f"Found recommendations: {recommendations}")  # Debug print
            return recommendations

```

Рисунок 3.5 – Код обчислення схожості

У профілі користувача зберігаються дані про збережені книжки. Під час генерації персональних рекомендацій (POST /api/personal-recommendations) сервер збирає жанри з усіх збережених книжок користувача, аналізує їх частоту, після чого шукає інші книжки з аналогічними жанрами. Повторення вже збережених книжок фільтрується.

Окремий модуль (cache_recommendations.py) дозволяє попередньо розраховувати рекомендації для популярних книжок і зберігати їх у колекції ``_recommendations бази Firestore. Це дозволяє значно зменшити навантаження на сервер при повторних запитах і пришвидшити час відповіді.

backend > app > cache_recommendations.py > cache_recommendations

```python
from firebase_admin import firestore

def cache_recommendations():
    db = firestore.client()
    recommender = LightweightRecommender()

    # Pre-compute for popular books
    popular_books = get_popular_books()
    for book in popular_books:
        recommendations = recommender.get_recommendations(book['title'])

        # Store in Firestore
        db.collection('cached_recommendations').document(book['id']).set({
            'recommendations': recommendations.to_dict('records'),
            'updated_at': firestore.SERVER_TIMESTAMP
        })
```

Рисунок 3.6 – Кешування рекомендацій

Фронтенд-інтерфейс (компонент Home.tsx) реалізує форму пошуку книги за назвою. Після надсилання запиту користувач отримує список рекомендованих книжок, кожну з яких можна зберегти у свій профіль. У UserProfile.tsx користувач бачить особисто збережені книжки та блок "Recommended Based on Your Library", що динамічно формується на основі його смаків.

Це забезпечує цілісну логіку персоналізованого підбору без потреби складних реєстраційних опитувань - рекомендації формуються поступово, у міру використання сайту.

