

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проекту (роботи)

магістра

(ступінь вищої освіти)

на тему ІНТЕРАКТИВНА ПЛАТФОРМА РЕКОМЕНДАЦІЙ ЩОДО
ВИБОРУ МЕНТОРА ІЗ КРИТЕРІАЛЬНОЮ ОЦІНКОЮ

Виконав: студент 2 курсу, групи КНТ-514м
спеціальності

123 Комп'ютерна інженерія

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні системи та мережі

СОЛОМКА А.С.

(ПРИЗВИЩЕ та ініціали)

Керівник ЗЕЛЕНЬОВА І.Я.

(ПРИЗВИЩЕ та ініціали)

Рецензент ХАРИТОНОВ О.Б.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти магістерський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні системи та мережі
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Зав. кафедри Кудерметов Р.К.

“ 15 ” жовтня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

СОЛОМКИ Артура Сергійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Інтерактивна платформа рекомендацій щодо вибору ментора із критеріальною оцінкою

керівник проєкту (роботи) кан. техн. наук, доцент, ЗЕЛЕНЬОВА Ірина Яківна

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом вищого навчального закладу від “30” жовтня 2025 року № 491

2. Строк подання студентом проєкту (роботи) 23 грудня 2025 року

3. Вихідні дані до проєкту (роботи) наявна бібліотека для системного аналізу WEKA

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Концептуальний аналіз;

2) Моделювання та проєктування системи;

3) Архітектура системи;

4) Програмна розробка системи;

5) Дослідження системи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Слайди презентації.

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-6	ЗЕЛЕНЬОВА І. Я., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 01.10.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Срок виконання етапів проєкту (роботи)	Примітка
1	Визначення проблеми, оцінка обмежень й ризиків, а також аналіз існуючих рішень	16.10.2025 р.	
2	Моделювання системи з визначенням вимог та обранням інструментів	21.10.2025 р.	
3	Розробка та обґрунтування підходів оцінки та рекомендацій	31.10.2025 р.	
4	Обрання та опис архітектури системи	01.11.2025 р.	
5	Програмна розробка системи	10.11.2025 р.	
6	Дослідження системи	24.11.2025 р.	
7	Оформлення отриманих результатів у ПЗ	01.12.2025 р.	
8	Оформлення графічного матеріалу	08.12.2025 р.	
9	Оформлення допоміжного матеріалу	15.12.2025 р.	

Студент Артур СОЛОМКА
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник проєкту (роботи) Ірина Зеленцова
(підпис) (ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 88 сторінки, 19 рисунка, 8 таблиць, 25 джерел

ANGULAR, JAVA, REST, SPRING, WEKA, АРХІТЕКТУРА, БАЗА ДАНИХ, КЛІЄНТ, МЕНТОР, МЕНТІ, МЕТРИК, МІКРОСЕРВІС, КРИТЕРІЇ, СЕРВЕР, СТАТИСТИЧНИЙ АНАЛІЗ

Метою роботи є поліпшення якості автоматизованого процесу щодо підбору наставників за необхідними навичками на основі результативності їх роботи.

Наукова новизна роботи полягає в тому, що розроблено комплексну методику для оцінки наставників та формування рекомендацій для їх вибору за необхідними навичками на основі результативності їх роботи з підтримкою контролю успіхів учня. Використання прогностичних характеристик при виборі наставників дає змогу формувати більш точну оцінку потенційного результату наставництва ще до його початку, що суттєво підвищує оптимальність процесу їх підбору.

Практична цінність результатів роботи полягає в тому, що розроблено інтерактивну систему, яка дає змогу здійснювати підбір наставників на основі визначених критеріїв та навичок, що потребують розвитку; акумулювати статистику як про діяльність менторів, так і про прогрес учня; забезпечити об'єктивність процесу навчання завдяки можливості відстеження динаміки змін у рівні компетентностей учня.

ABSTRACT

Explanatory note: 88 pages, 19 figures, 8 tables, 25 references

ARCHITECTURE, DATABASE, CLIENT, MENTOR, MENTEES, METRICS, MICROSERVICE, CRITERIA, SERVER, STATISTICAL ANALYSIS, ANGULAR, JAVA, REST, SPRING, WEKA

The goal of the work is to improve the quality of the automated process for selecting mentors based on required skills, according to the effectiveness of their work.

The scientific novelty of the work lies in the development of a comprehensive methodology for evaluating mentors and generating recommendations for their selection according to the required skills based on their performance while supporting monitoring of the student's success. The use of predictive characteristics when selecting mentors allows for a more accurate assessment of the potential outcome of mentorship even before it begins, which significantly increases the optimality of the selection process.

The practical significance of the results lies in the development of an interactive system that enables the selection of mentors based on defined criteria and skills that require development; accumulation of statistics both on mentor activity and student progress; and ensures the objectivity of the learning process by allowing the tracking of changes in the student's competency levels.

ЗМІСТ

Вступ.....	8
1 Концептуальний аналіз	9
1.1 Визначення та формулювання проблеми	9
1.2 Аналіз існуючих рішень та технологій	10
1.2.1 Освітні онлайн платформи з менторською підтримкою	10
1.2.2 Освітні спільноти та платформи менторства	11
1.2.3 Корпоративні системи навчання	11
1.2.4 Системи управління навчання	11
1.2.5 Узагальнення	12
1.3 Оцінка обмежень та ризиків.....	12
2 Моделювання та проектування системи.....	15
2.1 Постановка задачі розробки системи	15
2.1.1 Характеристика користувача	15
2.1.2 Вимоги до інтерфейсу.....	16
2.1.3 Системні вимоги.....	16
2.1.4 Обмеження	18
2.2 Функціональні можливості користувача	39
2.3 Опис та обґрунтування обраних інструментів	20
2.3.1 Системи керування базами даних.....	21
2.3.2 Мова програмування.....	25
2.3.3 Технологія проектування користувацького інтерфейсу	27
2.4 Проектування баз даних	28
2.5 Діаграми послідовностей.....	33
2.6 Оцінка результативності роботи ментора.....	36
2.7 Формування рекомендацій для менті.....	39
3 Архітектура системи	40
3.1 Зовнішня архітектура.....	41

3.2 Внутрішня архітектура	42
3.3 Апаратне розгортання архітектури	47
4 Програмна розробка системи	50
4.1 Розробка рекомендаційної частини	50
4.2 Розробка серверної частини	58
4.3 Розробка клієнтської частини	62
5 Дослідження системи.....	66
Висновки	73
Перелік джерел посилання	74
Додаток А Шаблон інтерфейсу компонента профілю менті	76
Додаток Б Стили компонента менті	77
Додаток В Слайди презентації.....	79

ВСТУП

Сучасний розвиток інформаційних технологій значно розширив можливості для здобуття нових знань та професійних навичок. Онлайн-курси, освітні платформи, індивідуальні консультації з фахівцями стали звичними інструментами навчання. Проте, попри широкий спектр освітніх ресурсів, залишається відкритим питання ефективного підбору наставника (ментора), який би максимально відповідав потребам і рівню підготовки здобувача знань [1].

Задачею даного дослідження є розробка комплексної методики для формування рекомендацій з метою підбору наставників за необхідними навичками на основі результативності їх роботи з підтримкою контролю успіхів учня. Систему реалізовано з використанням методів статистичного аналізу для підбору рекомендаційного переліку наставників.

У межах роботи розроблено критерії оцінки успішності наставників та методику формування рекомендацій їх підбору за необхідними навичками. За результатами дослідження побудовано програмний комплекс, який включає формування та попередню обробку набору даних, навчання моделей прогнозування на основі бібліотек машинного навчання, а також реалізацію серверної логіки для формування рекомендацій. Особлива увага приділяється питанням структуризації даних, побудові моделі підбору менторів і забезпеченню можливості масштабування системи.

Результатом проведеного дослідження є створення прототипу інтерактивної системи, здатної формувати обґрунтовані рекомендації щодо вибору наставників за визначеними критеріями. Такий підхід сприятиме подальшому розвитку методів аналітичної обробки інформації та підвищенню якості прийняття рішень у сфері освітніх технологій і персоналізованого навчання.

1 КОНЦЕПТУАЛЬНИЙ АНАЛІЗ

Перед моделюванням та розробкою системи слід провести глибоку аналітичну роботу, спрямовану на всебічне розуміння контексту проєкту, його передумов та потенційних викликів. Цей розділ є критично важливим початковим етапом, мета якого — закласти міцний теоретичний та практичний фундамент для подальшої роботи.

1.1 Визначення та формування проблеми

Однією з ключових проблем сучасного процесу менторства є відсутність об'єктивного механізму зіставлення потреб учня (менті) та компетентностей ментора [2]. У більшості випадків вибір наставника здійснюється випадково, на основі рекомендацій знайомих, суб'єктивних відгуків або ж загальної популярності спеціаліста. Це призводить до низки негативних наслідків: ментор може не мати достатніх компетенцій у конкретному навичку, який прагне вдосконалити менті; відсутність системного аналізу результатів роботи менті ускладнює оцінку ефективності наставництва.

Ще однією проблемою є складність відстеження прогресу учня. Традиційна модель навчання з ментором не передбачає обов'язкового збору статистичних даних щодо того, як саме змінюється рівень навичок менті протягом співпраці. У результаті учень не завжди може оцінити користь від занять, а ментор не має достатніх інструментів для коригування навчального процесу [3].

Таким чином, можна виокремити такі основні проблеми: відсутність системи обґрунтованого підбору ментора за конкретними навичками, які бажає вдосконалити користувач; недостатність інструментів для оцінки компетентностей менторів, що знижує прозорість процесу вибору наставника;

відсутність системного збору статистики щодо прогресу менті, що унеможливлює повну оцінку результативності наставництва; складність аналізу взаємодії «ментор – менті» у динаміці, що ускладнює вдосконалення навчальних програм та підходів.

Вирішення цих проблем потребує створення інтерактивної інформаційної системи, яка дозволить: здійснювати підбір наставників на основі визначених критеріїв та навичок, що потребують розвитку; акумулювати статистику як про діяльність менторів, так і про прогрес менті; забезпечити об'єктивність процесу навчання завдяки можливості відстеження динаміки змін у рівні компетентностей учня.

1.2 Аналіз існуючих рішень та технологій

Проблема підбору менторів та оцінки ефективності процесу наставництва вже певною мірою розглядається в сучасних освітніх та професійних сервісах. Серед них можна виокремити як вузькоспеціалізовані платформи для менторства, так і широкоформатні навчальні ресурси, що надають послуги індивідуальних консультацій. Проте більшість із них вирішують лише окремі аспекти проблеми, не пропонуючи комплексного підходу [4].

1.2.1 Освітні онлайн-платформи з менторською підтримкою

Основні представники: Coursera, Udemy, Udacity.

Багато освітніх платформ пропонують доступ до курсів із можливістю отримання консультацій від викладачів чи експертів. Деякі з них мають функцію закріплення за слухачем наставника.

Переваги: великий вибір курсів і спеціалістів, наявність перевірених програм навчання, можливість комунікації зі світовими експертами.

Недоліки: вибір ментора здебільшого не базується на глибокому аналізі конкретних навичок менті; відсутній інструмент відстеження реального прогресу

учня за навичками, а оцінювання часто зводиться до проходження тестів або виконання завдань без довгострокового моніторингу.

1.2.2 Професійні спільноти та платформи менторства

Основні представники: MentorCruise, ADPList, GrowthMentor.

Ці сервіси спеціалізуються саме на пошуку наставників для професійного розвитку. Користувач може обрати ментора за напрямом діяльності та досвідом.

Переваги: зручний пошук фахівців за галузями, можливість довготривалого співробітництва, пряма комунікація між ментором та менті.

Недоліки: відсутність глибокого аналітичного інструментарію для оцінки прогресу учня; об'єктивність вибору ментора залишається низькою, оскільки рішення здебільшого приймається на основі описів профілю, рейтингу або відгуків.

1.2.3 Корпоративні системи навчання

Основні представники: LinkedIn Learning, внутрішні HR-платформи.

Деякі великі організації створюють власні системи менторства, інтегровані в кадрові процеси. Тут підбір наставників може враховувати професійні досягнення співробітників, результати атестацій та розвиток у кар'єрних траєкторіях.

Переваги: можливість аналізу компетентностей на основі внутрішніх даних компанії; більш точний підбір наставників відповідно до потреб.

Недоліки: закритість системи (використання лише всередині компанії), відсутність універсальності для широкого кола користувачів; обмеженість у використанні для незалежних менті, які не є співробітниками компанії.

1.2.4 Системи управління навчанням

Основні представники: Moodle, Blackboard, Canvas.

LMS забезпечують інструменти організації освітнього процесу, контролю виконання завдань і відстеження успішності студентів.

Переваги: розвинені можливості збору статистики щодо навчання; інтеграція з різними освітніми інструментами.

Недоліки: зосередженість на формальній освіті; недостатня гнучкість для моделі «ментор – менті»; не орієнтованість на підбір наставника за конкретними навичками.

1.2.5 Узагальнення

Наявні рішення частково охоплюють окреслені проблеми, проте: системний підбір ментора за конкретними навичками залишається нерозв'язаним: вибір здебільшого базується на описах профілю, рейтингах чи популярності; збір статистики щодо прогресу менті не реалізований у більшості рішень: існуючі сервіси здебільшого зосереджені на курсах і тестах, а не на вимірюванні змін навичок; інтеграції аналітики обох сторін взаємодії (ментор – менті) майже не існує: відсутні механізми, які одночасно враховують компетентності наставника та розвиток учня.

Таким чином, актуальною є розробка інтерактивної системи, яка б поєднала алгоритми підбору наставників за критеріями навичок із механізмами збору та аналізу статистики прогресу менті. Це дозволить забезпечити більш прозорий, об'єктивний та ефективний процес наставництва.

1.3 Оцінка обмежень та ризиків

Розробка інтерактивної системи для рекомендацій щодо обрання ментора за визначеними критеріями, попри її високу актуальність і значні перспективи, супроводжується рядом обмежень та потенційних ризиків. Ці аспекти необхідно уважно враховувати як під час проєктування, так і на етапі подальшої експлуатації системи, адже їх ігнорування може суттєво вплинути на якість рекомендацій і загальну ефективність платформи.

Одним із ключових обмежень є якість вхідних даних. Ефективність рекомендацій значною мірою залежить від достовірності та актуальності інформації про навички менторів і рівень підготовки менті. Якщо дані є

неповними, застарілими або містять помилки, система може формувати неточні або навіть хибні рекомендації. Це, у свою чергу, знижує довіру користувачів і ставить під загрозу коректність роботи всієї системи.

Важливу роль відіграють і технічні ресурси, необхідні для повноцінного функціонування платформи. Для якісного збору, аналізу й збереження статистики потрібна надійна інфраструктура, зокрема сервери належної потужності, оптимізовані бази даних і сучасні засоби захисту. За умови значного збільшення кількості користувачів це може створювати труднощі з масштабуванням, що вимагатиме додаткових витрат і технічної підтримки. Окремо слід враховувати питання конфіденційності та безпеки даних. Оскільки система оперує персональною інформацією менторів і менті, вона повинна відповідати вимогам законодавства щодо захисту даних, як-от GDPR або локальні регуляції. Невиконання цих вимог може призвести не лише до технічних ризиків, а й до юридичної відповідальності, штрафів і втрати репутації, що суттєво ускладнить подальшу експлуатацію проєкту.

Не менш вагомим є блок правових ризиків, які виходять за межі лише захисту даних. Законодавські вимоги можуть стосуватися умов використання платформи, відповідальності за контент, трудових відносин між ментором і менті, а також питань інтелектуальної власності [5]. Невірно оформлені угоди, відсутність прозорих політик або неналежне інформування користувачів про умови співпраці можуть призвести до судових спорів і фінансових зобов'язань. Тому юридичний супровід, чіткі користувацькі угоди та механізми відстеження згоди користувачів повинні бути інтегровані в архітектуру проєкту від самого початку.

Крім технічних і правових аспектів, важливо пам'ятати про алгоритмічні обмеження. Рекомендаційні алгоритми потребують великого обсягу релевантних даних для навчання й адаптації, і на початкових етапах, коли вибірка ще невелика, їхні прогнози можуть бути менш точними. Додатково існує ризик упередженості моделі: якщо дані тренування не є збалансованими, алгоритми можуть

дискримінувати окремі групи менторів або менті, що вимагатиме введення механізмів контролю якості та корекції упереджень.

Ризик формування некоректних рекомендацій варто розглядати окремо від загальних алгоритмічних обмежень, оскільки його наслідки стосуються безпосередньо довіри користувачів. У випадку недостатньої кількості або низької якості даних система може пропонувати менторів, які не відповідають очікуванням користувача, що знижує задоволеність і призводить до відтоку аудиторії. Щоб мінімізувати цей ризик, необхідно впроваджувати механізми валідації даних, зворотного зв'язку та ручного перегляду критичних результатів.

Ризик низької залученості користувачів також є серйозним фактором, що впливає на успішність проєкту. Якщо інтерфейс буде недостатньо інтуїтивним або рекомендації виглядатимуть формальними і не персоналізованими, менті та ментори можуть просто не скористатися платформою. У відповідь на це слід приділяти особливу увагу UX-дизайну, механікам мотивації користувачів та прозорості роботи алгоритмів, щоб користувачі розуміли, чому їм пропонують ті чи інші варіанти.

Ще одним важливим викликом є ризик конкуренції. На ринку вже присутні відомі сервіси, такі як MentorCruise, ADPList і GrowthMentor, які мають свою базу користувачів і відпрацьовані бізнес-моделі. Щоб зайняти свою нішу, нова система має запропонувати унікальні переваги — наприклад, більш точну персоналізацію, спеціалізовані галузеві рекомендації, кращу інтеграцію з інструментами навчання або вигідні умови для менторів. Без чіткої конкурентної переваги продукт ризикує залишитися малопопулярним, навіть за умови технічної досконалості.

Врахування всіх перерахованих обмежень і ризиків дозволяє сформулювати основні завдання для реалізації проєкту: забезпечити високу якість і актуальність даних, побудувати надійну та масштабовану інфраструктуру, розробити прозорі й точні алгоритми рекомендацій, інтегрувати юридичні механізми для захисту прав користувачів і створити конкурентні переваги на ринку. Подолання цих викликів стане визначальним фактором для успішної реалізації та довгострокового функціонування системи.

2 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Постановка задачі розробки системи

Розроблювальна система є рекомендаційною онлайн-платформою, яка підтримує функціонал, як для внутрішніх сторін (адміністраторів, модераторів) так і для зовнішніх осіб (користувачів).

Система має бути загальною доступною, без надання дозволу на користування та повинна однаково працювати на таких популярних браузерах як: Chrome, Safari, Firefox, Opera. Час відповіді на запити користувачів не повинно перевищувати 5 секунд. Для користування застосунком, користувач повинен мати персональний комп'ютер або смартфон з доступом до мережі Інтернет, обладнаний веб-браузером.