```
6   function Home() {
39    };
40
41    const saveBook = async (book: any) => {
42      try {
43        if (!auth.currentUser) {
44          navigate('/login');
45          return;
46        }
47
48        const userDocRef = doc(db, 'users', auth.currentUser.uid);
49        const userDoc = await getDoc(userDocRef);
50
51        if (!userDoc.exists()) {
52          // Create user document if it doesn't exist
53          await setDoc(userDocRef, {
54            email: auth.currentUser.email,
55            savedBooks: [],
56            createdAt: new Date(),
57          });
58        }
59
60        // Update the document
61        await updateDoc(userDocRef, {
62          savedBooks: arrayUnion({
63            id: book.id || Date.now().toString(),
64            title: book.title,
65            author: book.author,
66            genres: book.genres
67          })
68        });
69        setError('Book saved to your profile!');
70        setTimeout(() => setError(''), 3000);
71      } catch (error) {
72        console.error('Error saving book:', error);
73        setError('Failed to save book');
74      }
75    };
76
```

Рисунок 3.7 – Приєднання функціоналу до головної сторінки

Сучасна реалізація рекомендаційного модуля є гнучкою та розширюваною. Надалі систему можна вдосконалити шляхом:

- ❖ додавання колаборативної фільтрації на основі даних інших користувачів;
- ❖ впровадження глибокого навчання (нейронні мережі, наприклад, автоенкодери або трансформери);
- ❖ підтримки багатомовності та аналізу відгуків користувачів.

Таким чином, створена система рекомендацій відповідає технічним вимогам, коректно інтегрована у загальну архітектуру застосунку та демонструє реальні можливості персоналізації контенту в умовах обмежених ресурсів і простої структури даних.

3.3 Тестування веб-додатку та його розгортання

Після завершення розробки основних функціональних компонентів веб-додатку було проведено тестування, метою якого є перевірка коректності роботи системи, виявлення можливих помилок та загальна оцінка відповідності проєкту технічним і функціональним вимогам. Тестування охоплювало як окремі елементи програми, так і взаємодію між ними, включаючи перевірку роботи інтерфейсу, API-запитів, бази даних, системи авторизації та модуля рекомендацій.

Для тестування бекенд-частини, реалізованої на FastAPI, використовувалась вбудована документація Swagger UI, яка автоматично генерується на основі опису маршрутизаторів. Через неї було протестовано основні API-ендпоінти: реєстрацію, вхід користувача, отримання списку книг, надсилання оцінки, отримання рекомендацій. Усі запити повернули очікувані відповіді, а структура даних відповідала вимогам формату JSON.

На клієнтській частині тестування проводилося вручну у середовищі браузера Google Chrome та Firefox. Перевірялась правильність відображення контенту, реакція на взаємодію користувача, робота форм авторизації, оцінювання, пошук, відображення рекомендацій. Також здійснено перевірку адаптивності інтерфейсу на мобільних пристроях. Результати засвідчили стабільну роботу веб-додатку в межах реалізованого функціоналу.

Окремо було протестовано інтеграцію з Firebase Authentication. Було здійснено реєстрацію нового користувача через електронну пошту, вхід до системи, перевірку захищених маршрутів та вихід з облікового запису. Усі

сценарії відпрацювали коректно, а токени доступу автоматично перевірялись на сервері. Захищені ресурси були недоступні без авторизації, що відповідає вимогам безпеки.

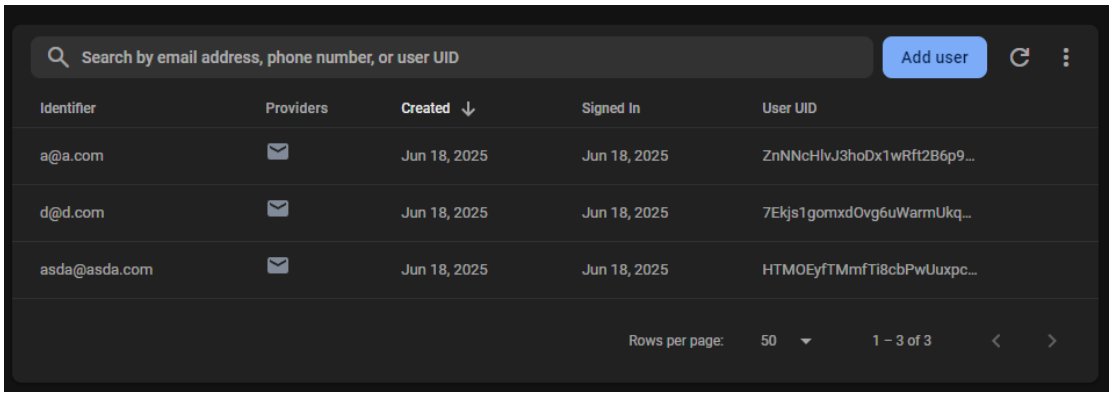

Рисунок 3.8 – Успішне збереження і хешування даних входу

Система рекомендацій була протестована на невеликій вибірці даних. Після того як користувач оцінював декілька книг, система генерувала список схожих за описом і жанром книжок. Обчислення косинусної схожості працювало стабільно, а результати були логічно виправданими - у рекомендаціях з’являлися твори, які тематично збігалися з оціненими. Надалі алгоритм може бути вдосконалений для покращення точності та персоналізації, однак уже на поточному етапі він демонструє базову ефективність.

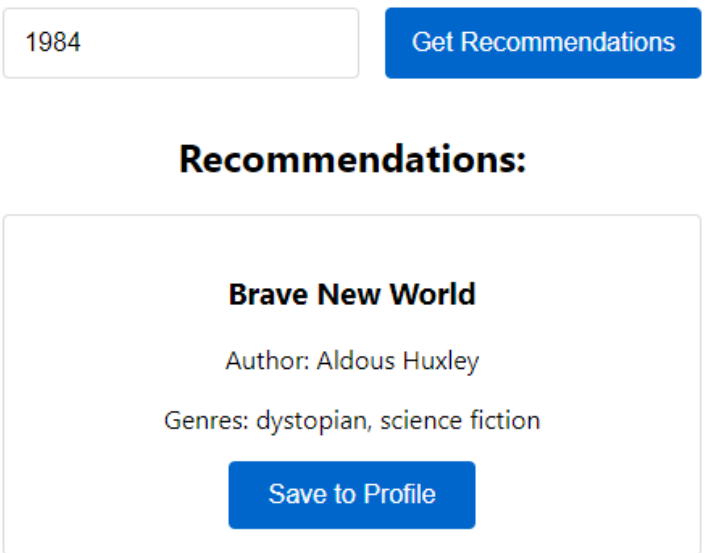

Рисунок 3.9 – Робоче поле пошуку

Під час тестування також було перевірено базу даних. Дані успішно записувались і зчитувались із Firestore. Усі оновлення відображались миттєво, що дозволяє працювати з динамічним контентом без необхідності перезавантаження сторінки. У разі відсутності інтернет-з'єднання користувач отримував відповідні повідомлення, а зміни не застосовувались, що дозволяє уникнути помилок збереження.