Система надасть змогу полегшити процес підбору наставників відповідно до потреб користувачів, підвищити ефективність навчання та зробити процес наставництва більш прозорим. Для цього під час розробки застосунку слід вирішити такі задачі: надати можливість користувачеві обирати навичку, яку він прагне вдосконалити; забезпечити механізм рекомендацій менторів на основі їхньої статистики; реалізувати збір та аналіз даних про прогрес менті для відстеження динаміки розвитку навички у часі; забезпечити зручний та інтуїтивний інтерфейс для пошуку, перегляду профілів менторів та відстеження власних результатів.

2.1.1 Характеристика користувача

Типовий користувач системи рекомендацій менторів – це людина, яка прагне вдосконалити певні професійні чи особисті навички за допомогою наставництва. Вона має доступ до Інтернету та користується веб-браузером або мобільним застосунком.

Передбачається, що користувач достатньо знайомий зі смартфоном чи комп'ютером, вміє працювати з браузером, клавіатурою та мишею, а також здатен взаємодіяти з простими веб-інтерфейсами.

Для врахування потреб і функцій різних груп користувачів системи слід визначити їхні ролі (табл. 2.1).

Таблиця 2.1 – Ролі користувачів онлайн платформи

Роль	Опис ролі
Гість	Користувач, який ще не зареєстрований або не пройшов автентифікацію. Має можливість переглядати загальну інформацію про платформу та ознайомлюватися з принципами її роботи.
Менті	Зареєстрований та авторизований користувач, який прагне покращити певну навичку. Має можливість обирати навичку для розвитку, переглядати список рекомендованих менторів та відстежувати власний прогрес.
Ментор	Зареєстрований та авторизований користувач, який надає послуги наставництва. Може створювати та редагувати власний профіль, відслідковувати результати своїх менті та отримувати оцінки своєї роботи.
Модератор	Користувач із розширеними повноваженнями, який може редагувати або видаляти некоректний контент, а також слідкувати за дотриманням правил платформи.
Адміністратор	Користувач із повним доступом до системи, який керує обліковими записами, контролює статистику, має можливість змінювати налаштування платформи та забезпечує її стабільне функціонування.

2.1.2 Вимоги до інтерфейсу

Для забезпечення максимального комфорту користувачів при роботі з інтерактивною системою рекомендацій, інтерфейс повинен відповідати таким ключовим вимогам:

- бути адаптивним для різних пристроїв (смартфонів, планшетів, ноутбуків, стаціонарних ПК) та коректно відображатися на різних розмірах екрану;
- бути сумісним з усіма популярними веб-браузерами (Chrome, Safari, Firefox, Opera, Edge);
- забезпечувати інтуїтивність та простоту у користуванні, уникати перевантаження зайвими елементами, концентруючи увагу на основних функціях платформи;
- надавати зручні інтерактивні форми для введення критеріїв пошуку та відбору менторів, а також для надсилання запитів і повідомлень;
- мати розширений та гнучкий інструмент пошуку з можливістю фільтрації за критеріями (досвід ментора, спеціалізація, рівень знань, відгуки, доступність тощо);
- забезпечувати наочне відображення результатів збору статистики та рекомендацій у вигляді списків, карток або рейтингових таблиць, графіків, зрозумілих для користувача;
- надавати інструментарій для модераторів та адміністраторів для управління обліковими записами, перевірки коректності даних і контролю за дотриманням правил спільноти;
- забезпечувати можливість швидкої навігації по профілях менторів із переглядом детальної інформації та контактів.

Кастомізація інтерфейсу кінцевими користувачами не передбачається.

2.1.3 Системні вимоги

Рекомендовані вимоги клієнтської частини:

- веб-браузер: остання версія Chrome, Firefox, Safari або Edge;
- процесор: чотириядерний від 2,5 ГГц;
- оперативна пам'ять: від 4 ГБ;
- роздільна здатність екрану: Full HD (1920×1080) і вище;
- стабільний доступ до Інтернету від 10 Мбіт/с.

Рекомендовані вимоги серверної частини:

- операційна система: Linux Ubuntu 22.04 LTS / Debian 11;

- процесор: 8 ядер від 2,8 ГГц;
- оперативна пам'ять: від 16 ГБ;
- дисковий простір: від 250 ГБ SSD NVMe;
- використання CDN для швидкої доставки статичного контенту;
- підтримка резервного копіювання та відновлення даних.

2.1.4 Обмеження

Користувачі системи можуть мати тільки одну роль.

Сторінка з інструментарієм адміністраторів повинна бути приватною, тобто будь-який користувач який не має ролі «Адміністратор» не повинен мати можливість потрапити на дану сторінку. Користувачі сайту з роллю «Гість» має право перебувати на всіх публічних або не персоналізованих сторінках (наприклад, сторінка з графіком з зібраною статистикою), але не має можливості знайти ментора, обрати ментора, переглянути свою статистику.

2.2 Функціональні можливості користувача

Виходячи з постановки задачі можна зробити висновок, що в системі наявні наступні ролі, які мають відповідний набір функціональних можливостей та повноважень: «Гість», «Менті», «Ментор», «Модератор», «Адміністратор».

«Гість» матиме найменший набір функціоналу, наведений на рисунку 2.1.

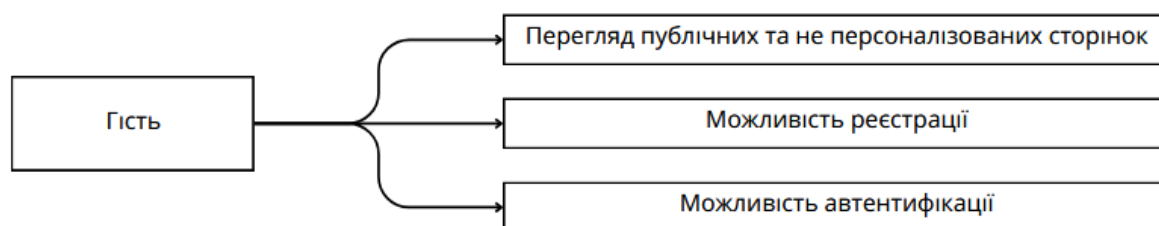


Рисунок 2.1 - Функціонал користувача з роллю «Гість»

Користувач з роллю «Менті» в свою чергу вже матиме увесь необхідний функціонал для повноцінного користування сайтом, що надасть змогу знайти необхідного ментора та розпочати з ним співпрацю.

Функціонал користувача з роллю «Менті» наведений на рисунку 2.2.



Рисунок 2.2 – Функціонал користувача з роллю «Менті»

Користувач з роллю «Ментор» матиме увесь функціонал необхідний функціонал для повноцінного користування сайтом, що надасть змогу полегшити співпрацю з менті.

Функціонал користувача з роллю «Ментор» наведений на рисунку 2.3.



Рисунок 2.3 – Функціонал користувача з роллю «Ментор»

Роль «Модератор» надасть список повноваження, наведені на рисунку 2.4.

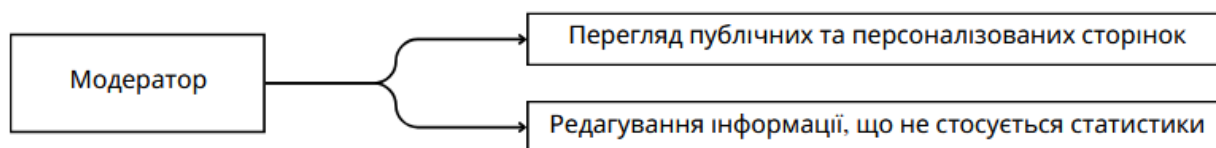


Рисунок 2.4 – Функціонал користувача з роллю «Модератор»

Роль «Адміністратор» вміщує в собі найбільший функціонал, в який в ходять повноваження ролі «Модератор», а також власні, які нададуть повний контроль над вмістом сайту.

Повноваження та функціонал ролі «Адміністратор» на рисунку 2.5.

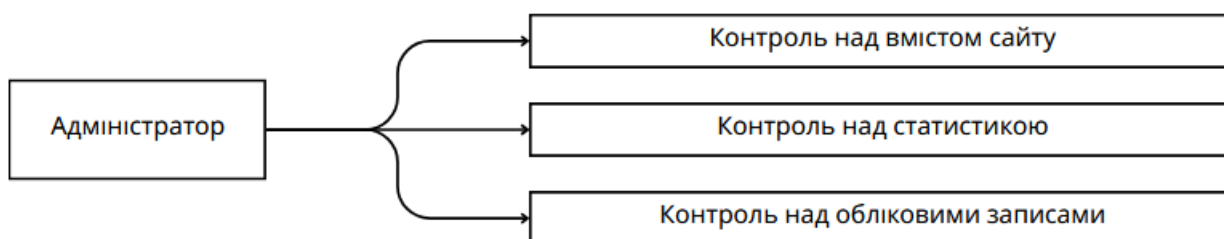


Рисунок 2.5 – Повноваження та функціонал ролі «Адміністратор»

2.3 Опис та обґрунтування обраних інструментів

Розробка системи потребує використання сучасних інструментів та технологій, які забезпечать масштабованість та зручність у подальшій експлуатації. Вибір кожного інструменту ґрунтується на його відповідності вимогам до продуктивності, надійності, простоти інтеграції та підтримки.

У даному підрозділі буде наведено перелік обраних технологій та засобів розробки, а також обґрунтовано доцільність їх використання для реалізації проєкту.

2.3.1 Системи керування базами даних

PostgreSQL — це потужна об'єктно-реляційна система управління базами даних з відкритим вихідним кодом, яка підтримує SQL-стандарт і розширені можливості для роботи з даними. Вона є однією з найпопулярніших СУБД у світі завдяки своїй надійності, продуктивності та активній спільноті розробників [6].

PostgreSQL була створена як проєкт на базі університету Берклі (США) ще у 1986 році, і з того часу розвивається як незалежна від комерційних компаній технологія з відкритим кодом. Система має гнучку архітектуру та забезпечує високий рівень відповідності стандартам SQL, підтримує складні типи даних, зберігає функції та тригери, а також дозволяє розширювати функціонал через власні модулі [7].

Переваги: підтримка складних запитів та транзакційного контролю (ACID); розширюваність: можливість створювати власні типи даних, функції, оператори; сумісність із сучасними мовами програмування та фреймворками; активна спільнота та регулярні оновлення; висока надійність і відмовостійкість, перевірена роками у виробничих середовищах.

Недоліки: потребує більших ресурсів у порівнянні з деякими «легкими» СУБД; складніша в адмініструванні, ніж, наприклад, MySQL; у деяких випадках поступається за швидкістю спеціалізованим базам даних, оптимізованим під читання.

MySQL вважається більш легкою у налаштуванні та використанні, особливо для невеликих проєктів, а також має значну популярність серед веб-додатків. Водночас PostgreSQL забезпечує кращу відповідність стандартам SQL, має розширені можливості роботи зі складними структурами даних, а також вищу стабільність при великих навантаженнях. У сфері наукових, освітніх та корпоративних систем PostgreSQL часто визнається більш перспективним рішенням [8].

Попри те, що PostgreSQL може бути складнішою у адмініструванні, її переваги, такі як підтримка складних транзакцій, масштабованість та відповідність вимогам до інтерактивної системи з великим обсягом даних,

роблять її оптимальним вибором для реалізації даного проєкту. PostgreSQL дозволяє не лише зберігати статистику менторів та користувачів, але й здійснювати гнучкі аналітичні запити, що є ключовим для системи рекомендацій.

Разом із основною реляційною базою даних, проєкт вимагає додаткових інструментів для роботи з даними, які мають інші характеристики. PostgreSQL забезпечує надійне збереження та обробку структурованої інформації, однак для задач кешування, підвищення швидкодії при запитах та тимчасового зберігання даних доцільним є використання нереляційної бази даних Redis.

Redis (Remote Dictionary Server) — це високопродуктивна нереляційна система управління базами даних, яка працює в оперативній пам'яті та реалізує модель ключ-значення. Redis використовується як кеш, брокер повідомлень або тимчасове сховище даних, що забезпечує надзвичайно швидкий доступ до інформації [9].

Redis було створено у 2009 році як рішення для роботи з даними в реальному часі. На відміну від реляційних СУБД, Redis зберігає всю інформацію в оперативній пам'яті з можливістю періодичної синхронізації з диском, що дозволяє досягати дуже високої швидкодії. Система підтримує різні структури даних, такі як рядки, множини, хеші, списки, відсортовані множини, а також має механізми TTL (часу життя ключа), що робить її особливо зручною для кешування [10].

Переваги: надзвичайно висока швидкодія завдяки роботі з оперативною пам'яттю; підтримка різноманітних структур даних; простота інтеграції з більшістю сучасних мов програмування; можливість використання як кеша, черги завдань або брокера повідомлень; підтримка реплікації та кластеризації для масштабування.

Недоліки: зберігання даних в оперативній пам'яті потребує великих ресурсів сервера; не оптимізований для складних аналітичних запитів чи обробки великих обсягів історичних даних; відсутність жорстких механізмів транзакційного контролю на рівні, що притаманний реляційним СУБД.

Memcached також є високошвидкісним рішенням для кешування, однак на відміну від Redis підтримує лише просту модель ключ-значення без розширених структур даних. Redis надає розробникам значно більшу гнучкість завдяки підтримці множин, черг, лічильників, відсортованих колекцій тощо [11]. Крім того, Redis має вбудовану підтримку стійкості даних (запис на диск), що відсутнє в Memcached.

Попри обмеження, пов'язані з використанням оперативної пам'яті, Redis є оптимальним вибором для даної системи. Він забезпечує миттєвий доступ до даних, що критично важливо для роботи алгоритму рекомендацій у режимі реального часу. Наприклад, кешування часто використовуваних запитів, збереження активних сесій користувачів або миттєвий підрахунок статистики робить Redis незамінним інструментом. Його швидкодія та підтримка різних структур даних дозволяють розвантажити інші компоненти системи.

Redis вирішує завдання високошвидкісного доступу до тимчасових даних і кешування, однак у межах системи також існує потреба у зберіганні напівструктурованої інформації, зокрема аналітичних даних. Для таких випадків доцільно використовувати документно-орієнтовану базу даних MongoDB, яка краще підходить для роботи з динамічними структурами та не потребує жорсткої схеми зберігання.

MongoDB — це документно-орієнтована система управління базами даних із відкритим вихідним кодом, яка зберігає дані у форматі BSON (бінарний JSON). Вона належить до класу NoSQL-баз та забезпечує гнучку роботу з напівструктурованими даними [12].

MongoDB була створена у 2009 році компанією 10gen (нині MongoDB Inc.) як рішення для сучасних веб-додатків, яким притаманний динамічний та швидкозмінний характер даних. На відміну від реляційних СУБД, MongoDB не вимагає суворої схеми таблиць, що робить її зручною для зберігання інформації з непостійною структурою: відгуки, коментарі, логи, історія взаємодії користувачів із системою тощо [13].

Переваги: відсутність жорсткої схеми зберігання, що дозволяє легко адаптувати структуру під змінні вимоги; зручність роботи з JSON-подібними документами, що добре інтегрується з сучасними веб-технологіями; масштабованість за рахунок шардінгу та реплікації; висока продуктивність при роботі з великими масивами напівструктурованих даних; активна спільнота та комерційна підтримка.

Недоліки: менша надійність транзакцій у порівнянні з реляційними СУБД (хоча з версії 4.0 підтримуються багатодокументні транзакції, вони поступаються PostgreSQL за стабільністю); значні витрати пам'яті через зберігання документів у BSON-форматі; ефективна робота з великими реляційними зв'язками, які характерні для традиційних СУБД.

Cassandra, як і MongoDB, є NoSQL-базою даних, проте орієнтується на модель «колонкових сімейств», що робить її кращою для аналітики великих обсягів даних у розподілених системах. У свою чергу, MongoDB забезпечує більш просту модель роботи з документами та краще підходить для веб-додатків, де дані мають гнучку та непостійно змінювану структуру. Таким чином, Cassandra виграє у масштабуванні для великих дата-центрів, але MongoDB надає більше зручності та швидкості при інтеграції у проекти, орієнтовані на користувачів [14].

Попри недоліки, MongoDB є оптимальним вибором для даного проекту, оскільки дозволяє зберігати напівструктуровані дані, пов'язані з відгуками користувачів, історією взаємодії та додатковою аналітикою. Її гнучкість дає змогу без складних змін у структурі бази розширювати функціонал системи. На відміну від PostgreSQL, яка потребує суворої схеми, MongoDB дає змогу швидко адаптуватися до нових вимог без суттєвої перебудови моделі даних.

Таким чином, у межах проекту було обрано три бази даних, кожна з яких виконує власну роль:

- PostgreSQL: для збереження основних структурованих даних та виконання складних транзакцій;
- Redis: для кешування, швидкодії та роботи з тимчасовими даними в реальному часі;

- MongoDB: для збереження напівструктурованої інформації, зокрема відгуків, історії взаємодій та додаткової аналітики.

Таке поєднання дозволяє забезпечити баланс між надійністю, продуктивністю та гнучкістю, що робить систему оптимальною для реалізації поставлених цілей.

2.3.2 Мова програмування

Java — це об'єктно-орієнтована мова програмування загального призначення, створена компанією Sun Microsystems у 1995 році (нині розвивається компанією Oracle). Її ключовим принципом є девіз «Write Once, Run Anywhere», що означає можливість виконання програмного коду на будь-якій платформі, де встановлена Java Virtual Machine (JVM) [15].

Java використовується для створення корпоративних застосунків, мобільних додатків (Android), системного та серверного програмного забезпечення. Мова поєднує високий рівень абстракції з великою екосистемою бібліотек та фреймворків, що робить її однією з найбільш поширених технологій у світі [16].

Переваги: кросплатформність завдяки JVM; висока продуктивність у поєднанні з автоматичним управлінням пам'яттю (Garbage Collector); розвинена екосистема (Spring, Hibernate, Kafka, тощо); велика спільнота та довготривала підтримка; наявність широкого спектру інструментів для інтеграції з базами даних, мікросервісами та хмарними рішеннями.

Недоліки: вища «вербозність» коду у порівнянні з сучасними мовами (наприклад, Kotlin чи Python); повільніший старт додатків порівняно з C# або Go; іноді потребує більше ресурсів через роботу JVM.

C# є головним конкурентом Java, оскільки має подібну об'єктно-орієнтовану парадигму та синтаксис. Обидві мови забезпечують високу продуктивність та підтримку корпоративних рішень. Основною перевагою C# є глибока інтеграція з екосистемою Microsoft (.NET Framework, Azure), тоді як Java забезпечує більшу кросплатформність і має ширший вибір серверних фреймворків. У результаті Java часто обирають для мультиплатформних систем із можливістю розгортання у різних середовищах [17].

Spring — один із найбільш поширених фреймворків з відкритим вихідним кодом, призначений для створення веб-застосунків на основі мови Java [18]. Його ключовими характеристиками є універсальність та розширюваність, що забезпечує широкі можливості для побудови програмних рішень різної складності. Серед основних переваг варто відзначити гнучкі механізми конфігурації, інтегровану підтримку транзакцій, розвинені засоби безпеки (зокрема завдяки модулю Spring Security), а також ефективний контейнер для управління об'єктами — бінами [18]. Водночас використання додаткового шару абстракції може призводити до зниження продуктивності та ускладнювати процес налаштування системи. Однак поява інструментів на кшталт Spring Boot значно оптимізувала конфігурацію та розгортання застосунків, забезпечивши швидший і зручніший старт нових проєктів.

У межах проєкту важливим завданням є опрацювання даних та побудова рекомендацій на основі статистики користувачів. Для реалізації цього функціоналу необхідно використати інструменти, що забезпечують ефективні методи машинного навчання та аналізу даних. Саме на цьому етапі доцільним є використання спеціалізованих бібліотек.

Weka (Waikato Environment for Knowledge Analysis) — це програмне середовище та бібліотека з відкритим вихідним кодом, розроблена в Університеті Вайкато (Нова Зеландія), яка надає набір алгоритмів і засобів для виконання завдань у сфері машинного навчання та інтелектуального аналізу даних. Вона підтримує широкий спектр алгоритмів класифікації, регресії, кластеризації, а також має інструменти для передобробки та візуалізації даних [19].

Вибір саме Weka зумовлений кількома причинами. По-перше, бібліотека є нативною для Java, що забезпечує її повну сумісність із вже обраною мовою програмування та фреймворком Spring. По-друге, Weka має зручний програмний інтерфейс, який дозволяє інтегрувати алгоритми машинного навчання без необхідності реалізації їх «з нуля». По-третє, бібліотека відзначається широким спектром готових до використання алгоритмів, що скорочує час розробки та підвищує гнучкість системи.

Попри певні недоліки, Java була обрана як основна мова програмування завдяки своїй кросплатформності, стабільності, широкій екосистемі та підтримці великих корпоративних рішень. У поєднанні зі Spring вона забезпечує ефективну розробку мікросервісної архітектури, а використання бібліотеки Weka дозволяє інтегрувати інструменти машинного навчання у систему без зайвих витрат часу.

2.3.3 Технологія для проєктування користувацького інтерфейсу

Angular — це платформа та фреймворк для розробки веб-застосунків з відкритим вихідним кодом, створена компанією Google. Angular дозволяє створювати односторінкові застосунки (SPA) з використанням компонентного підходу та інструментів для управління даними та маршрутизації [20].

Angular був створений як повністю оновлений фреймворк на базі попередньої версії AngularJS, і з моменту випуску Angular 2+ став більш сучасним та модульним рішенням. Він підтримує TypeScript, що забезпечує статичну типізацію та полегшує масштабування великих проєктів.

Angular пропонує комплексний набір інструментів, включаючи двостороннє зв'язування даних, реактивні форми, маршрутизацію, залежності та CLI для автоматизації розробки [21].

Переваги: компонентна архітектура, що спрощує підтримку та повторне використання коду; інтегрована система двостороннього зв'язування даних (two-way data binding); вбудована підтримка модульності, маршрутизації та сервісів; активна підтримка Google та велика спільнота; статична типізація через TypeScript, що підвищує надійність коду.

Недоліки: більш крута крива навчання для новачків у порівнянні з React; великий розмір бібліотек та складність для малих проєктів; складніша інтеграція з іншими бібліотеками порівняно з React.

React — це бібліотека для створення інтерфейсів користувача, розроблена Facebook. Основна відмінність полягає в тому, що React надає лише візуальний шар (UI), а для маршрутизації, управління станом та інших функцій необхідно використовувати додаткові бібліотеки. Angular є більш комплексним

фреймворком «із коробки», тоді як React дає більшу гнучкість у виборі інструментів, але потребує додаткової конфігурації [22].

Попри наявні недоліки, Angular був обраний як основний фронтенд-фреймворк завдяки своїй комплексності, можливості швидко масштабувати застосунок, інтеграції з TypeScript та наявності готових інструментів для побудови SPA. Використання Angular дозволяє реалізувати всі необхідні функції інтерактивної системи для рекомендацій менторів з мінімальними додатковими залежностями.

2.4 Проєктування баз даних

Проєктування бази даних є одним із визначальних етапів створення будь-якої інформаційної системи. Саме на цьому етапі формується структура, організація та взаємозв'язки даних, які надалі забезпечуватимуть функціонування системи. Проведення якісного проєктування ще до початку розробки є необхідною умовою, адже правильно побудована модель бази даних гарантує високу продуктивність, ефективний доступ до інформації, а також надійність і безпеку збережених даних.

У процесі розроблення концептуальної моделі бази даних для сервісів, що використовують PostgreSQL, першочерговим завданням є визначення сутностей. Кожна сутність описується назвою, набором атрибутів та первинним ключем. Далі формуються зв'язки між таблицями, а також визначаються обмеження цілісності даних, що дозволяє підтримувати коректність і узгодженість інформації.

Спираючись на попередній аналіз, для сервісу навиків, можна виділити наступні сутності: «Навики» (табл. 2.1), «Метрики_Навиків» (табл. 2.2).

Таблиця 2.1 – Сутність «Навики»

Атрибут	Тип даних	Примітка
skill_id	BIGINT	Первинний ключ
name	VARCHAR(30)	

Таблиця 2.2 – Сутність «Метрики_Навиків»

Атрибут	Тип даних	Примітка
skill_metric_id	BIGINT	Складний ключ
skill_id	BIGINT	Складний ключ
name	VARCHAR(50)	
weight	DOUBLE	

Описавши сутності бази стає можливим створення моделі «сутність-зв'язок» (ER-модель) – модель даних, яка вказує на те, як різні сутності зв'язані один між одним в системі. На рисунку 2.6 зображено створену ER-модель.

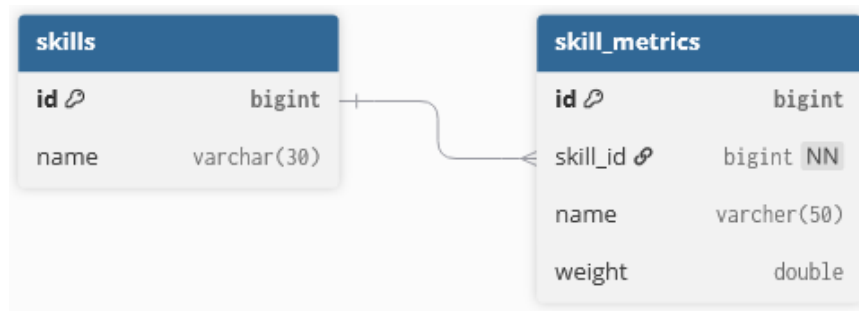


Рисунок 2.6 – ER-модель бази даних сервісу навиків

Для сервісу авторизації, можна виділити сутність, наведену в табл. 2.3.

Таблиця 2.3 – Сутність «Облікові_Дані»

Атрибут	Тип даних	Примітка
user_id	BIGINT	Первинний ключ
email	VARCHAR(255)	
password	VARCHAR(255)	
role	ENUM('USER', 'MODERATOR', 'ADMINISTRATOR')	

На поточному етапі розробки сутність «Облікові_Дані» не передбачає наявності зв'язків з іншими об'єктами, проте в майбутньому інтеграція таких зв'язків може бути реалізована без істотних ускладнень.

Для сервісів, які базуються на MongoDB, замість традиційних сутностей і таблиць визначаються документи. Кожен документ відображає логічно цілісну одиницю інформації, яка має власну структуру атрибутів. На цьому етапі особливу увагу приділяють побудові індексів, що забезпечують оптимізацію пошуку та підвищують продуктивність роботи системи за умови великих обсягів даних. Так для сервісу з обліком менторів та ментів, можна виділити наступні документи: «Менті» (табл. 2.4), «Ментор» (табл. 2.5).

Таблиця 2.5 – Опис документу «Менті»

Атрибут	Додаткова структура			Тип даних
userId				String
firstName				String
lastName				String
email				String
currentMentors				Array
	userId			String
	firstName			String
	lastName			String
	desiredSkill			String
stats				Array
	year			Int32
	months			Array
		month		String
		skills		Array
			skill	String
			score	Double

Таблица 2.6 – Опис документу «Ментор»

Атрибут	Додаткова структура		Тип даних
Id			String
firstName			String
lastName			String
email			String
since			Date
totalMentees			Int32
costPerMonth			Double
mentoringLocation			String
mentoring			Array
	start		Date
	end		Date
	desiredSkill		String
	durationInMonths		Int32
	mentee		Document
		userId	String
		firstName	String
		lastName	String
	efficiencyPercent		Double
	startSkillScore		Double
	endSkillScore		Double
	improvementPoints		Double
	improvementPercent		Double

Для сервісу відповідальний за змагання, спираючись на попередній аналіз, також було виділено документ під назвою «Результати_Змагань» (табл. 2.7).

Таблиця 2.7 – Опис документу «Результати_Змагань»

Атрибут	Додаткова структура		Тип даних
Id			String
name			String
date			Date
summery			Array
	userId		String
	firstName		String
	lastName		String
	metrics		Array
		metricId	String
		metricName	String
		result	Double

Такий підхід надасть можливість візуалізації статистики за допомогою графіків, а також спростить формування дата сету для навчання моделі при подальшому застосуванні.

Сервіси, що застосовують Redis, використовують іншу парадигму збереження інформації — модель ключ–значення, що використовуватиметься для сервісу аналізу даних, який що забезпечуватиме користувачів рекомендаціями щодо оптимальних менторів відповідно до обраного навичу. У такій структурі кожен елемент даних ідентифікується унікальним ключем, а доступ до нього здійснюється безпосередньо, що дозволяє досягти максимальної швидкості виконання операцій читання та запису.

У нашій моделі ключем виступатиме назва навичу, а значенням — сформований список відповідних менторів. Застосування цього підходу дає змогу суттєво підвищити швидкість обслуговування клієнтських запитів.

2.5 Діаграми послідовностей

Діаграми послідовностей належать до класу діаграм взаємодії та призначені для детального відображення процесу виконання операцій у системі. Вони моделюють обмін повідомленнями між об'єктами в межах певного сценарію співпраці, забезпечуючи чітке уявлення про послідовність їхніх дій.

У контексті розроблюваної системи основними об'єктами взаємодії виступають: клієнт, API Gateway, вбудований веб-сервер Tomcat (що використовується за замовчуванням у Spring Boot), Spring Boot мікросервіс та сховище даних (MongoDB, Redis або PostgreSQL залежно від сервісу).

Діаграми послідовностей мають часову орієнтацію: вони використовують вертикальну вісь як репрезентацію часу, відображаючи порядок обміну повідомленнями, а також момент їх відправлення та отримання. Це дозволяє наочно продемонструвати логіку виконання сценарію й узгодженість взаємодії між компонентами системи.

На рисунку 2.7 зображено діаграму послідовностей збору статистики.

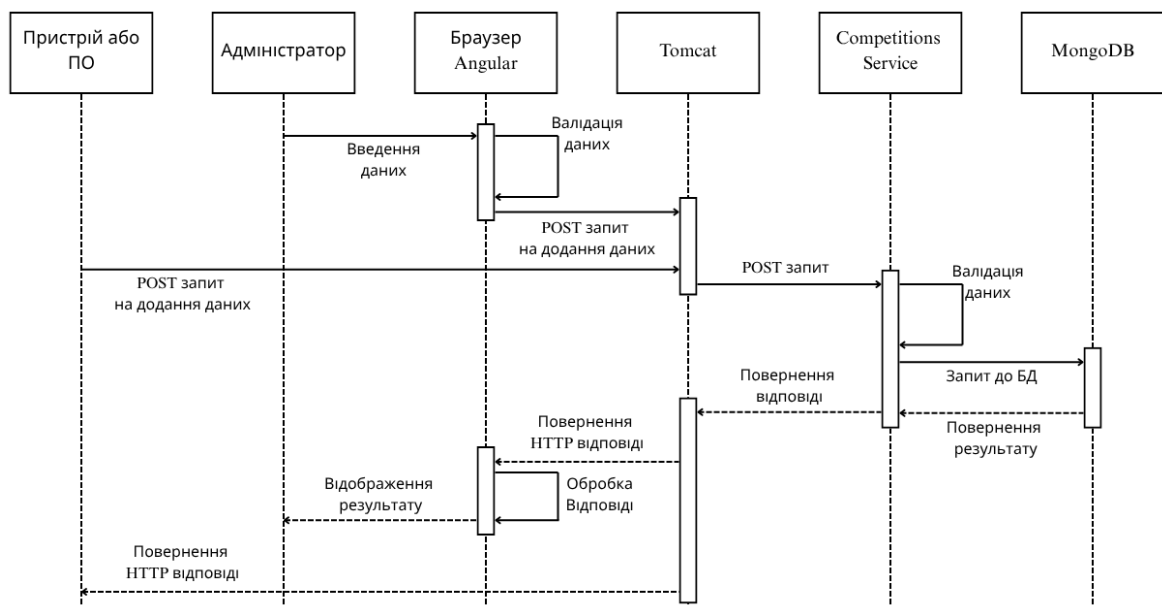


Рисунок 2.7 – Діаграма послідовностей збору статистики

Для забезпечення збору статистичних даних необхідна участь адміністратора, який здійснюватиме введення даних у процесі змагань, або використання технічного засобу чи програмного забезпечення, здатного формувати REST-запити та обробляти HTTP-відповіді. Після успішного проходження процедури валідації та збереження даних клієнтська сторона отримує відповідь із відповідним статусом успішного виконання операції.

Для обчислення актуального значення навичок менті Mentoring Service на початку кожного місяця ініціює запит на отримання статистичних даних за попередній період. Після опрацювання отриманих відомостей сервіс зберігає розраховані показники у базі даних.

На рисунку 2.8 зображено діаграму послідовностей для обробки статистики.

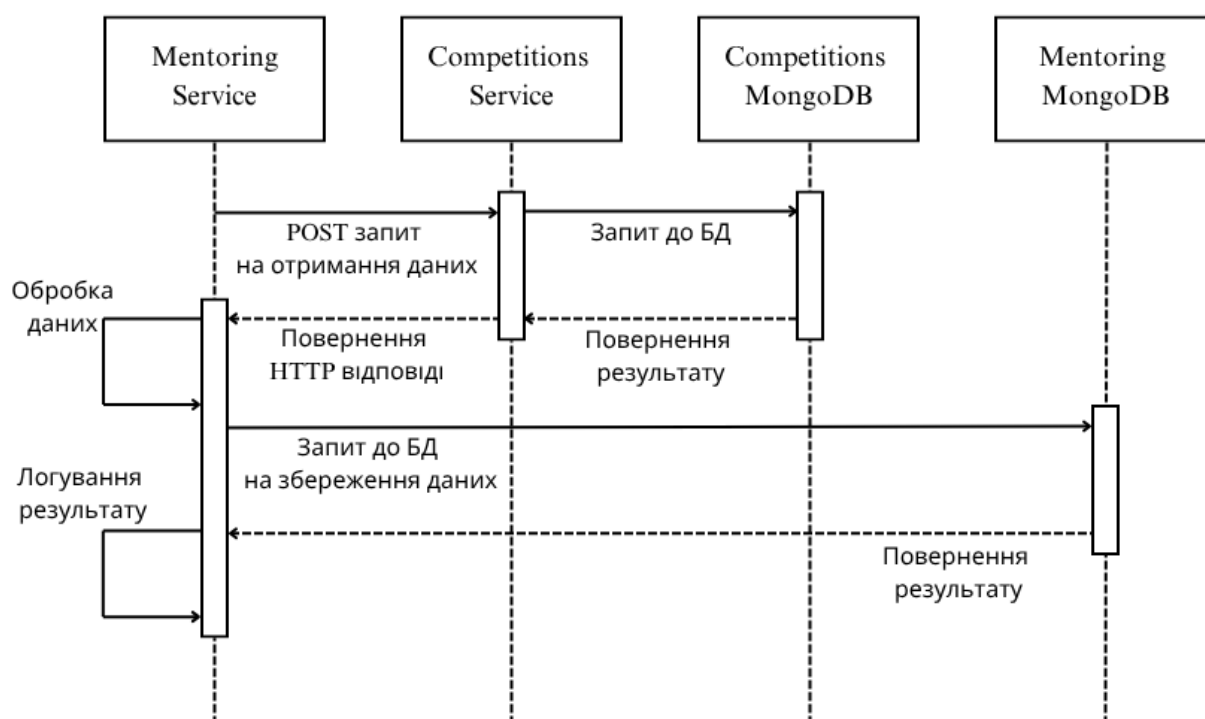


Рисунок 2.8 – Діаграма послідовностей обробки статистики

Для спрощення схеми API Gateway та Tomcat не було додано.

Подальше використання цих результатів можливе в рамках навчальної моделі, що забезпечує їх аналітичну та прогностичну цінність.

На рисунку 2.9 зображено як пов'язані рекомендаційна, серверна та клієнтська частина системи.

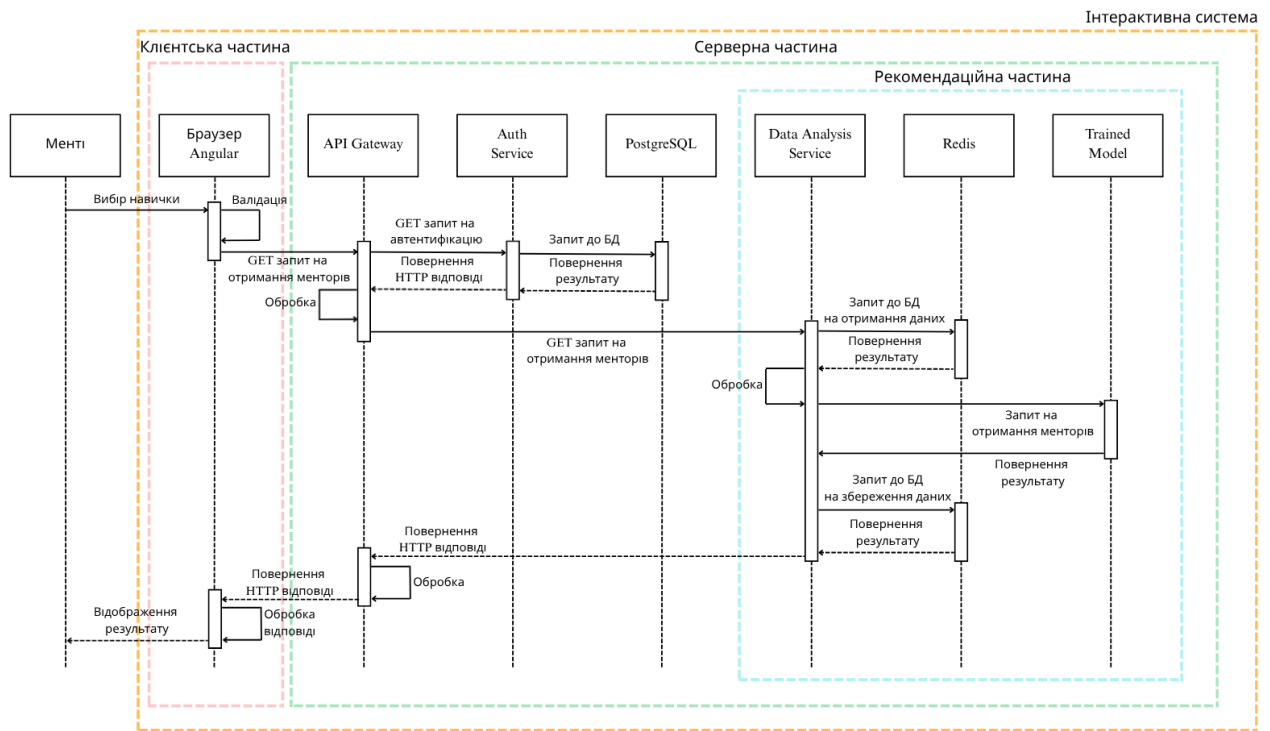


Рисунок 2.9 – Діаграма послідовностей отримання рекомендацій

Для спрощення схеми Tomcat не було додано.