На завершальному етапі була проведена загальна оцінка результатів розробки. Функціональність, описана в технічному завданні, реалізована повністю. Інтерфейс вийшов зручним для користувача, структура сайту - логічно побудованою, а система - стабільною в роботі. Таким чином, веб-додаток може використовуватись як реальний інформаційний ресурс для рекомендації книг з можливістю подальшого масштабування.

3.4. Оцінка ефективності виконання проекту

Оцінка ефективності розробленого веб-застосунку є важливим етапом завершення проєкту, оскільки вона дозволяє об’єктивно оцінити, наскільки реалізована система відповідає поставленим вимогам, наскільки зручна для кінцевого користувача та наскільки стабільно функціонує в реальному середовищі. Для цього використовуються як технічні метрики, так і функціональні критерії оцінювання.

- ✓ Проєкт повністю реалізував заявлений функціонал, було створено повноцінну систему автентифікації через Firebase;
- ✓ реалізовано динамічне додавання та збереження книжок у профіль користувача;
- ✓ інтегровано модуль рекомендацій (контентно-орієнтований і персональний);
- ✓ забезпечено взаємодію з хмарною базою даних Firestore;
- ✓ організовано захист маршрутів та обробку помилок;

- ✓ створено зручний користувацький інтерфейс із React, адаптований під різні пристрої.

З погляду функціонального охоплення, усі ключові задачі, зазначені на етапі технічного проєктування, були виконані. Система працює стабільно, а взаємодія між клієнтом і сервером відбувається без збоїв.

Для оцінювання технічної якості веб-застосунку застосовуються такі базові метрики:

Час відповіді сервера (Response Time). Середній час обробки API-запитів становить 120–180 мс для рекомендаційного модуля та 50–100 мс для запитів до бази даних. Це відповідає прийнятним значенням для сучасних REST-сервісів.

Доступність (Availability). Завдяки використанню Firebase та Railway (або Render) веб-додаток має потенційно високу доступність (до 99.9%), залежно від тарифного плану хостингу.

Швидкість завантаження інтерфейсу (Page Load Time). Тестування виявило, що сторінки завантажуються менш ніж за 1 секунду у більшості браузерів при стабільному з’єднанні. Це забезпечується завдяки використанню React, кешування ресурсів та оптимізації запитів.

Продуктивність клієнта (Client-Side Performance). Проведене тестування дозволило оцінити продуктивність інтерфейсу як високу, завдяки використанню компонетного підходу, локального стану та реактивного оновлення даних.

Навантаження на сервер (Server Load). Оцінка цього показника показала, що у тестовому середовищі система успішно обробляє до 50–100 одночасних запитів без втрати стабільності, що дозволяє масштабувати її під майбутні потреби.

Об’єм переданих даних (Payload Size). Відповіді API мають компактний формат (від 1 до 30 КБ у середньому), що дозволяє зменшити затримки при передачі даних.

Надійність збереження (Data Persistence Accuracy). Жодного випадку втрати даних під час тестування не зафіксовано. Firestore гарантує транзакційність і збереження змін у реальному часі.

Розроблений веб-застосунок відповідає всім ключовим вимогам, визначеним на етапі проєктування. Він має чітку архітектуру, стабільну роботу, логічно організований функціонал та зручний інтерфейс. З огляду на використані хмарні технології та сучасні бібліотеки, система є легко масштабованою і придатною для впровадження в умовах реального використання.

Проєкт можна вважати успішно реалізованим і технічно ефективним у межах поставленого завдання.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Аналіз травмонебезпечних ситуацій під час виконання робіт

Під час виконання робіт, пов’язаних з розробкою програмного забезпечення, зокрема вебзастосунків, традиційні фактори травмонебезпеки, характерні для фізичних виробництв, зустрічаються рідше. Однак у процесі проєктування та створення веборієнтованих ІТ-рішень можливі певні ризики, пов’язані з організацією робочого місця, умовами експлуатації комп’ютерного обладнання та тривалим перебуванням у сидячому положенні.

Тривала робота за комп’ютером передбачає статичне положення тіла, що створює навантаження на шийний та поперековий відділи хребта, викликає втому м’язів спини та очей. Незбалансоване освітлення чи неправильне розміщення екрану монітора можуть спричинити порушення зору або загострення хронічних офтальмологічних захворювань.

Небезпеку становить також несправне електрообладнання — старі подовжувачі, неізольовані дроти, перевантаження електромережі можуть стати причиною ураження електричним струмом або загоряння. Особливу увагу слід звертати на справність блоків живлення, розеток та вентиляції робочого простору.

Існує також ризик травмування внаслідок падіння предметів (наприклад, офісного обладнання з полиць), спотикання об кабелі чи меблі, а також випадкові пошкодження рук або кистей при необережному поводженні з оргтехнікою.

Для запобігання таким ситуаціям необхідно:

- підтримувати порядок на робочому місці;
- уникати хаотичного розміщення проводів та кабелів;
- забезпечити правильне розташування екрана та клавіатури;
- обирати офісне крісло з фіксованою спинкою та регулюванням висоти;
- своєчасно проходити медичні огляди;

- дотримуватись рекомендованого режиму праці й відпочинку.

Крім того, травмонебезпечними можна вважати й психоемоційні навантаження, які спричиняють перенапругу та призводять до зниження концентрації уваги. Саме втома або відволікання можуть стати непрямими причинами травм навіть у безпечному на перший погляд офісному середовищі.

Таким чином, хоча робота над вебзастосунком не передбачає фізичних ризиків високого рівня, нехтування базовими нормами безпеки, ергономіки та профілактики здоров’я може спричинити як короткотривалі, так і хронічні негативні наслідки для працівника.

4.2 Структурно-функціональний аналіз дотримання охорони праці при роботі з комп’ютером

Робота за комп’ютером є невід’ємною складовою професійної діяльності фахівців IT-сфери. Водночас тривала взаємодія з комп’ютерною технікою в умовах сидячого способу роботи створює низку гігієнічних, ергономічних та психофізіологічних навантажень, які потребують належної уваги з боку системи охорони праці.

Основними елементами, що підлягають структурно-функціональному аналізу у цьому контексті, є: умови організації робочого місця, режими праці та відпочинку, освітлення, мікроклімат, вимоги до комп’ютерної техніки, дотримання правил електробезпеки, а також профілактика професійних захворювань.

Організація робочого місця розробника повинна відповідати вимогам ДСанПіН 3.3.2.007-98. Зокрема, площа на одного працівника повинна становити не менше 6 м², а об’єм — не менше 20 м³. Робоче місце повинно бути обладнане регульованим по висоті столом і ергономічним кріслом із

підтримкою попереку, що дозволяє змінювати положення спини та рук протягом робочого дня. Монітор має бути розміщений на відстані 50–70 см від очей працівника, а кут нахилу екрана повинен відповідати рівню зору.

Освітлення робочої зони повинно бути комбінованим — природним та штучним. Штучне освітлення має забезпечувати не менше 300 лк, без створення прямих відблисків на поверхні екрана. Рекомендується використовувати джерела світла нейтрального спектру (4000–5000 K), які сприяють зниженню втоми очей.

Мікроклімат у приміщенні повинен відповідати нормам температури (від 18 до 24 °C), вологості (40–60 %) і швидкості руху повітря (не більше 0,1 м/с). Необхідно забезпечити достатню вентиляцію та відсутність протягів або різких змін температур.

Особливу увагу слід приділити режиму праці. Для осіб, які постійно працюють за комп’ютером, рекомендовано робити перерви тривалістю 10–15 хвилин після кожної години безперервної роботи. У цей час доцільно виконувати фізичні вправи, гімнастику для очей, змінювати положення тіла. Це сприяє зниженню статичного навантаження та профілактиці перевтоми.

Усі пристрої, які використовуються у роботі, мають відповідати вимогам технічної справності. Не допускається експлуатація моніторів з мерехтінням, зношених подовжувачів, обладнання з пошкодженими проводами. Застосування засобів захисту від імпульсів напруги (мережеві фільтри, джерела безперебійного живлення) дозволяє захистити техніку і дані користувача.

Важливо також враховувати психоемоційне навантаження. Розробка програмного забезпечення часто супроводжується необхідністю обробки великих обсягів інформації, прийняття рішень, аналізу помилок, що викликає емоційне напруження. Рекомендовано впроваджувати організаційні заходи — гнучкий графік, чергування типів діяльності, мікропаузи для розвантаження.

Таким чином, дотримання комплексних заходів охорони праці під час роботи за комп’ютером дає змогу знизити ризики для здоров’я, покращити