Для отримання рекомендацій щодо вибору менторів менті повинен обрати зі списку навичку, яку він бажає вдосконалити. Після цього формується запит, який надсилається до API Gateway. Запит перенаправляється до Auth Service для перевірки автентичності та авторизації користувача. У випадку успішної валідації запит передається до Data Analysis Service, який звертається до Redis, щоб отримати вже збережені рекомендації, сформовані раніше.

Якщо у кеші відсутні дані для обраного навичку, сервіс надсилає запит до навченої моделі, яка формує прогноз у вигляді списку менторів. Отримані результати зберігаються у Redis із прив'язкою до конкретного навичку та повертаються для подальшої обробки.

2.6 Оцінка результативності роботи ментора

Для оцінювання результативності роботи ментора виконується аналіз успішності досягнень менті. Для цього в системі використовується регулярний збір даних на основі участі в змаганнях та інших заходах, де відбувається пряма конкуренція між учасниками. Оскільки кількість таких подій протягом місяця може бути значною, система агрегує їхні результати, трансформуючи показники у структуровані метрики.

Теоретичною основою такого підходу є концепції освітньої аналітики та кількісної оцінки ефективності навчання, які спираються на методи статистичного аналізу та математичного моделювання результатів навчальної діяльності. Використання математичного апарату аналізу даних для інтерпретації динаміки досягнень дозволяє об'єктивно порівнювати прогрес менті та виявляти вплив менторства на кінцеві результати [23].

Кожна навичка формується з набору метрик, які відображають специфічні аспекти його прояву.

Наприклад, для навички «Точність» ключовою метрикою стане «Кількість влучень», що безпосередньо фіксується під час змагань. Такий підхід забезпечує репрезентативність, оскільки оцінювання ґрунтується на реальних змагальних результатах.

Кожна навичка оцінюється в діапазоні від 0 до 100 балів. Вибір саме такої шкали обумовлений кількома чинниками: діапазон 0-100 інтуїтивно асоціюється з відсотковою шкалою, що спрощує розуміння результатів як для менті, так і для менторів; 100-бальна шкала дозволяє легко співставляти результати між різними навичками та учасниками, не потребуючи додаткового перетворення; завдяки цьому діапазону всі метрики приводяться до єдиного масштабу незалежно від їхніх одиниць вимірювання (наприклад, кількість влучень, кількість помилок, час реакції тощо).

Кожна навичка складається з кількох метрик, що мають власну вагу. Сума ваг усіх метрик однієї навички не перевищує 0,10.

Обґрунтування такого підходу: використання діапазону від 0,0 до 0,10 дозволяє уникнути домінування окремих метрик у межах однієї навички; при масштабуванні ваг у межах 0,10 зберігається пропорційність, і при підсумовуванні результатів вихідне значення залишається у діапазоні 0-100; оскільки $0,10 = 10\%$, це забезпечує логічне співвідношення між внеском метрики та її впливом на кінцеву оцінку навички.

Таким чином, дані змагань (кількість влучень, кількість промахів, швидкість реакції тощо) безпосередньо трансформуються в метрики з вагами, які у свою чергу формують кінцевий бал навички.

Для обчислення підсумкового результату виконується нормалізація значень відносно «типових» низьких та високих результатів.

Якщо більше значення метрики є кращим то підсумковий результат визначається за формулою 2.1.

$$MetricScore = 100 * \frac{EntrantValue - L}{U - L} \quad (2.1)$$

Якщо менше значення метрики є кращим то підсумковий результат визначається за формулою 2.2.

$$MetricScore = 100 * \frac{U - EntrantValue}{U - L} \quad (2.2)$$

де: *EntrantValue* – значення метрики конкретного учасника;

L – нижня межа (5-й перцентиль, що відсікає найгірші випадки);

U – верхня межа (95-й перцентиль, що відсікає найкращі випадки);

100 – коефіцієнт шкали для приведення результатів у діапазон 0–100.

Підсумкова оцінка навички обчислюється як зважена сума результатів усіх її метрик обчислюється за формулою 2.3.

$$SkillScore = \sum(MetricScore * Weight) \quad (2.3)$$

де *Weight* – вага навички.

Такий підхід дозволяє уникати впливу крайніх значень, а також гарантує, що кінцевий бал відображає типовий рівень продуктивності.

Щомісячний моніторинг дозволяє визначати прогрес менті за конкретними навичками. Для цього застосовується формула 2.4.

$$Improvement = \frac{SkillScoreEnd - SkillScoreStart}{50} * 100 \quad (2.4)$$

де: *SkillScoreStart* – результат на початку періоду;

SkillScoreEnd – результат наприкінці періоду;

50 – нормувальний коефіцієнт, що забезпечує інтерпретацію приросту у відсотках.

Для оцінки впливу роботи ментора на розвиток менті використовується формула 2.5.

$$Efficiency = \frac{SkillScoreEnd - SkillScoreStart / AmountOfMonths}{50} * 100 \quad (2.5)$$

де: *AmountOfMonths* – кількість місяців співпраці з ментором.

Цей показник враховує не лише загальний приріст результатів, а й динаміку змін у часі, що дає можливість відстежувати швидкість розвитку менті.

У результаті підхід забезпечує наступне: дані збираються регулярно та базуються на реальних подіях; усі результати нормуються до єдиної шкали; метод підходить для різних видів навичок і метрик завдяки універсальності формул.

2.7 Формування рекомендацій для менті

Головне завдання системи полягає у формуванні рекомендацій для менті щодо вибору найоптимальнішого ментора з урахуванням конкретної навички, яку менті прагне покращити.

Вхідними даними є: досвід ментора, кількість поточних менті, історія попереднього менторингу, вибрана навичка.

Метою є побудова моделі, яка: визначає ймовірний відсоток покращення навичку; прогнозує очікуваний кінцевий рівень навичку; оцінює передбачувану тривалість навчання. На основі цих трьох показників формується рейтинг менторів, з якого система обирає десять найкращих.

Для поставленої задачі найбільш придатним є підхід багатофакторної регресії. Обґрунтування вибору: змінні (стаж, кількість менті, ефективність, тривалість, покращення) мають числову природу й прямий причинно-наслідковий зв'язок з результатами; регресійна модель дозволяє передбачати похідні показники; коефіцієнти регресії показують внесок кожного фактору, що дає змогу пояснювати роботу системи; якщо згодом буде потрібно враховувати нові фактори (наприклад, специфічні методи роботи ментора), вони можуть бути легко інтегровані в регресійну модель.

Для забезпечення коректності та стабільності регресійної моделі перед етапом навчання виконується попередня обробка даних. Саме якість вхідних даних безпосередньо впливає на точність прогнозу, тому застосовуються відповідні методи очищення, нормалізації та відбору атрибутів.

Перед навчанням моделі використовується набір фільтрів Weka для забезпечення якості даних: Weka надасть змогу ідентифікації значення, що виходять за межі міжквартильного розмаху. Це дозволяє уникнути впливу аномальних випадків (наприклад, надзвичайно швидке покращення або навпаки нульовий прогрес); всі числові змінні масштабуються до спільного діапазону. Це гарантує, що показники з великими абсолютними значеннями (наприклад,

кількість місяців) не домінуватимуть у моделі над показниками у відсотках; з використанням певного фільтра, вилучаються ті змінні, які не несуть прогностичної цінності (наприклад, ідентифікатори користувачів).

Під час навчання Weka застосовує класичний метод найменших квадратів для підбору коефіцієнтів регресії (формула 2.6).

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon \quad (2.6)$$

де y – прогнозований показник;

x_n – незалежні змінні (стаж, кількість менті, ефективність, тощо);

β_i – коефіцієнти, що визначають вагомість кожного фактора,

ϵ – похибка моделі.

Для виявлення аутлайнерів Weka застосовує правило (формула 2.7).

$$x < Q1 - 1.5 \cdot IQR \vee x > Q3 + 1.5 \cdot IQR \quad (2.7)$$

де $Q1, Q3$ – нижній та верхній квартилі, а $IQR = Q3 - Q1$.

Таким чином, підхід поєднує: математичну строгість регресії, систематичну очистку даних (IQR, нормалізація, VIF), високу інтерпретованість результатів.

3 АРХІТЕКТУРА СИСТЕМИ

У цьому підрозділі буде розглянуто як зовнішню архітектуру системи, що описує взаємодію клієнта та сервера, так і внутрішню архітектуру серверної частини, що визначає обрану модель побудови застосунку — монолітну або мікросервісну.

3.1 Зовнішня архітектура

Мережа Інтернет заснована на архітектурі клієнт-сервер [24].

Клієнт представляє собою комп'ютер або пристрій користувача, який надсилає запити до сервера для отримання певної інформації або виконання заданих функцій. Сервер, у свою чергу, є більш потужним комп'ютером або обладнанням із встановленим програмним забезпеченням та базою даних, що забезпечує обробку запитів клієнта та надання необхідних файлів, ресурсів або контенту. Клієнт-серверна архітектура зображена на рис. 3.1.

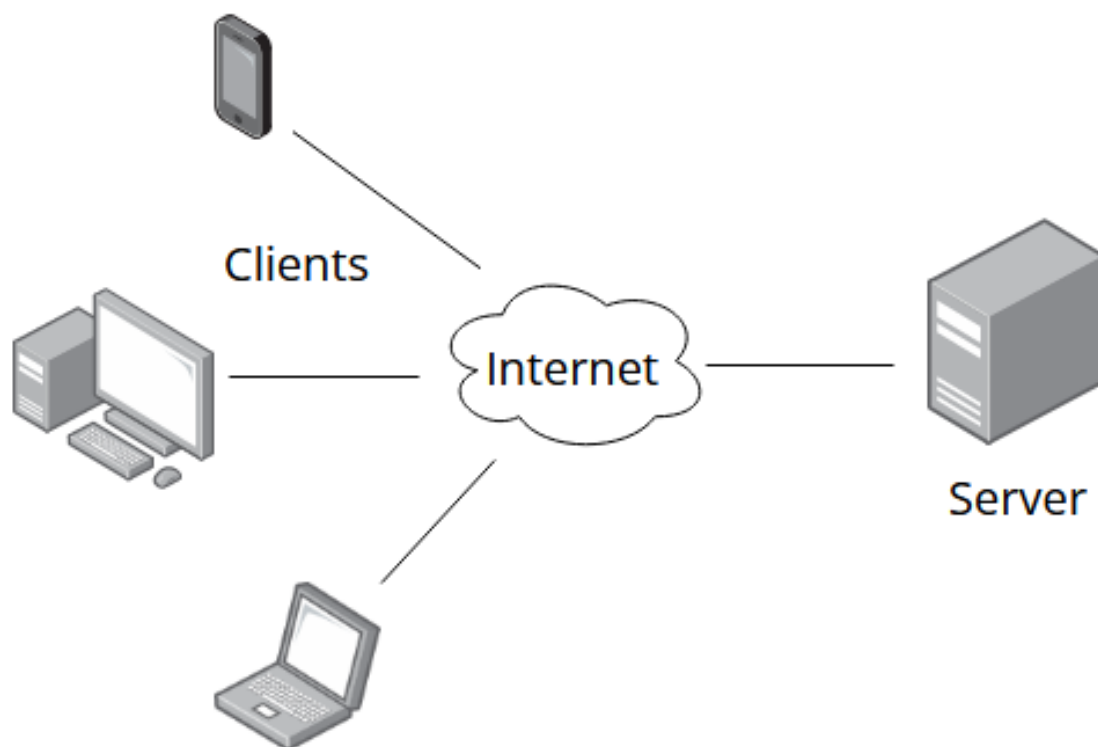


Рисунок 3.1 – Клієнт-серверна архітектура

Слід підкреслити, що взаємодія між клієнтом і сервером відбувається за принципом ініціативи клієнта. Запит від клієнта зазвичай передбачає операції з даними, такі як додавання, отримання, оновлення або видалення, що зумовлює необхідність використання бази даних.

Обмін даними між клієнтом та сервером реалізується через мережеві протоколи, такі як HTTP, FTP, TCP, IP або UDP. Вибір протоколу визначається типом надаваних сервером послуг. Оскільки розробляється веб-застосунок, основним протоколом для передачі даних буде HTTP разом зі стеком TCP/IP. Для забезпечення безпеки переданих даних застосовується протокол HTTPS, який є розширенням HTTP та використовує TLS або SSL для шифрування інформації, на відміну від HTTP, який передає дані у відкритому вигляді.

Сервер обробляє HTTP-запити, витягує необхідну інформацію та передає її для обробки програмі на сервері. Результат обробки упаковується у відповідь HTTP і надсилається клієнтові. Виходячи з цього, серверна програма повинна мати такі властивості: запуск здійснюється сервером; забезпечувати прийом та обробку запитів від клієнта; формувати та надсилати відповідь клієнту.

Клієнтська частина системи реалізується у вигляді веб-браузера.

Перевагою такого підходу є зниження витрат на супровід додатку розробниками, а також дозволить кожному клієнту постійно користуватися самою останньою версією програми, оскільки саме вона завжди буде знаходитися на сервері [24].

3.2 Внутрішня архітектура

Внутрішня архітектура серверної частини визначає, як організовані компоненти системи, як вони взаємодіють між собою та як обробляються запити клієнтів. Сучасні програмні системи зазвичай реалізуються за двома найпоширенішими підходами: монолітною та мікросервісною архітектурою. Кожна з цих архітектур має свої особливості, переваги та обмеження, що визначає доцільність їх застосування залежно від масштабу проекту, вимог до продуктивності та підтримки системи.

В таблиці 3.1 наведено порівняння моноліту з мікросервісами.

Таблиця 3.1 – Порівняння моноліту з мікросервісами

Моноліт	Мікросервіси
Єдина велика кодова база.	Кожен мікросервіс має свою кодову базу.
Впровадження змін у великому додатку є складним процесом.	Впровадження змін є простішим, оскільки робота йде з меншим обсягом коду.
Складність розробки збільшується з розширенням застосунку.	Кожен мікросервіс може розширюватись незалежно.
Легше запустити цілу систему, час запуску залежить від обсягу проєкту.	Необхідні окремі ресурси або окремі сервіси ускладнює запуск системи.
Єдиний дефект впливає на всю систему.	Дефект в одному сервісі має низький шанс вплинути на інші сервіси.
Для одного застосунку легше впровадити захист.	Комунікація між сервісами може призвести до прогалин в безпеці.
Тестування може бути легшим та прямим.	Процес тестування може стати складнішим, оскільки кожен сервіс потребує індивідуального дослідження.
Вся система написана на одній мові.	Кожен сервіс може бути написаний на будь-якій мові.
Більш за все підійде для малих додатків.	Ідеально для велико-масштабованих додатків та опрацювання даних.

Беручи до уваги переваги мікросервісної архітектури, було прийнято рішення обрати саме мікросервісний підхід. Використання цієї архітектури передбачає організацію серверної частини у вигляді набору невеликих, незалежних сервісів, кожен з яких відповідає за певну функціональність. Такий

підхід дозволяє не лише підвищити гнучкість системи, а й ефективно реалізувати окремі бізнес-процеси та інтегрувати їх у єдину екосистему.

У зв'язку з цим особлива увага приділяється застосуванню патернів проектування мікросервісів. Використання таких патернів дозволяє формалізувати архітектурні рішення та створити стійку і легко підтримувану систему.

Так, використавши патерн Database per service, гарантується, що база даних, що виділена для одного сервісу, не може бути доступною для інших сервісів [25]. Це один із факторів, який значно полегшує масштабування системи та її розуміння з точки зору кінцевого бізнес-процесу. Дані, якими володіє один сервіс, можуть бути здебільшого реляційними, тоді як другому сервісу може більше підходити NoSQL-рішення, а третьому сервісу може знадобитися векторна база даних. У такому сценарії використання окремих баз даних для кожного сервісу допомагає легше ними керувати. Сервіси змушені взаємодіяти через опубліковані інтерфейси. Недоліком є те, що виділені бази потребують механізму захисту від збоїв у разі проблем із комунікацією.

Наступний патер Saga – це серія локальних транзакцій. У застосунках на базі мікросервісів патерн saga допомагає підтримувати узгодженість даних під час розподілених транзакцій [25]. Saga є альтернативним рішенням іншим патернам проектування, що дозволяє виконувати кілька транзакцій із можливістю їх відкоту (rollback). Використовуючи підхід хореографії, сервіс виконує транзакцію та публікує подію. В деяких випадках інші сервіси реагують на ці події та виконують завдання відповідно до закодованих інструкцій. Ці вторинні завдання можуть, але не обов'язково, також публікувати події залежно від заданих умов;

Окрім того для великих застосунків із численними клієнтами реалізація патерну API Gateway є доцільним варіантом [25]. Однією з головних переваг є те, що клієнт звільняється від необхідності знати, як розподілені сервіси. Крім того, API Gateway може виконувати ключові завдання, такі як автентифікація, завершення SSL-з'єднання та кешування, що робить застосунок більш безпечним і зручним для користувача. В свою чергу «Service Discovery» дозволить

мікросервісам динамічно знаходити один одного та взаємодіяти через реєстр сервісів. Сервіси реєструють себе у реєстрі та можуть знаходити інші сервіси без ручного налаштування.