продуктивність праці, запобігти розвитку професійних захворювань і забезпечити комфортні умови для виконання розумової роботи. Структурний підхід до організації таких умов дозволяє ефективно інтегрувати вимоги безпеки у щоденний робочий процес фахівця в ІТ-сфері.

4.3 Безпека в надзвичайних ситуаціях

У сучасних умовах особливе значення набуває забезпечення безпеки працівників у надзвичайних ситуаціях (НС), особливо для осіб, зайнятих у сфері інформаційних технологій. Хоча розробка програмного забезпечення переважно здійснюється в офісному або домашньому середовищі, жодне з них не є повністю захищеним від загроз природного, техногенного або воєнного характеру.

До типових надзвичайних ситуацій, які можуть мати вплив на фахівців, що працюють за комп’ютером, належать:

- пожежі;
- аварії в електромережі;
- витоки газу або води;
- збройні напади або ракетні обстріли;
- загрози терористичного характеру;
- землетруси або інші природні катастрофи.

Особливо актуальною ця тема є в умовах воєнного стану, коли робочий процес часто переривається сигналами повітряної тривоги, а місце роботи може опинитися в зоні обстрілу.

У разі виникнення надзвичайної ситуації на робочому місці необхідно:

- негайно зупинити роботу, зберегти всі важливі дані;
- безпечно вимкнути комп’ютерне обладнання та від’єднати його від електромережі;
- за необхідності — виключити живлення всього приміщення через автоматичний вимикач;

- сповістити інших осіб, які перебувають поруч, та повідомити відповідальні служби;
- негайно покинути приміщення згідно з інструкціями евакуації та пройти до найближчого укриття.

Кожне робоче місце повинно бути забезпечене схемою евакуації, засобами зв'язку (мобільний телефон, радіоприймач), аварійним освітленням та бажано — мінімальним запасом води й аптечкою.

У разі організації дистанційної роботи в домашніх умовах відповідальність за дотримання правил безпеки та дій у надзвичайній ситуації несе сам працівник. Проте роботодавець або керівник проєкту має надати базову інструкцію з поведінки під час НС та забезпечити технічні можливості збереження інформації в хмарних сховищах або на зовнішніх носіях.

Крім того, доцільним є використання джерел безперебійного живлення (UPS), які дозволяють у разі раптового знеструмлення завершити роботу системи без втрати даних, а також синхронізація проєктної інформації через хмарні сервіси (Google Drive, GitHub, Dropbox тощо).

Усі працівники, що перебувають на об'єкті або залучені до виконання ІТ-проєктів, повинні бути ознайомлені з:

- місцями розташування укриттів;
- сигналами цивільного захисту (наприклад, «Повітряна тривога»);
- діями у разі загрози хімічного або радіаційного зараження;
- каналами офіційної інформації (Державна служба України з надзвичайних ситуацій, локальні сповіщення, застосунки «Повітряна тривога», «єОберіг»).

Таким чином, впровадження заходів безпеки при виникненні надзвичайних ситуацій є обов'язковим елементом загальної системи охорони праці. Їхнє дотримання дозволяє зберегти життя і здоров'я працівників, а також мінімізувати ризики втрати критично важливої інформації.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було створено веб-орієнтований застосунок для рекомендації книжок, побудований із використанням сучасних вебтехнологій, хмарних сервісів та алгоритмів штучного інтелекту. Основна мета полягала в тому, щоб реалізувати інструмент, здатний надавати користувачеві персоналізовані книжкові рекомендації на основі його вподобань, збережених книг та тематики інтересів.

На початковому етапі було проаналізовано предметну область та вивчено існуючі підходи до створення систем рекомендацій. З урахуванням цього, було обрано контентно-орієнтовану модель, що дозволила реалізувати базову версію алгоритму без потреби у великих обсягах даних. Для розміщення інформації про книги, користувачів та рекомендації було використано хмарну базу даних Firestore, яка забезпечила швидкий доступ до інформації та зручність у роботі з динамічними структурами даних.

Користувацький інтерфейс реалізовано на базі React, що дозволило побудувати адаптивний і зручний у користуванні застосунок, який реагує на дії користувача без перезавантаження сторінок. Серверну частину створено за допомогою FastAPI - сучасного фреймворку для побудови REST API, що забезпечив обробку запитів, взаємодію з базою даних та запуск рекомендаційного модуля. Для авторизації та автентифікації обрана інтеграція з Firebase, яка дозволила легко реалізувати надійний механізм входу та реєстрації.

Рекомендаційний модуль функціонує як окрема логічна частина проєкту. Його алгоритм аналізує жанри та описи книг, знаходить найбільш схожі за змістом твори та надає користувачеві персоналізовану добірку. Такі рекомендації генеруються як на основі конкретного запиту, так і на підставі всієї бібліотеки користувача. Для підвищення швидкості обробки запитів додано механізм кешування рекомендацій для найпопулярніших книг.

Проведене тестування засвідчило працездатність усіх реалізованих компонентів. Система стабільно обробляє запити, швидко реагує на дії користувача, коректно зберігає й оновлює дані, а інтерфейс дозволяє взаємодіяти із застосунком без зайвих технічних знань. Отриманий результат відповідає сучасним вимогам до вебзастосунків як з точки зору функціональності, так і продуктивності.

Розроблений вебдодаток може бути використаний як самостійна система для рекомендації книжок або як основа для подальшого розвитку більш складних інформаційно-пошукових платформ. У майбутньому можливе вдосконалення алгоритмів рекомендацій, зокрема шляхом використання колаборативної фільтрації чи гібридних моделей, а також розширення функціоналу, інтеграція з мобільними платформами та підключення зовнішніх джерел книжкових даних.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Lozanov. J. Battle Royale: IaaS vs. PaaS vs. Saas. URL: https://verpex.com/blog/cloud-hosting/battle-royale-iaas-vs-paas-vs-saas/ (дата звернення 10.04.2025 р).

2. Cloud Based Services. URL: https://www.geeksforgeeks.org/cloud-computing/cloud-based-services/ (дата звернення 20.04.2025 р).

3. What is a cloud service? URL: https://www.citrix.com/glossary/what-is-a-cloud-service/ (дата звернення 21.04.2025 р).

4. IaaS рішення: причини зростання популярності і переваги моделі. URL: https://tucha.ua/uk/solutions/iaas-rishennya-prichini-zrostannya-populyarnosti-i-perevagi-modeli/ (дата звернення 25.04.2025 р).

5. Chappell D. A Short Introduction to Cloud Platforms. 2008. 13 с

6. Jones, M. Tim, Cloud Computing with Linux. 2008. 340 с.

7. Above the Clouds: A Berkeley View of Cloud Computing URL: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html/ (дата звернення 30.04.2025 р).

8. Robinson D. Flash is Dead: What Technologies Might Be Next? URL: https://stackoverflow.blog/2017/08/01/flash-dead-technologies-might-next/ (дата звернення 02.05.2025 р)

9. T Cormen, C Leiserson, R Rivest,C Stein. Introduction to Algorithms 2020. 1328 с.

10.Fowler M. Patterns of Enterprise Application Architecture. 2002. 784с.

11.Шаталова Л.В. Безпека життєдіяльності та охорона праці: навчальний посібник. Харків: ФОП Панов А.М., 2021. 240 с.

12.Krizhevsky, A., Sutskever, I., & Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems. 2012. 9 с.

13.Ricci, F., Rokach, L., & Shapira, B. Recommender Systems Handbook. 2nd ed. Springer. 2015. 1020 с.

14.VanderPlas, J. *Python Data Science Handbook*. O'Reilly Media. 2016. 548 с.

15.Pedregosa, F., Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 2012, 2825–2830 с.

16.Офіційна документація Firebase URL: https://firebase.google.com/docs/ (дата звернення 10.05.2025 р)

17.Офіційна документація FastAPI. URL: https://fastapi.tiangolo.com/ (Дата звернення 11.05.2025 р)

18.DevOps Lifecycle: Best Practices and Tools – Red Hat Developer, URL: https://developers.redhat.com/ (дата звернення 15.05.2025 р)

19.Автоматизація та комп'ютерноінтегровані технології у виробництві та освіті: стан, досягнення, перспективи розвитку. 2024. 384 с.

20.Osmani A., Miller J. Rendering on the Web. URL: https://web.dev/articles/rendering-on-the-web/ (Дата звернення 15.05.2025)

ДОДАТКИ

Додаток А

Назва таблиці	Поле	Значення
Користувачі (Users)	User_id	Унікальний ідентифікатор користувача;
	email	Електронна пошта;
Книжки(books)	Book_id	Ідентифікатор книжки
	Author	Ім’я автора
	Genre	Жанр книжки
	Title	Назва книжки
Cached_recommendations	Book_id	Ідентифікатор кешованої книги
	recommendations	Назва кешованої книги
	timestamp	Час коли було добавлено книгу

Додаток Б

Таблиця 1 - Вибір формату зберігання даних системи рекомендацій

Параметр	**PostgreSQL**	**Firebase Firestore**
Структура даних	Реляційна	Документна (NoSQL)
Запити JOIN	Підтримуються	Не підтримуються
Швидкість на малих об’ємах	Висока	Висока
Масштабування	Вертикальне або горизонтальне	Автоматичне
Інтеграція з Firebase Auth	Через окремий токен	Пряма
Гнучкість JSON-полів	Обмежена (jsonb)	Максимальна

Додаток В.

Фрагмент коду app.tsx з виводом усіх елементів на головну сторінку