Для сервісів, що взаємодіють синхронно, також слід застосувати Circuit Breaker [25]. Він запобігає накопиченню викликів і зайняттю системних ресурсів, що могло б спричинити значні затримки або навіть каскадні відмови сервісів. Реалізація патерну вимагає об'єкта, який відстежує умови збоїв. Коли виявляється збій, спрацьовує circuit breaker. Після цього всі виклики через circuit breaker завершуються помилкою або спрямовуються на інший сервіс. Альтернативно, виклики можуть повертати стандартне повідомлення про помилку. Існують три стани функцій патерну circuit breaker, про які треба знати: circuit breaker відкритий, коли кількість збоїв перевищила поріг і у цьому стані мікросервіс повертає помилки для всіх викликів без виконання бажаної функції; у закритому стані всі виклики обробляються нормально, це ідеальний стан для роботи мікросервісу, за якого система функціонує стабільно; у напіввідкритому стані circuit breaker перевіряє наявність проблем. Деякі виклики можуть оброблятися нормально, інші — ні. Це залежить від того, чому спочатку було переведено breaker у цей стан.

Схема роботи Circuit Breaker зображено на рисунку 3.2.

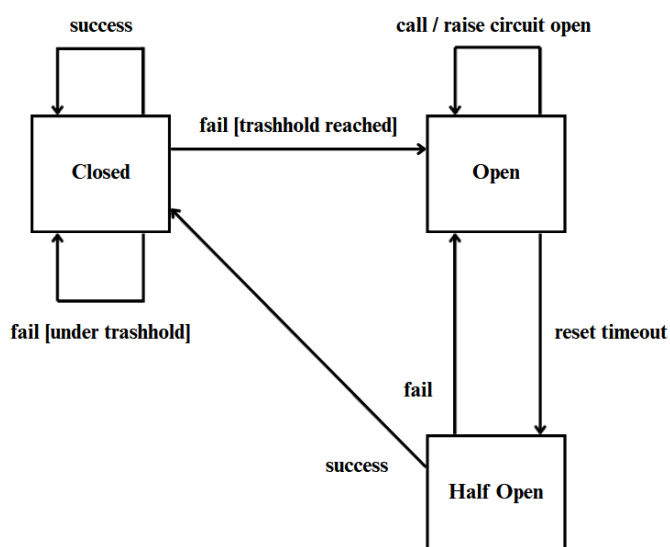


Рисунок 3.2 – Схема роботи Circuit Breaker

Після визначення архітектурного підходу та вибору основних патернів проєктування можна перейти до формування детальної внутрішньої структури системи. Обрані патерни — визначають спосіб організації сервісів, їх взаємодію та обробку даних.

На основі цих рішень стає можливим скласти схему внутрішньої архітектури системи, яка наочно відображає: розподіл компонентів на окремі мікросервіси; шляхи синхронної та асинхронної взаємодії між ними; роботу з базами даних та зовнішніми сервісами; місця застосування патернів для забезпечення надійності та масштабованості.

Такий підхід дозволяє не лише візуалізувати архітектурну організацію системи (рис. 3.3), а й визначити ключові точки інтеграції та потенційні місця для оптимізації.

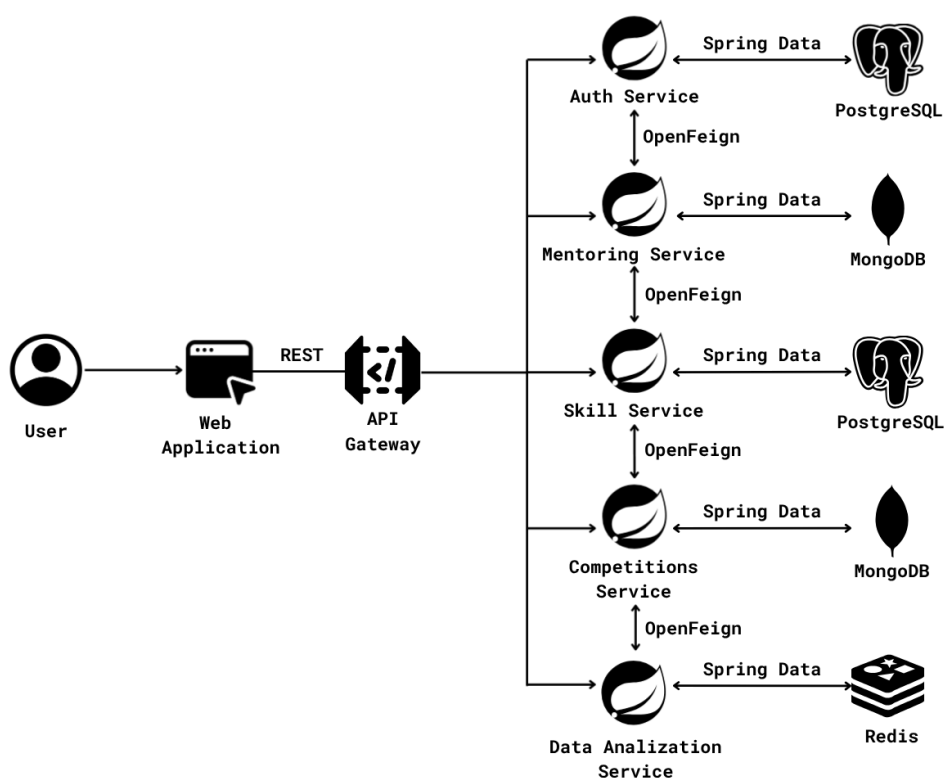


Рисунок 3.3 – Внутрішня архітектура системи

З рисунка 3.3 можна спостерігати, що система складається з п'яти мікросервісів, кожен із яких виконує чітко визначені функції та має власну базу даних.

3.3 Апаратне розгортання архітектури

Апаратне забезпечення мікросервісної системи має відповідати вимогам масштабованості, відмовостійкості та стабільної продуктивності, що обумовлено характером обраної архітектури. Оскільки кожен мікросервіс функціонує як незалежний програмний модуль, апаратна інфраструктура повинна забезпечувати не лише їх фізичну ізоляцію, але й можливість швидкого горизонтального масштабування у разі підвищеного навантаження. У цьому контексті доцільним є використання набору окремих серверів або контейнеризованих середовищ, які розміщуються на декількох фізичних або віртуальних вузлах.

Загальна архітектура інфраструктури передбачає розміщення кожного мікросервісу на окремому сервері або інстансі, що надає змогу ізолювати потенційні збої та оптимізувати споживання ресурсів залежно від функціональних особливостей кожного сервісу. Найбільше навантаження припадає на API Gateway і Auth Service, оскільки саме вони обробляють критичні операції зі значною інтенсивністю запитів. У зв'язку з цим на початковому етапі передбачається розгортання цих компонентів щонайменше у двох екземплярах для забезпечення безперервності роботи та можливості балансування навантаження. Інші мікросервіси, такі як Mentoring Service, Skill Service, Competitions Service та Data Analysis Service, можуть функціонувати на одному виділеному сервері кожен, проте їх конфігурація також враховує потенційну необхідність масштабування.

Рекомендована конфігурація серверів застосунків включає від чотирьох до восьми віртуальних процесорів, від восьми до шістнадцяти гігабайтів оперативної пам'яті та твердотільні накопичувачі обсягом не менше 50–100 гігабайтів. Такі параметри забезпечують достатній рівень продуктивності Spring Boot застосунків та дозволяють уникнути затримок при обробці бізнес-логіки. Усі сервери мають бути з'єднані у внутрішню мережу з пропускною здатністю не менше одного

гігабіта на секунду, що є важливим чинником для мінімізації затримок під час взаємодії між мікросервісами.

Важливою особливістю апаратної частини є розгортання окремих баз даних відповідно до принципів патерну Database per Service. Так, для Auth Service та Skill Service застосовується реляційна база даних PostgreSQL, яка вимагає від серверів високої швидкості дискових операцій. З цією метою рекомендується використовувати NVMe-накопичувачі, об'єднані у RAID 10 для підвищення надійності та продуктивності. Сервери, на яких розміщується PostgreSQL, повинні містити щонайменше чотири віртуальні ядра, від восьми до шістнадцяти гігабайтів оперативної пам'яті та накопичувачі обсягом до двохсот гігабайтів.

Для розміщення документно-орієнтованих баз даних MongoDB, що використовуються в Mentoring Service та Competitions Service, достатньою є конфігурація з чотирма віртуальними ядрами, вісьмома гігабайтами оперативної пам'яті та SSD-накопичувачем обсягом не менше ста гігабайтів. MongoDB характеризується високою гнучкістю та здатністю до горизонтального масштабування шляхом додавання нових шард- або репліка-сетів, тому апаратна інфраструктура повинна передбачати можливість такого розширення.

Окрему категорію становить Redis, який використовується в Data Analysis Service для кешування даних та забезпечення високошвидкісного доступу до інформації. Оскільки Redis повністю працює в оперативній пам'яті, сервер для його розміщення має містити не менше чотирьох-восьми гігабайтів RAM та процесор із двома-чотирма віртуальними ядрами. Дискові ресурси Redis не є критичними, проте для забезпечення надійності збереження snapshot-даних рекомендується використовувати SSD-накопичувач.

Забезпечення високої доступності системи значною мірою залежить від механізмів балансування навантаження. На рівні API Gateway доцільно задіяти високопродуктивні програмні або апаратні балансувальники, такі як Nginx чи HAProxy, що працюють на сьомому рівні моделі OSI та можуть інтелектуально розподіляти HTTP-трафік. На рівні сервісів балансування здійснюється за допомогою механізмів Service Discovery, які дозволяють динамічно визначати

доступні інстанси та виконувати маршрутизацію запитів без необхідності ручної конфігурації.

Компоненти, відповідальні за збереження даних, також можуть функціонувати у відмовостійких конфігураціях. MongoDB може бути розгорнута у вигляді репліка-сету, що забезпечує підвищену доступність та збереження даних у разі виходу з ладу окремих вузлів. PostgreSQL може використовувати реплікацію для розділення навантаження між основним сервером запису та допоміжними вузлами читання. Redis у свою чергу може працювати у конфігурації Redis Sentinel або Redis Cluster, що забезпечує автоматичне перемикання на резервний вузол у разі несправності основного.

Зібравши до купи усі елементи архітектури було створено UML схему розгортання ресурсів, яка наведена на рисунку 3.4.

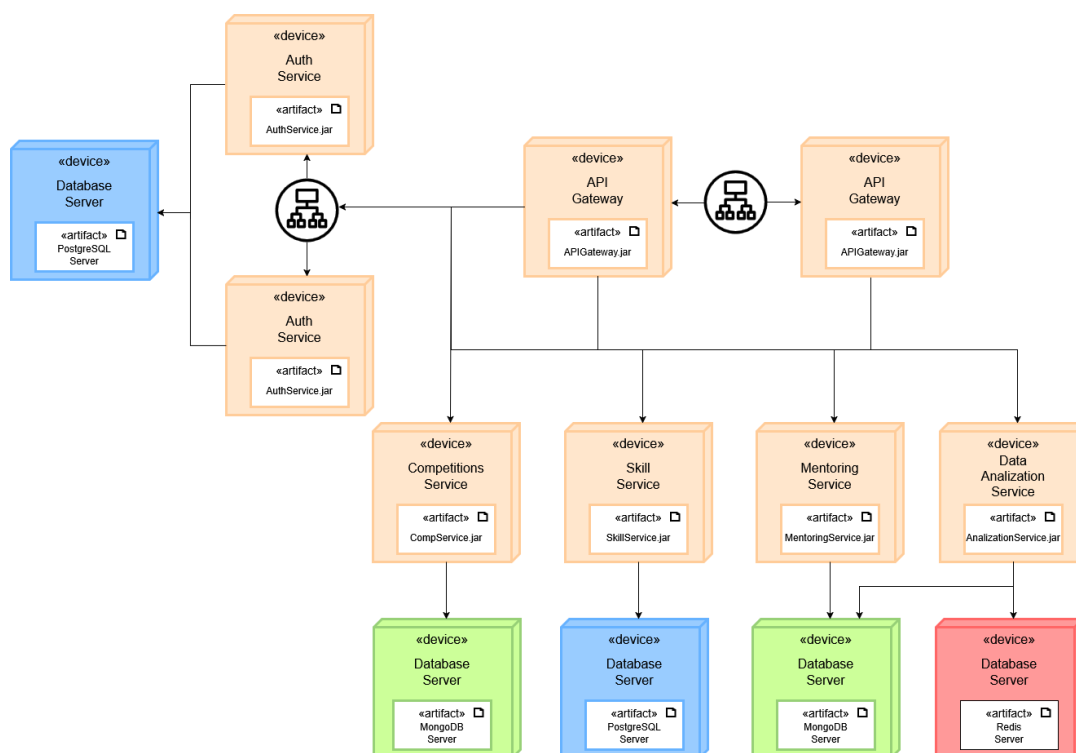


Рисунок 3.4 – UML розгортання ресурсів

Таким чином, апаратне забезпечення системи формує основу для її ефективного функціонування, забезпечуючи ізольованість, продуктивність та відмовостійкість кожного мікросервісного компонента.

4 ПРОГРАМНА РОЗРОБКА СИСТЕМИ

4.1 Розробка рекомендаційної частини

На цьому етапі було реалізовано рекомендаційний модуль, який на основі історичних даних про менторинг формує прогноз ефективності кожного ментора. Для цього використано бібліотеку Weka, що забезпечує зручні засоби для навчання, оцінювання та використання моделей машинного навчання у середовищі Java.

Клас DatasetPreprocessor реалізує етап попередньої обробки даних, який є одним із ключових етапів підготовки даних до машинного навчання. Його основним призначенням є формування навчального набору даних у форматі, сумісному з бібліотекою Weka, а також обчислення похідних характеристик, нормалізація структур даних та видалення аномальних значень.

Метод createStructure створює початкову структуру набору даних, тобто перелік атрибутів та їх типів (лістинг 4.1).

Лістинг 4.1 – Код створення структури набору даних

```
public static Instances createStructure() {
    ArrayList<Attribute> attributes = new ArrayList<>();
    attributes.add(new Attribute("yearsActive"));
    attributes.add(new Attribute("totalMentees"));
    attributes.add(new Attribute("costPerMonth"));
    attributes.add(new Attribute("avgImprovementPercent"));
    attributes.add(new Attribute("avgDurationMonths"));
    attributes.add(new Attribute("efficiencyPercent"));
    attributes.add(new Attribute("durationMonths"));
    attributes.add(new Attribute("improvementPoints"));
    attributes.add(new Attribute("improvementPercent"));
    attributes.add(new Attribute("desiredSkill", SKILLS));

    Instances structure = new Instances("MentorPerformance",
attributes, 0);
    structure.setClassIndex(-1);
    return structure;
}
```

Отримана структура використовується під час створення екземплярів та навчання моделей.

Метод `removeOutliers` реалізує очищення набору даних від аномальних значень для вказаного атрибуту. Для цього обчислюються середнє значення (mean) та стандартне відхилення (standard deviation), після чого всі спостереження, які відхиляються від середнього більш ніж на три стандартних відхилення, видаляються з вибірки (лістинг 4.2).

Лістинг 4.2 – Код очищення набору даних

```
public static Instances removeOutliers(Instances data, String
targetAttr) {
    Attribute attr = data.attribute(targetAttr);
    if (attr == null) return data;

    double mean = data.meanOrMode(attr);
    double stdDev = Math.sqrt(data.variance(attr));

    Instances filtered = new Instances(data);
    int removed = 0;
    for (int i = filtered.numInstances() - 1; i >= 0; i--) {
        double val = filtered.instance(i).value(attr);
        if (Math.abs(val - mean) > 3 * stdDev) {
            filtered.delete(i);
            removed++;
        }
    }
    System.out.printf("Removed %d outliers for %s\n", removed,
targetAttr);
    return filtered;
}
```

Таке попереднє очищення дозволяє зменшити вплив шуму та покращити узагальнювальну здатність моделі.

Метод `buildInstanceValues` формує числове представлення одного запису даних. Він розраховує низку похідних характеристик, які відображають як загальний досвід ментора, так і специфіку окремої сесії. Зокрема, тут обчислюються: кількість років активності ментора, розрахована на основі дати початку діяльності; середній відсоток покращення результатів серед усіх менторських сесій; середня тривалість усіх сесій у місяцях. Крім того,

номінальний атрибут `desiredSkill` перетворюється у числовий індекс відповідного значення у структурі `Instances` (лістинг 4.3).

Лістинг 4.3 – Код формування запису даних

```
private static double[] buildInstanceValues(Mentor mentor,
MentoringSession s, Instances structure) {
    double yearsActive = calculateYearsActive(mentor.getSince());
    double avgImprovement = mentor.getMentoring().stream()
.mapToDouble(MentoringSession::getImprovementPercent).average().orElse(0);
    double avgDuration = mentor.getMentoring().stream()
.mapToDouble(MentoringSession::getDurationMonths).average().orElse(0);
    double[] vals = new double[structure.numAttributes()];
    vals[0] = yearsActive; vals[1] = mentor.getTotalMentees();
    vals[2] = mentor.getCostPerMonth();
    vals[3] = avgImprovement; vals[4] = avgDuration;
    vals[5] = s.getEfficiencyPercent();
    vals[6] = s.getDurationMonths();
    vals[7] = s.getImprovementPoints();
    vals[8] = s.getImprovementPercent();
    vals[9] = structure.attribute("desiredSkill").indexOfValue(s.getDesiredSkill());
    return vals;
}
```

Отриманий масив чисел описує усі релевантні характеристики ментора та його сесії в кількісному вигляді.

Метод `toInstance` відповідає за створення окремого екземпляра даних (об'єкта типу `DenseInstance`) для пари ментор – менторська сесія. На вхід подається об'єкт ментора, конкретна сесія та структура даних, що описує набір атрибутів. Метод викликає допоміжну функцію `buildInstanceValues()`, яка формує масив числових значень для всіх атрибутів, після чого створює екземпляр та прив'язує його до загальної структури `Instances` (лістинг 4.4).

Лістинг 4.4 – Створення екземпляру даних

```
public static DenseInstance toInstance(Mentor mentor, MentoringSession
s, Instances structure) {
    double[] vals = buildInstanceValues(mentor, s, structure);
    DenseInstance instance = new DenseInstance(1.0, vals);
    instance.setDataset(structure);
    return instance;
}
```

Таким чином, кожен рядок датасету відповідає окремій менторській взаємодії.

Метод `prepareDataset` формує повний навчальний набір даних на основі вихідного списку менторів та інформації про їхні менторські сесії. Спочатку створюється порожня структура набору даних за допомогою методу `createStructure()`, після чого кожен об'єкт типу `Mentor` та його вкладені сесії (`MentoringSession`) перетворюються у формат, сумісний із Weka (лістинг 4.6).

Лістинг 4.6 – Код формування повного начального набору даних

```
public static Instances prepareDataset(List<Mentor> mentors) {
    Instances dataset = createStructure();
    for (Mentor m : mentors) {
        for (MentoringSession s : m.getMentoring()) {
            DenseInstance instance = toInstance(m, s, dataset);
            dataset.add(instance);
        }
    }
    return dataset;
}
```

Отримані екземпляри додаються до об'єкта типу `Instances`, що у підсумку утворює повний датасет, придатний для подальшого навчання моделей прогнозування.

Клас `Modeller` виконує ключову роль у побудові, навчанні, збереженні та подальшому використанні моделей машинного навчання для прогнозування ефективності менторів. Його функціональність охоплює повний цикл моделювання — від підготовки даних і тренування моделей до здійснення передбачень на основі раніше збережених моделей. Цей клас є центральним елементом системи рекомендацій, оскільки саме він перетворює оброблений датасет у модель, здатну прогнозувати показники покращення учнів і тривалість навчання на основі характеристик менторів та їхніх сесій.

Функція `trainAllModels` виконує навчання всіх моделей, визначених у системі. Вона послідовно проходить за списком цільових змінних, зокрема `improvementPoints` і `durationMonths`, і для кожної з них викликає метод навчання та збереження моделі. Таким чином, забезпечується створення окремих моделей для

прогнозування різних аспектів менторської діяльності, що дозволяє досягти більшої точності та гнучкості під час подальшого використання (лістинг 4.7).

Лістинг 4.7 – Функція початку навчання

```
public static void trainAllModels(Instances dataset) throws Exception {
    for (String target : TARGETS) {
        trainAndSaveModel(dataset, target, getModelPath(target));
    }
}
```

Метод `trainAndSaveModel` реалізує основний процес побудови моделі (лістинг 4.8).

Лістинг 4.8 – Код побудови моделі

```
private static void trainAndSaveModel(Instances dataset, String
targetAttr, String modelPath) throws Exception {
    Instances trainData = new Instances(dataset);
    for (String attr : TARGETS) {
        if (!attr.equals(targetAttr) && trainData.attribute(attr) !=
null) {
            trainData.deleteAttributeAt(trainData.attribute(attr).index());
        }
    }

    trainData.setClassIndex(trainData.attribute(targetAttr).index());
    trainData = DatasetPreprocessor.removeOutliers(trainData,
targetAttr);
    NominalToBinary nominalToBinary = new NominalToBinary();
    nominalToBinary.setInputFormat(trainData);

    trainData = Filter.useFilter(trainData, nominalToBinary);
    Normalize normalize = new Normalize();
    normalize.setInputFormat(trainData);

    trainData = Filter.useFilter(trainData, normalize);
    Classifier model = new LinearRegression();
    model.buildClassifier(trainData);

    Evaluation eval = new Evaluation(trainData);
    eval.crossValidateModel(model, trainData, 5, new Random(1));
    System.out.printf("Model %s RMSE: %.4f\n", targetAttr,
eval.rootMeanSquaredError());
    weka.core.SerializationHelper.writeAll(modelPath, new
Object[]{model, nominalToBinary, normalize});
    System.out.println("Model " + targetAttr + " saved -> " +
modelPath);
}
```

На початковому етапі створюється копія вихідного датасету, після чого з нього видаляються непотрібні цільові атрибути, щоб уникнути витоків інформації між моделями. Потім встановлюється цільова змінна, за якою здійснюється навчання, і проводиться очищення даних від викидів із використанням методу `DatasetPreprocessor.removeOutliers`. Подальша обробка включає перетворення номінальних атрибутів у бінарні за допомогою фільтра `NominalToBinary`, що дозволяє враховувати категоріальні ознаки у числовій формі, а також нормалізацію даних через фільтр `Normalize`, яка забезпечує однакову шкалу для всіх атрибутів. Після підготовки даних модель типу `LinearRegression` проходить процес навчання. Для оцінки якості побудованої моделі використовується п'ятикратна крос-валідація, яка дозволяє мінімізувати ризик перенавчання та перевірити узагальнюючу здатність алгоритму. Після завершення навчання модель разом із фільтрами серіалізується та зберігається у файл для подальшого використання.

Функція `predictForSkill` забезпечує етап інференсу, тобто передбачення результатів на основі вже навчених моделей (лістинг 4.9).

Лістинг 4.9 – Код передбачення результатів вже навчених моделей

```
public static List<MentorPredictionDTO> predictForSkill(String skill,
List<Mentor> mentors, int topK) throws Exception {
    Instances structure = DatasetPreprocessor.createStructure();
    Object[] pointsModelData =
    weka.core.SerializationHelper.readAll(MODEL_PATH_POINTS);
    Object[] durationModelData =
    weka.core.SerializationHelper.readAll(MODEL_PATH_DURATION);
    List<MentorPredictionDTO> results = new ArrayList<>();
    for (Mentor mentor : mentors) {
        Optional<MentoringSession> example = mentor.getMentoring().stream()
        .filter(s -> s.getDesiredSkill().equals(skill)).findFirst();
        if (example.isEmpty()) continue;
        MentoringSession session = example.get();
        DenseInstance instance = DatasetPreprocessor.toInstance(mentor,
        session, structure);
        double predictedPoints = predictValue(pointsModelData,
        structure, instance, "improvementPoints");
        double predictedDuration = predictValue(durationModelData,
        structure, instance, "durationMonths");
        results.add(new MentorPredictionDTO(
        mentor.getId(), mentor.getFirstname(), mentor.getLastname(), mentor.getMentori
        ngLocation(), mentor.getCostPerMonth(),
        round(predictedPoints, 2), round(predictedDuration, 2)));
    }
```

Функція завантажує раніше збережені моделі для обох цільових змінних і створює нову структуру датасету, сумісну з ними. Для кожного ментора здійснюється вибір сесії, що відповідає заданій навичці (skill), і на основі її характеристик формується об'єкт DenseInstance. Потім для кожної моделі викликається функція передбачення, що повертає прогнозовані значення покращення та тривалості навчання. Усі отримані результати збираються у список об'єктів MentorPredictionDTO, який містить інформацію про ментора, його передбачувані результати та основні характеристики. На завершальному етапі список сортується за двома критеріями — зменшенням прогнозованого покращення та збільшенням тривалості навчання — і обрізається до визначеної кількості найкращих кандидатів (topK). Таким чином, метод забезпечує формування рейтингу менторів, що відповідає заданій навичці.

Метод predictValue реалізує процес передбачення для однієї моделі. Він розпаковує модель та відповідні фільтри з файлу, застосовує ті самі етапи перетворення даних, які були використані під час навчання, і передає оброблений приклад у модель для класифікації. Це гарантує консистентність між навчальною та прогножною фазами. Отримане значення є прогнозом для конкретного атрибуту, наприклад improvementPoints або durationMonths (лістинг 4.10).

Лістинг 4.10 – Код процесу передбачення для моделі

```
private static double predictValue(Object[] modelData, Instances
structure, DenseInstance instance, String targetAttr) throws Exception {
    Classifier model = (Classifier) modelData[0];
    NominalToBinary nominalToBinary = (NominalToBinary) modelData[1];
    Normalize normalize = (Normalize) modelData[2];
    Instances temp = new Instances(structure, 1);
    temp.add(instance);
    removeOtherTargets(temp, targetAttr);
    temp = Filter.useFilter(temp, nominalToBinary);
    temp = Filter.useFilter(temp, normalize);
    return model.classifyInstance(temp.instance(0));
}
```

Допоміжний метод removeOtherTargets відповідає за видалення інших цільових атрибутів зі структури даних, які не є релевантними для поточного прогнозу.

Це запобігає потенційному витoku інформації, який міг би спотворити результати передбачення. Функція `round` виконує округлення числових результатів до заданої кількості десяткових знаків, що підвищує зручність відображення та зчитування результатів.

Лістинг 4.11 – Код видалення не релевантних атрибутів

```
private static final List<String> TARGETS =
List.of("improvementPoints", "durationMonths");
private static void removeOtherTargets(Instances data, String
targetAttr) {
    for (String attr : TARGETS) {
        if (!attr.equals(targetAttr) && data.attribute(attr) !=
null) {
            data.deleteAttributeAt(data.attribute(attr).index());
        }
    }
}
```

У цілому клас `Modeller` формує логічну основу модуля машинного навчання в системі. Він інтегрує всі основні етапи моделювання — від навчання до прогнозування — і забезпечує стабільність, відтворюваність та точність процесу.

Його архітектура побудована таким чином, щоб забезпечити повторне використання моделей без необхідності повторного навчання, що є критично важливим для ефективної роботи рекомендаційної системи.

Клас `MentorRecommender` виконує функцію високорівневого координатора, який об'єднує всі етапи процесу формування рекомендацій для вибору найефективніших менторів.

Його основним завданням є інтеграція компонентів попередньої обробки даних, навчання моделей та здійснення прогнозів на основі вже наявних або щойно створених моделей машинного навчання. Цей клас виступає посередником між сирими даними про менторів, збереженими у форматі JSON, і кінцевим користувачем, який отримує ранжований список рекомендацій залежно від заданої навички (лістинг 4.12).

Лістинг 4.12 – Клас MentorRecommender

```
public class MentorRecommender {
    private static final JSON_PATH = "data/mentors.json";
    public List<MentorPredictionDTO> getRecommendations(String
desiredSkill, int topK, boolean withTraining) {
        List<Mentor> mentors =
JsonReader.loadMentorsFromJson(jsonPath);
        if (withTraining) {
            Instances dataset =
DatasetPreprocessor.prepareDataset(mentors);
            Modeller.trainAllModels(dataset);
        }
        return Modeller.predictForSkill(desiredSkill, mentors, topK);
    }
}
```

Така архітектура забезпечує інкапсуляцію складної логіки підготовки даних, навчання моделей та прогнозування у єдиному методі, що робить процес формування рекомендацій простим, гнучким і відтворюваним.

Завдяки цьому клас може легко інтегруватися у REST API або веб-додатки, де користувачеві необхідно лише вказати бажану навичку та отримати найкращих менторів, визначених на основі статистично обґрунтованих моделей машинного навчання.

4.2 Розробка серверної частини

На цьому етапі розробки передбачається створення серверного застосунку у вигляді REST API, на прикладі сервісу аналізу даних, до якого надалі зможуть звертатися клієнтські компоненти системи для отримання рекомендацій.

При ініціалізації проекту були встановлені такі параметри як: тип проекту – Maven, версія Spring Boot – 3.4.12, мова програмування – Java 17. Також з огляду на функціонал проекту було додано наступні залежності:

- Lombok: Java бібліотека анотацій, яка допоможе значно зменшити час на написання шаблонного коду (наприклад гетери та сетери);

- Spring Web: фреймворк, який використовується для побудови веб, включно з RESTful, який забезпечує надійну та гнучку архітектуру для розробки веб додатків. Включає в себе веб-сервер Tomcat;
- Resilience4J: реалізований патерн Circuit Breaker;
- Spring Data Redis та MongoDB: частина широкої родини Spring Data (проект який включає в себе багато під проектів, що спеціалізуються на доступі до даних), що спрощує впровадження функціоналу для управління NoSQL базами даних;
- Spring Security: фреймворк, який спрямований на надання аутентифікації та авторизації для Java-додатків.

Далі була проведена робота над логікою функціонування сервісу.

Клас MentorRecommendationDTO виконує роль об'єкта передавання даних, який використовується для представлення інформації про ментора під час обміну даними між сервером та клієнтом (лістинг 4.13).

Лістинг 4.13 – Клас MentorRecommendationDTO

```
@Data
public class MentorRecommendationDTO {
    private String id;
    private String firstname;
    private String lastname;
    private String mentoringLocation;
    private double costPerMonth;
    private double predictedImprovementPoints;
    private double predictedDurationMonths;
}
```

Використання Lombok анотації @Data дозволяє автоматично згенерувати стандартні методи доступу.

Клас RecommendationService виконуватиме роль сервісного компонента в архітектурі застосунку, відповідального за формування рекомендацій менторів відповідно до запитуваної навички. Його основним завданням є забезпечення взаємодії між рівнем бізнес-логіки та підсистемою прогнозування (лістинг 5.14).

Лістинг 4.14 – Клас RecommendationService

```
@Service
@RequiredArgsConstructor
public class RecommendationService {
    @Cacheable(value = "recommendations", key = "#desiredSkill")
    public List<MentorRecommendationDTO> getRecommendations(String
desiredSkill, int topK) {
        return Modeller.predictForSkill(desiredSkill, topK);}}

```

Клас позначено анотацією `@Service`, що вказує на його належність до сервісного шару та дає змогу Spring Framework автоматично виявляти і керувати його життєвим циклом у контексті інверсії керування.

Для спрощення створення екземпляра класу використано Lombok анотацію `@RequiredArgsConstructor`, яка генерує конструктор із параметрами для всіх фінальних полів.

Основна логіка побудови рекомендацій делегується до зовнішнього модуля моделювання, що реалізує алгоритмічну частину процесу прогнозування.

Метод позначено анотацією `@Cacheable`, яка активує механізм кешування результатів виконання. Використання параметрів `value = "recommendations"` та `key = "#desiredSkill"` означає, що результати запитів для кожної унікальної навички зберігаються у кеші під відповідним ключем.

Клас `MentorController` реалізує рівень представлення серверного застосунку та відповідає за оброблення HTTP-запитів, пов'язаних із отриманням рекомендацій менторів.

Цей клас забезпечує механізм взаємодії між користувачами і сервісною логікою, що міститься на більш глибоких рівнях програми. Він отримує запити від клієнтських застосунків, такі як запит на отримання рекомендацій менторів, та передає їх на відповідні сервіси, де відбувається основна обробка даних. Далі, після виконання логіки, контролер повертає оброблені результати у вигляді відповіді на запит користувача.

Його основне призначення полягає у забезпеченні інтерфейсу взаємодії між клієнтськими застосунками та бізнес-логікою системи, що реалізована у відповідних сервісних компонентах (лістинг 4.15).

Лістинг 4.15 – Клас

```

@RestController
@RequestMapping(path = "/api/v1/recommendations", produces =
"application/json")
@RequiredArgsConstructor
public class RecommendationController {
    private final RecommendationService recommendationService;
    @GetMapping
    public List<MentorRecommendationDTO> getRecommendations(
        @RequestParam String desiredSkill,
        @RequestParam(defaultValue = "10", required = false) int
topK
    ) {
        return recommendationService.getRecommendations(desiredSkill,
topK);
    }
}

```

Клас позначено анотацією `@RestController`, яка поєднує функціональність анотацій `@Controller` та `@ResponseBody`. Це означає, що всі методи класу автоматично повертають результати у вигляді JSON-об'єктів, без необхідності додаткового маркування кожного методу. Анотація `@RequestMapping` визначає базовий шлях для всіх HTTP-запитів, які обробляються цим контролером. У даному випадку встановлено маршрут `/api/v1/recommendations`, а також зазначено, що відповідь формується у форматі `application/json`. Це гарантує узгодженість маршрутизації в межах системи та чітко визначає формат передавання даних.

У класі реалізовано метод `getRecommendations`, який обробляє HTTP-запити типу GET. Метод позначено анотацією `@GetMapping`, що визначає маршрут виклику цього методу безпосередньо за базовим шляхом контролера. Він приймає два параметри запиту: `desiredSkill`, який визначає навичку, за якою здійснюється пошук менторів, та `topK`, який задає кількість рекомендацій, що мають бути повернуті (за замовчуванням — 10). Обидва параметри марковані анотацією `@RequestParam`, що забезпечує їх автоматичне отримання з URL-запиту.

Розроблена логіка серверної частини забезпечує можливість клієнтських застосунків отримувати доступ до рекомендацій менторів у стандартизований спосіб через REST API. Крім того, використання фреймворку Spring продемонструвало високу зручність і простоту в реалізації серверної логіки, оскільки його модульна архітектура дозволяють швидко створювати нові сервіси

без необхідності значних витрат часу та ресурсів. Це підтверджує доцільність застосування Spring як основи для побудови масштабованих та розширюваних серверних рішень, що зіграє значну роль у подальшому розвитку системи.

4.3 Розробка клієнтської частини

Для ініціалізації проекту потрібно ввести команду: `ng new mentor-recommendation-system`. Після чого будуть встановлені усі необхідні пакети та залежності, які потрібні Angular і можна перейти до створення сервісу і копоненту. Для запуску Angular додатку треба прописати команду: `ng serve --open`. Після чого програма буде запущена використовуючи порт 4200 за замовченням і для того щоб відкрити її з браузера, слід звернутися за URL: `http://localhost:4200/`.

У контексті Angular, сервіс - це об'єкт, який забезпечує спільну функціональність для різних частин додатку. Сервіси використовуються для організації та спільного використання логіки, яка не пов'язана безпосередньо з відображенням інтерфейсу користувача.

Для створення сервісу було прописано команду: `ng generate service recommendation`. Після чого були створені файли “`recommendation.service.ts`” та “`recommendation.service.spec.ts`”, один для написання логіки, інший для тестування відповідно. В лістингу 4.16 наведено методи для створення отримання рекомендацій.

Лістинг 4.16 – Метод отримання рекомендацій

```
@Injectable({providedIn: 'root'})
export class RecommendationService {
  private readonly apiUrl = '/api/v1/recommendations';
  constructor(private http: HttpClient) {}
  getRecommendations(desiredSkill: string, topK: number = 10):
  Observable<MentorRecommendationDTO[]> {
    const params = new HttpParams().set('desiredSkill',
desiredSkill).set('topK', topK);
    return this.http.get<MentorRecommendationDTO[]>(this.apiUrl, {
params });}}}
```

Анотація `@Injectable` означає, що Angular автоматично надає цей сервіс в корені застосунку (singleton), тобто він буде створений один раз і доступний в усіх компонентах і сервісах, які його інжектують.

Поле `apiURL` вміщує у собі базовий URL для серверного API, до якого будуть надходити запити. Метод `getRecommendations` виконує HTTP метод GET, що і надасть змогу повернути список ментрів.

В контексті Angular, компонент є основним будівельним блоком додатку. Компонент відповідає за відображення певної частини користувацького інтерфейсу і управління відповідною логікою. Для створення компоненту було прописано команду: `ng generate component recommendations-component`. Після чого було створено чотири файли: файл класу компонента, шаблон компонента, стилі компонента, тестовий файл компонента. Створений клас компоненту з описаною логікою наведено в лістингу 4.17.