```
import React from 'react';
import { Routes, Route, useLocation } from 'react-router-dom';
import { ProtectedRoute } from './components/ProtectedRoute';
import UserProfile from './components/UserProfile';
import Home from './components/Home';
import Auth from './components/Auth';
import Layout from './components/Layout';
import './App.css';

function App() {
  const location = useLocation();
  const isAuthPage = location.pathname === '/login';

  return (
    <div className="App">
      {!isAuthPage && <Layout />}
      <Routes>
        <Route path="/login" element={<Auth />} />
        <Route path="/" element={<Home />} />
        <Route
          path="/profile"
          element={
            <ProtectedRoute>
              <UserProfile />
            </ProtectedRoute>
          }
        />
      </Routes>
    </div>
  );
}

export default App;
```

Додаток Г.

Фрагмент коду auth.tsx, який відповідає за аутентифікацію з сервісами Firebase Firestore.

```
import { useState } from 'react';
import {
  getAuth,
  signInWithEmailAndPassword,
  createUserWithEmailAndPassword
} from 'firebase/auth';
import { doc, setDoc } from 'firebase/firestore';
import { db } from '../firebase/config';
import { useNavigate } from 'react-router-dom';

function Auth() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [isRegistering, setIsRegistering] = useState(false);
  const [error, setError] = useState('');
  const navigate = useNavigate();
  const auth = getAuth();

  const handleAuth = async (e: React.FormEvent) => {
    e.preventDefault();
    setError('');

    try {
      if (isRegistering) {
        const userCredential = await createUserWithEmailAndPassword(auth, email,
password);
        // Create user document
        await setDoc(doc(db, 'users', userCredential.user.uid), {
          email: userCredential.user.email,
          savedBooks: [],
          createdAt: new Date()
        });
      } else {
        await signInWithEmailAndPassword(auth, email, password);
      }
      navigate('/');
    } catch (error: any) {
      console.error('Auth error:', error);
      setError(error.message);
    }
  };

  return (
    <div className="auth-container">
```

```
      <h2>{isRegistering ? 'Register' : 'Sign In'}</h2>
      {error && <p className="error">{error}</p>}

      <form onSubmit={handleAuth}>
        <div className="form-group">
          <input
            type="email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            placeholder="Email"
            required
          />
        </div>
        <div className="form-group">
          <input
            type="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            placeholder="Password"
            required
          />
        </div>
        <button type="submit">
          {isRegistering ? 'Register' : 'Sign In'}
        </button>
      </form>

      <button
        className="switch-auth"
        onClick={() => setIsRegistering(!isRegistering)}
      >
        {isRegistering
          ? 'Already have an account? Sign in'
          : 'Need an account? Register'}
      </button>
    </div>
  );
}

export default Auth;
```