Лістинг 4.17 – Клас компоненту `recommendations.component.ts`

```
@Component({
  selector: 'app-recommendations',
  templateUrl: './recommendations.component.html',
  styleUrls: ['./recommendations.component.css']
})
export class RecommendationsComponent implements OnInit {
  recommendations: MentorRecommendationDTO[] = [];
  desiredSkill = ''; topK = 10;
  constructor(
    private recommendationService: RecommendationService,
    private route: ActivatedRoute) {}
  ngOnInit(): void {
    this.route.queryParamMap.subscribe(params => {
      const skill = params.get('desiredSkill');
      const topKParam = params.get('topK');
      if (skill) {this.desiredSkill = skill; }
      if (topKParam && !isNaN(+topKParam)) { this.topK =
topKParam; }this.loadRecommendations();});}
  loadRecommendations(): void {if
(!this.desiredSkill) {
    console.warn('desiredSkill parameter is missing in URL.');
```

```
    return;}
    this.recommendationService
      .getRecommendations(this.desiredSkill,this.topK).subscribe({
        next: (data) => this.recommendations = data,
        error: (err) => console.error('Failed to load
recommendations', err)
      });}
  getRankClass(index: number): string { const rank = index + 1;
  if (rank === 1) return 'rank rank-1'; if (rank === 2) return 'rank rank-2';
  if (rank === 3) return 'rank rank-3'; return 'rank num';}}
```

Компонент реалізує інтерфейс `OnInit`, що зобов'язує його мати метод `ngOnInit`. Angular автоматично викликає цей метод у момент створення та початкової ініціалізації компонента.

Декоратор `@Component` задає метадані, необхідні для визначення поведінки та структури компонента. Параметр `selector` встановлює назву HTML-тега, через який компонент відображатиметься в шаблоні. Властивість `standalone` вказує, що компонент є незалежним і може існувати без оголошення в окремому модулі. Через масив `imports` визначаються модулі, доступні всередині компонента: вони надають додаткову функціональність та компоненти, наприклад, `CommonModule` забезпечує базові директиви Angular (такі як `ngIf` і `ngFor`), а модулі `Angular Material` — елементи інтерфейсу. Параметр `templateUrl` посилається на HTML-файл, що визначає візуальну структуру компонента, а `styleUrl` містить шлях до SCSS-файлу зі стилями, які застосовуються до нього.

В лістингу 4.18 шаблон компоненту

Лістинг 4.18 – Шаблон компоненту `recommendations.component.html`

```
<ol class="list" aria-label="Top mentors list">
  <li *ngFor="let mentor of recommendations; let i = index"
      class="card" [attr.aria-posinset]="i + 1">
    <div [ngClass]="getRankClass(i)">{{ i + 1 }}</div>
    <div class="main"><p class="name">
      <a [routerLink]="['/mentors', mentor.id]">{{
mentor.firstname }} {{ mentor.lastname }}</a></p>
      <div class="meta"> Location: {{ mentor.mentoringLocation
}}<span class="chip">predicted: {{ mentor.predictedImprovementPoints |
number:'1.1-1' }} pts</span>
      <span class="chip">duration: {{
mentor.predictedDurationMonths | number:'1.1-1' }} mo</span>
      </div>
    </div><div class="right">
      <div class="money">${{ mentor.costPerMonth |
number:'1.2-2' }} / mo</div></div></li></ol>
```

Після ініціалізації компонент отримує значення параметрів `desiredSkill` та `topK` із рядка запиту та викликає метод `loadRecommendations()`, що звертається до `RecommendationService` для отримання даних з бекенду. HTML-шаблон компонента формує впорядкований список менторів із зазначенням їхнього імені, локації, вартості менторингу, прогнозованого покращення та тривалості навчання.

Перші три місця в рейтингу виділяються різними стилями через метод `getRankClass()`.

За аналогічним принципом було реалізовано й інші структурні компоненти клієнтської частини програмного забезпечення, що забезпечує узгодженість архітектурних рішень, підтримуваність коду та уніфікованість підходів до розроблення інтерфейсу користувача. Код зі шаблоном профіля менті для відображення графіку та стилями наведено в додатках А та Б відповідно.

Маршрутизація в Angular відповідає за організацію переходів між різними компонентами застосунку, дозволяючи користувачам переміщуватися між окремими сторінками чи представленнями через зміну URL-адреси. За цей механізм відповідає Angular Router — спеціальний модуль, що керує навігацією. Набір маршрутів задається у вигляді масиву `Routes`, де кожен об'єкт визначає конкретний шлях та компонент, який має бути відображений. Директива `RouterOutlet` резервує область у шаблоні, у якій з'являтиметься відповідний компонент при переході за маршрутом. Для створення навігаційних посилань використовується директива `RouterLink`, що дозволяє легко перемикатися між заданими маршрутами.

В лістинг 4.19 наведено налаштування маршрутизації між компонентами розробленого додатку.

Лістинг 4.19 – Налаштування маршрутизації

```
export const routes: Routes = [
  {
    path: '',
    redirectTo: 'home',
    pathMatch: 'full'
  },
  { path: 'home', component: HomeComponent },
  { path: 'auth', component: AuthComponent },
  { path: 'profile', component: ProfileComponent },
  { path: 'mentors/:id', component: MentorComponent },
  { path: 'menties/:id', component: MenteeComponent },
  { path: 'recommenations/:skill', component:
RecommendationsComponent },
  { path: '**', component: PageNotFoundComponent }
];
```

Так, якщо звернутися з URL “http://localhost:4200/” Angular автоматично відкриє компонент HomeComponent. Якщо буде здійснено звернення за URL, який не було налаштовано, то буде відкрито компонент PageNotFoundComponent. В головному шаблоні додатку буде використано RouterOutlet (лістинг 4.20).

Лістинг 4.20 – HTML шаблон головного компоненту додатку

```
<div>
  <div>
    <app-navbar></app-navbar>
    <router-outlet></router-outlet>
  </div>
  <app-footer></app-footer>
</div>
```

Так головний контент постійно буде змінюватися, а компоненти інтерфейсу для навігації та компонент footer будуть лишатися на місці. Щоб перевести користувача на інший компонент потрібно скористатися модулем Route за прикладом: `this.router.navigate(['/auth'])`. Для додання посилання на компонент в HTML тезі `<a></a>`, слід скористатися RouterLink за прикладом: `<a [routerLink]="['/profile']"></a>`.

На цьому можна зробити висновок, що налаштування навігації в Angular є доволі простим завданням.

5 ДОСЛІДЖЕННЯ СИСТЕМИ

Розділ присвячений проведенню експериментальної перевірки ефективності розробленої рекомендаційної підсистеми. Дослідження виконано в контексті дисципліни фехтування, зокрема оцінюючи результати моделі на основі п'яти ключових навичок: точність (Accuracy), захист (Defense), атака (Attack), робота ніг (Footwork) та адаптивність (Adaptability). У межах цього розділу передбачається проведення тестування моделі, аналіз отриманих прогнозів щодо

ефективності менторів, а також оцінка точності та практичної користі. Такий підхід дозволяє як кількісно, так і якісно оцінити працездатність системи у реальних умовах застосування.

Система здійснюватиме навчання на даних, представлених у форматі JSON, зразковий вміст яких наведено в лістингу 5.1.

Лістинг 5.1 – Зразок вмісту навчальних даних

```
{ "mentors": [
  {
    "id": "mentor-001", "firstname": "John",
    "lastname": "Doe", "email": "john.doe@example.com",
    "since": "2010-05-12", "totalMentees": 15,
    "costPerMonth": 1200.0,
    "mentoringLocation": "London",
    "mentoring": [
      {
        "desiredSkill": "Accuracy",
        "durationMonths": 3,
        "efficiencyPercent": 4.5,
        "startSkillScore": 60.0, "endSkillScore": 70.0,
        "improvementPoints": 10.0, "improvementPercent": 16.7
      },
      {
        "desiredSkill": "Defense",
        "durationMonths": 3,
        "efficiencyPercent": 4.0,
        "startSkillScore": 55.0, "endSkillScore": 63.0,
        "improvementPoints": 8.0, "improvementPercent": 14.5
      },
      {
        "desiredSkill": "Attack",
        "durationMonths": 3,
        "efficiencyPercent": 4.2,
        "startSkillScore": 58.0, "endSkillScore": 66.0,
        "improvementPoints": 8.0, "improvementPercent": 13.8
      },
      {
        "desiredSkill": "Footwork",
        "durationMonths": 3,
        "efficiencyPercent": 3.9,
        "startSkillScore": 62.0, "endSkillScore": 70.0,
        "improvementPoints": 8.0, "improvementPercent": 12.9
      },
      {
        "desiredSkill": "Adaptability",
        "durationMonths": 3,
        "efficiencyPercent": 4.1,
        "startSkillScore": 60.0, "endSkillScore": 68.0,
        "improvementPoints": 8.0, "improvementPercent": 13.3
      },
      ...
    ]
  }, ...
], ... ] }
```

Після навчання моделей було проведено оцінювання їхньої якості за допомогою метрики RMSE (Root Mean Squared Error) — середньоквадратичної похибки.

Цей показник відображає середнє відхилення прогнозованих значень від фактичних, тобто характеризує точність моделі: чим менше значення RMSE, тим точніше модель відтворює залежності у даних (рис. 5.1).

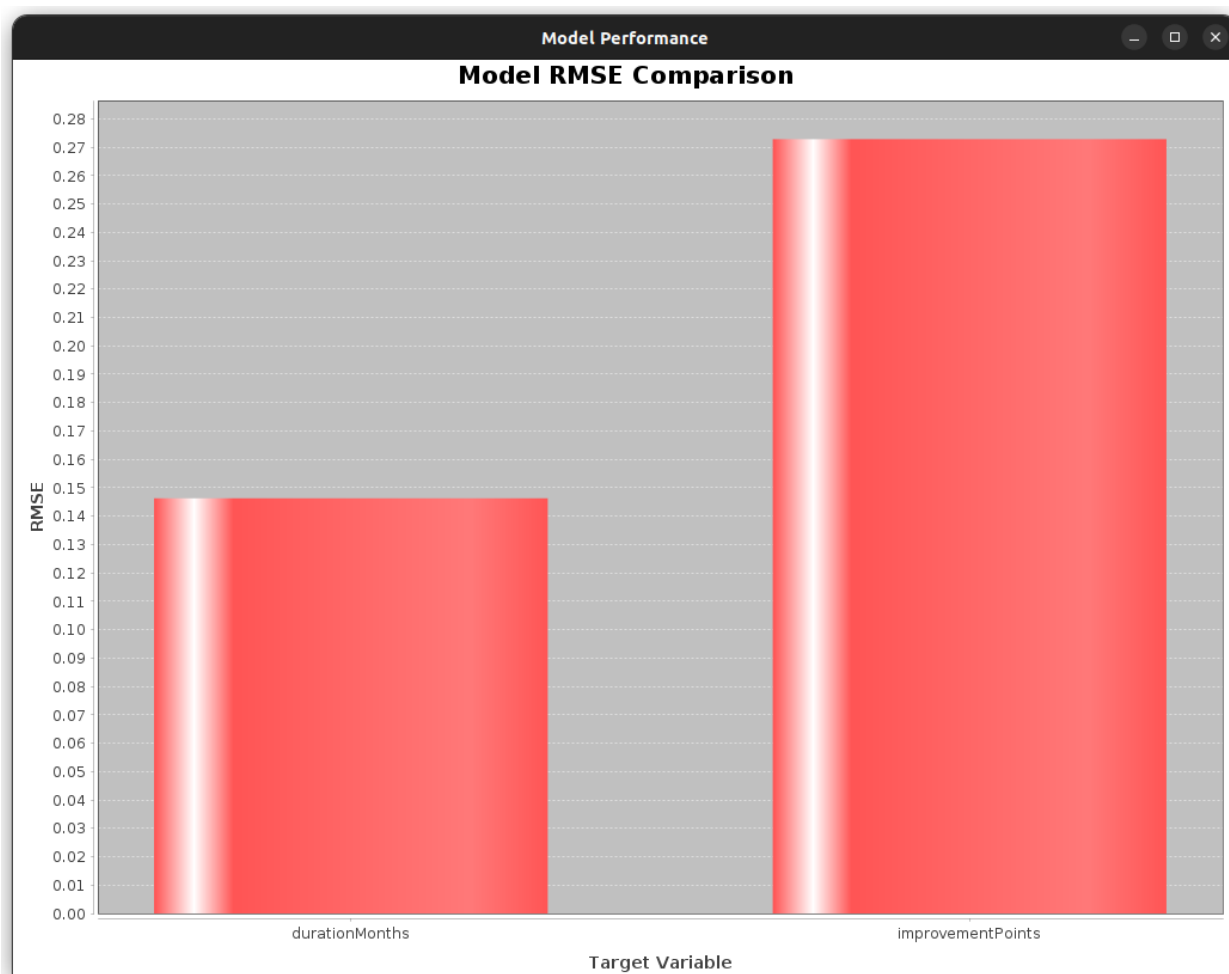


Рисунок 5.1 – Отримані результати RMSE

У результаті навчання отримано такі показники: для моделі прогнозування improvementPoints значення RMSE становить 0.2728, тоді як для моделі durationMonths — 0.1463.

Отримані результати свідчать про достатньо високу точність побудованих моделей, оскільки величини похибки є незначними відносно діапазону цільових

змінних. Це дозволяє зробити висновок, що моделі здатні ефективно узагальнювати навчальні дані та забезпечують адекватну якість прогнозування у контексті рекомендаційної системи для вибору менторів.

Крім того, результати моделювання було представлено у формі рейтингової таблиці менторів, поданої на рисунку 5.2, що забезпечує наочне порівняння їх прогнозованої ефективності.

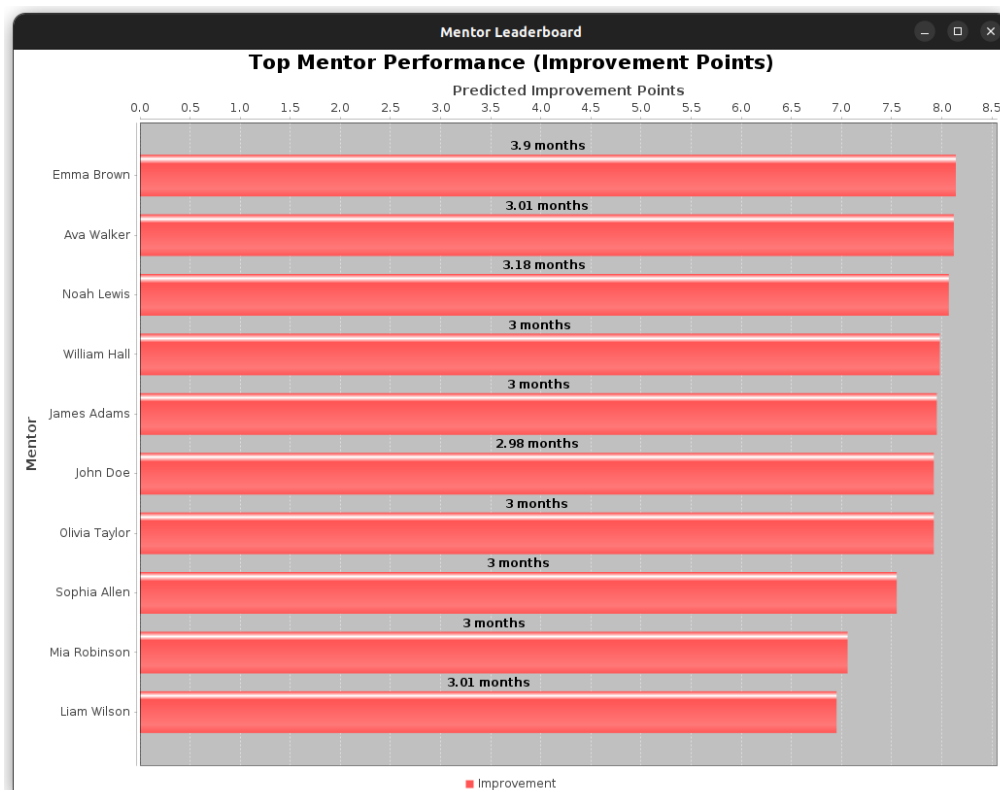


Рисунок 5.2 - Рейтингова таблиця менторів

По осі ординат розташовано імена менторів, а по осі абсцис — значення прогнозованого рівня покращення, вираженого у балах, разом із тривалістю періоду прогнозу у місяцях.

Отримані результати свідчать про те, що розроблена модель успішно виконує поставлене завдання з прогнозування ефективності менторів.

Показники якості моделі, зокрема значення RMSE, підтверджують її здатність формувати точні та стабільні передбачення. Таким чином, система рекомендацій на основі побудованої моделі може бути ефективно використана

для підтримки процесу прийняття рішень щодо підбору оптимального ментора для розвитку певних навичок менті.

Зокрема процес обрання бажаної навички для покращення зображено на рисунку 5.3.

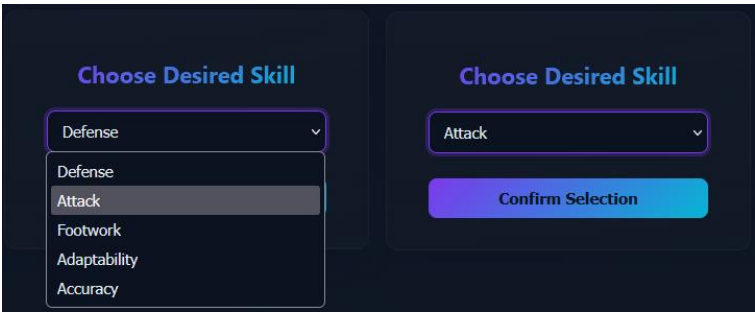


Рисунок 6.3 – Обрання навички для покращення

Після того як менті підтвердить вибір він отримає результат (рис 5.4.).

M Top-10 Mentors By Defense skill.	
1 Anna Petrenko Location: SC "Noire" predicted: 18.4 pts duration: 2.5 mo \$1,200.00 / mo	2 Bohdan Ivanov Location: SC "Dnipro" predicted: 16.0 pts duration: 3.0 mo \$980.50 / mo
3 Clara Novak Location: SC "Hope" predicted: 14.7 pts duration: 2.0 mo \$860.00 / mo	4 Dmytro Kovalenko Location: SC "Spirit" predicted: 13.2 pts duration: 3.5 mo \$720.00 / mo
5 Elena Marchenko Location: SC "Dnipro" predicted: 12.9 pts duration: 4.0 mo \$690.00 / mo	6 Fedor Sokolov Location: SC "Spirit" predicted: 12.1 pts duration: 2.8 mo \$640.00 / mo
7 Halyna Moroz Location: SC "Step" predicted: 11.6 pts duration: 3.2 mo \$610.30 / mo	8 Ihor Levchenko Location: SC "Noire" predicted: 10.9 pts duration: 2.4 mo \$580.00 / mo
9 Julia Bondar Location: SC "Hope" predicted: 9.8 pts duration: 3.6 mo \$520.00 / mo	10 Konstantin Yavorski Location: SC "Spirit" predicted: 8.5 pts duration: 4.2 mo \$470.00 / mo

Рисунок 5.4 – Отримані рекомендації щодо ментора

Менті має можливість натиснути на ім'я ментора, який його зацікавив, та отримати його профіль (рис.5.5).

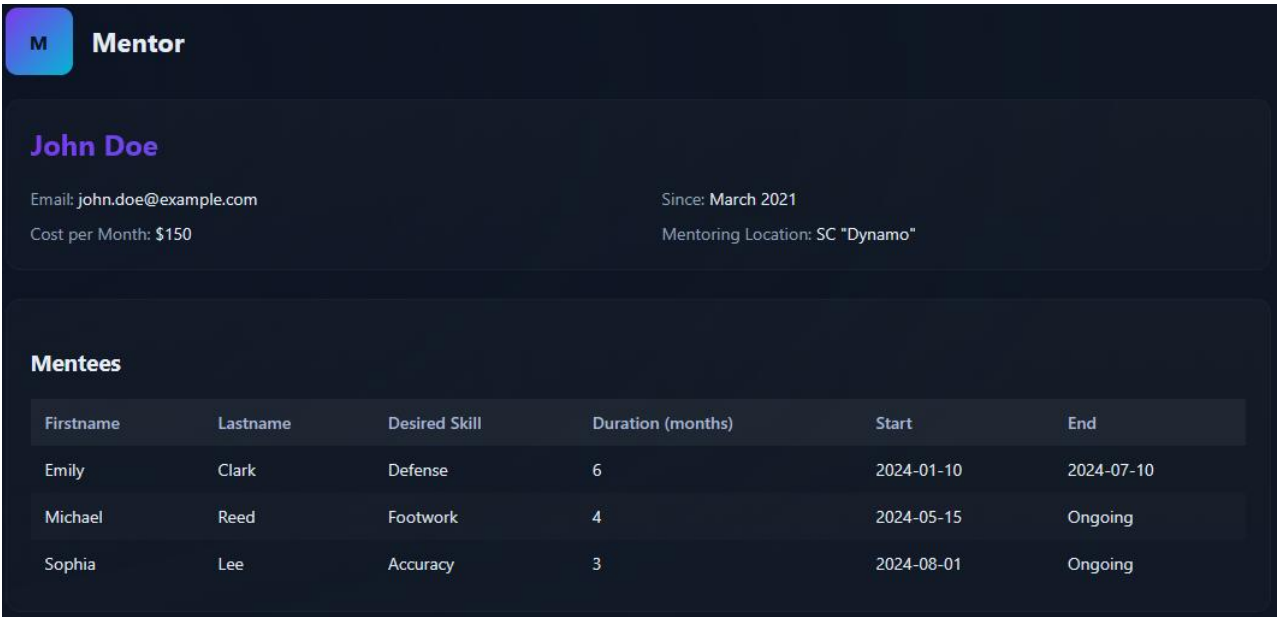


Рисунок 5.5 – Профіль ментора

Цей функціонал забезпечує користувачеві можливість більш детально ознайомитися з досвідом, кваліфікацією та професійною спеціалізацією конкретного ментора, а також отримати уявлення про його стиль роботи через перегляд відгуків інших менті. Така інформаційна підтримка допомагає сформувати більш об’єктивне бачення щодо відповідності ментора потребам користувача. Завдяки зручному та інтуїтивно зрозумілому інтерфейсу можна легко перейти до повного профілю фахівця, переглянути доступні форми взаємодії та за потреби одразу виконати наступні кроки — наприклад, встановити контакт, поставити уточнювальні запитання або записатися на консультацію.

На основі роботи системи можна зробити висновок, що реалізований клієнтський застосунок забезпечує динамічне отримання та коректне відображення рекомендацій менторів відповідно до параметрів, які задає користувач у запиті. Це свідчить про ефективну інтеграцію з серверною частиною, стабільність інформаційного обміну та належну організацію користувацького інтерфейсу.

Крім того, система надасть можливість переглянути історію покращення навичок менті за допомогою графіку в його профілі (рис. 5.6).



Рисунок 5.6 – Графік покращення навичок ментні

Зображений графік дозволяє візуалізувати та відстежувати динаміку розвитку навичок у часі. Це допомагає менті побачити свій прогрес та зрозуміти, які області потребують додаткової уваги. Таким чином, система не тільки підтримує мотивацію менті, але й служить інструментом для досягнення конкретних цілей у процесі навчання.

ВИСНОВКИ

У результаті виконання роботи було розроблено інтерактивну платформу рекомендацій, що ґрунтується на використанні методів статистичного аналізу для рекомендації менторів. У ході дослідження було здійснено повний цикл побудови аналітичної моделі — від підготовки та попередньої обробки даних до навчання та оцінювання точності отриманих результатів. Зокрема, сформовано навчальну вибірку, проведено очищення та нормалізацію даних, побудовано моделі лінійної регресії для прогнозування ключових показників, а також оцінено якість моделювання за допомогою метрики RMSE.

Наукова новизна роботи полягає в тому, що запропоновано підхід, який забезпечує можливість автоматизованого відбору менторів відповідно до конкретних навичок і прогнозних показників їхньої ефективності. Це дозволяє мінімізувати вплив людського фактору під час прийняття рішень, який у більшості існуючих систем базується на рейтингах, популярності або суб'єктивних відгуках користувачів. Використання прогнозних характеристик дає змогу формувати більш точну оцінку потенційного результату менторингу ще до його початку, що суттєво підвищує оптимальність процесу підбору наставників.

Практична цінність результатів роботи полягає в тому, що розроблено мікросервісну систему, яка дає змогу здійснювати підбір наставників на основі визначених критеріїв та навичок, що потребують розвитку; акумулювати статистику як про діяльність менторів, так і про прогрес учня; забезпечити об'єктивність процесу навчання завдяки можливості відстеження динаміки змін у рівні компетентностей учня. Побудована архітектура відзначається гнучкістю, масштабованістю та придатністю до подальшого розвитку. Отримані результати створюють підґрунтя для подальших досліджень і вдосконалення системи.

Таким чином, мета роботи, що полягала у поліпшенні якості автоматизованого процесу щодо підбору наставників за необхідними навичками на основі результативності їх роботи, була досягнута у повному обсязі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Crisp, G., & Cruz, I. (2009). Mentoring college students: A critical review of the literature between 1990 and 2007. *Research in Higher Education*, 50(6), 525–545 p.;
2. Zachary, L. J. *The Mentor's Guide: Facilitating Effective Learning Relationships*. Jossey-Bass, 2011. 112 p.;
3. Clutterbuck, D., & Megginson, D. *Mentoring Executives and Directors*. Butterworth-Heinemann. 2004. 53 p.;
4. Ricci, F., Rokach, L., & Shapira, B. (2015). *Recommender Systems Handbook*. Springer, 29p.;
5. GDPR (General Data Protection Regulation). Regulation (EU) 2016/679 of the European Parliament, 123p.;
6. PostgreSQL Global Development Group. *PostgreSQL: The World's Most Advanced Open Source Relational Database*. URL: <https://www.postgresql.org/> (дата звернення: 01.12.2025);
7. Stonebraker, M. et al. *The Design of Postgres*. ACM, 1986, 211p.;
8. Vance, A. *PostgreSQL vs. MySQL: A Comparative Review*. *Database Journal*, 2021, 35p.;
9. Redis.io. *Redis Documentation*. URL: <https://redis.io/> (дата звернення: 01.12.2025);
10. Sanfilippo, S. *Redis: In-Memory Data Structure Store*. GitHub, 2009, 92p.;
11. Li, J. *Redis vs. Memcached: A Detailed Comparison*. *Database Trends and Applications*, 2022, 67p.;
12. MongoDB Inc. *MongoDB Documentation*. URL: <https://www.mongodb.com/> (дата звернення: 01.12.2025);
- 13 Chodorow, K. *MongoDB: The Definitive Guide*. O'Reilly Media, 2019, 241p.;
14. Lakshman, A., Malik, P. *Cassandra: A Decentralized Structured Storage System*. *ACM SIGOPS*, 2010, 152p.;

15. Oracle. Java Documentation. URL: <https://docs.oracle.com/javase/> (дата звернення: 01.12.2025);
16. Horstmann, C. Core Java, Volume I–II. Pearson, 2022, 116p.;
17. Albahari, J., Albahari, B. C# 10 in a Nutshell. O'Reilly Media, 2022, 211p.;
18. Spring Framework Documentation. Spring By VMware Tanzu. URL: <https://spring.io/projects/spring-framework> (дата звернення: 01.12.2025);
19. Weka official website – University of Waikato. URL: <https://www.cs.waikato.ac.nz/ml/weka/> (дата звернення: 01.12.2025);
20. Angular. Angular Documentation. URL: <https://angular.io/> (дата звернення: 01.12.2025);
21. Freeman, A., Sanderson, N. Angular Up & Running: Learning Angular, Step by Step. O'Reilly Media, 2020, 110p.;
22. Banks, A., Porcello, E. Learning React: Functional Web Development with React and Redux. O'Reilly Media, 2020, 46p.;
23. Siemens, G., & Baker, R. S. J. d. (2012). Learning analytics and educational data mining: Towards communication and collaboration. In Proceedings of the 2nd International Conference on Learning Analytics and Knowledge (pp. 252–254). ACM;
24. О.С Бунке, Серверні Web-Технології: Київ КПІ ім. Ігоря Сікорського 2023. 109с.;
25. Capital One Financial Corporation. URL: <https://www.capitalone.com/tech/software-engineering/microservices-design-patterns/> (дата звернення: 01.12.2025).

ДОДАТОК А

ШАБЛОН ІНТЕРФЕЙСУ КОМПОНЕНТА ПРОФІЛЮ МЕНТНІ

Лістинг А1 – Конопент профілю менті

```

<section class="profile-card">
  <h2>{{ menteeName }}</h2>
  <p><strong>Status:</strong> {{ status }}</p>

  <div class="mentors">
    <h3>Mentors</h3>
    <table>
      <thead>
        <tr>
          <th>Firstname</th>
          <th>Lastname</th>
          <th>Desired Skill</th>
          <th>Since</th>
        </tr>
      </thead>
      <tbody>
        <tr *ngFor="let mentor of mentors">
          <td>{{ mentor.firstname }}</td>
          <td>{{ mentor.lastname }}</td>
          <td>{{ mentor.desiredSkill }}</td>
          <td>{{ mentor.since }}</td>
        </tr>
      </tbody>
    </table>
  </div>
</section>

<section class="chart-card">
  <div class="year-selector">
    <label for="year">Year:</label>
    <select id="year" [(ngModel)]="selectedYear">
      <option *ngFor="let year of years" [value]="year">{{ year
    }}</option>
    </select>
  </div>
  <canvas id="skillsChart"></canvas>
</section>

```

ДОДАТОК Б

СТИЛІ КОМПОНЕНТА ПРОФІЛЮ МЕНТІ

Лістинг Б1 – Стилi компонента профiлю ментi

```
.profile-card, .chart-card {
  background: linear-gradient(180deg, rgba(255,255,255,0.02),
  rgba(255,255,255,0.01));
  border: 1px solid rgba(255,255,255,0.03);
  padding: 20px;
  border-radius: 12px;
  margin-bottom: 24px;
}

.profile-card h2 {
  margin-top: 0;
}

.mentors {
  margin-top: 16px;
}

table {
  width: 100%;
  border-collapse: collapse;
  font-size: 14px;
  color: #e6eef8;
}

th, td {
  padding: 10px 12px;
  text-align: left;
}

th {
  background: rgba(255,255,255,0.05);
  font-weight: 600;
  color: #a9b8d6;
}

tr:nth-child(even) {
  background: rgba(255,255,255,0.02);
}

tr:hover {
  background: rgba(255,255,255,0.05);
}

.canvas-container {
  width: 100%;
  height: 360px;
}
```

```
.chart-card canvas {
  width: 100%;
  height: 360px;
}

.year-selector {
  margin-bottom: 12px;
  display: flex;
  align-items: center;
  gap: 8px;
}

select {
  background: #0b1220;
  border: 1px solid rgba(255,255,255,0.08);
  color: #e6eef8;
  border-radius: 8px;
  padding: 6px 10px;
  font-size: 14px;
}
```

ДОДАТОК В

СЛАЙДИ ПРЕЗЕНТАЦІЇ

Мета: поліпшити якість автоматизованого процесу щодо підбору наставників за необхідними навичками на основі результативності їх роботи.

Задача: розробка та обґрунтування комплексної методики для формування рекомендацій.

Основні етапи роботи:

1. Концептуальний аналіз
2. Моделювання та проектування системи
3. Опис та обґрунтування підходів оцінки та рекомендацій
4. Визначення **архітектури** системи - апаратна і програмна складові
5. Програмна розробка системи
6. Дослідження системи

Рисунок В.1 – Слайд з метою та задачею

Актуальність задачі

Однією з **ключових проблем** сучасного процесу менторства є **відсутність об'єктивного механізму зіставлення потреб учня (менті) та компетентностей ментора**. У більшості випадків вибір наставника здійснюється випадково, на основі рекомендацій знайомих, суб'єктивних відгуків або ж загальної популярності спеціаліста. Це призводить до **низки негативних наслідків**: ментор може не мати достатніх компетенцій у конкретному навичку; відсутність системного аналізу результатів менті ускладнює оцінку ефективності наставництва.

Вирішення цих проблем потребує **створення інтерактивної інформаційної системи**, яка дозволить:

- здійснювати **підбір наставників** на основі **визначених критеріїв** та навичок;
- **акумулювати статистику** як про діяльність менторів, так і про прогрес менті;
- забезпечити можливості **відстеження динаміки змін** у рівні компетентностей учня.

Рисунок В.2 – Слайд з актуальністю задачі

Постановка задачі

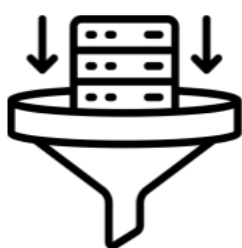
Розроблювальна система є **рекомендаційною онлайн-платформою**, яка підтримує функціонал, як для **внутрішніх** сторін (адміністраторів, модераторів) так і для **зовнішніх** осіб (користувачів).

Система надасть змогу **полегшити процес** підбору наставників відповідно до потреб користувачів, **підвищити ефективність** навчання та зробити процес наставництва **більш прозорим**. Для цього під час розробки застосунку слід вирішити такі задачі:

- надати можливість користувачеві **обирати навичку**, яку він прагне вдосконалити;
- **забезпечити механізм** рекомендацій менторів на основі їхньої статистики;
- **реалізувати збір та аналіз даних** про прогрес менті для відстеження динаміки розвитку навички у часі;
- **забезпечити зручний та інтуїтивний інтерфейс** для пошуку, перегляду профілів менторів та відстеження власних результатів.

Рисунок В.3 – Слайд з постановкою задачі

Оцінка обмежень та ризиків



Якість вхідних даних



Правові ризики пов'язані зі збором даних



Технічні ресурси

Рисунок В.4 – Слайд з оцінкою обмежень та ризиків

Підхід оцінки та рекомендацій

$$MetricScore = 100 * \frac{EntrantValue - L}{U - L}$$
$$MetricScore = 100 * \frac{U - EntrantValue}{U - L}$$
$$SkillScore = \sum (MetricScore * Weight)$$
$$Improvement = \frac{SkillScoreEnd - SkillScoreStart}{50} * 100$$
$$Efficiency = \frac{SkillScoreEnd - SkillScoreStart / AmountOfMonths}{50} * 100$$

де: *EntrantValue* – значення метрики конкретного учасника;
L – нижня межа (5-й процентиль, що відсікає найгірші випадки);
U – верхня межа (95-й процентиль, що відсікає найкращі випадки);
100 – коефіцієнт шкали для приведення результатів у діапазон 0–100.

де: *SkillScoreStart* – результат на початку періоду;
SkillScoreEnd – результат наприкінці періоду;
50 – нормувальний коефіцієнт, що забезпечує інтерпретацію приросту у відсотках.

де *Weight* – вага навички.

Рисунок В.5 – Слайд з підходами для оцінки та рекомендацій

Функціональні можливості користувача



Рисунок В.6 – Рисунок з функціональними можливостями користувачів



Рисунок В.7 – Слайд функціональними можливостями користувачів з привілеями

Серверні технології



Мова програмування: [Java 17](#)

Фреймворк: [Spring Boot 3](#)

Допоміжні бібліотеки: [Weka](#)

Системи керування базами даних:
[PostgreSQL](#), [MongoDB](#), [Redis](#)

Рисунок В.8 – Слайд з серверними технологіями

Клієнтські технології



Мова програмування: [TypeScript](#)

Фреймворк: [Angular 17](#)

Допоміжні: [Angular Material](#)

Рисунок В.9 – Слайд з клієнтськими технологіями

Зовнішня архітектура

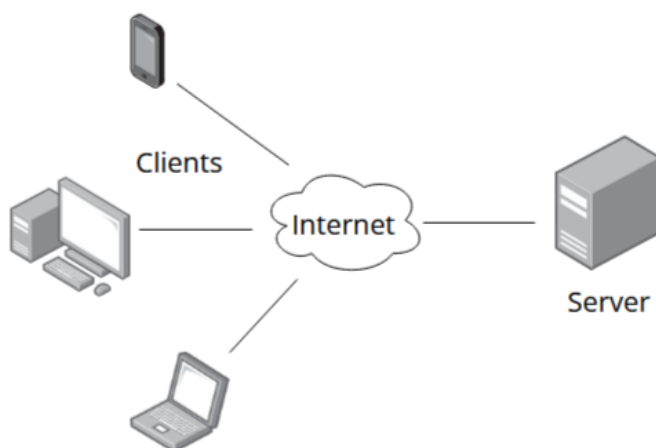


Рисунок В.10 – Слайд з зовнішньою архітектурою

Внутрішня архітектура

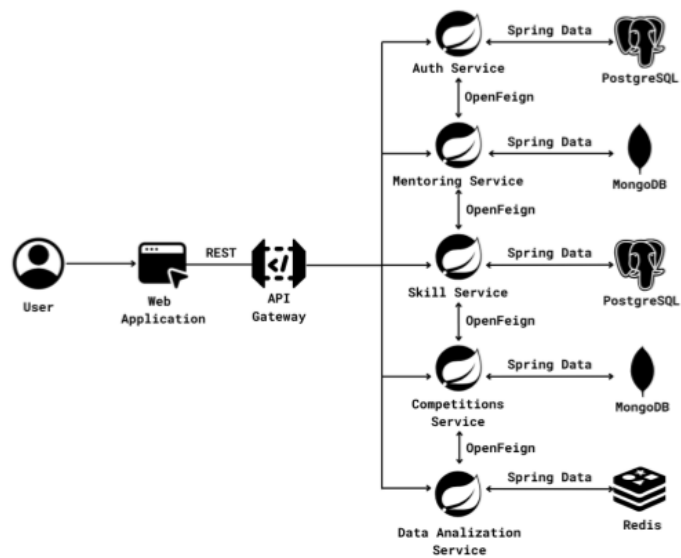


Рисунок В.11 – Слайд з внутрішньою архітектурою

Апаратне розгортання архітектури

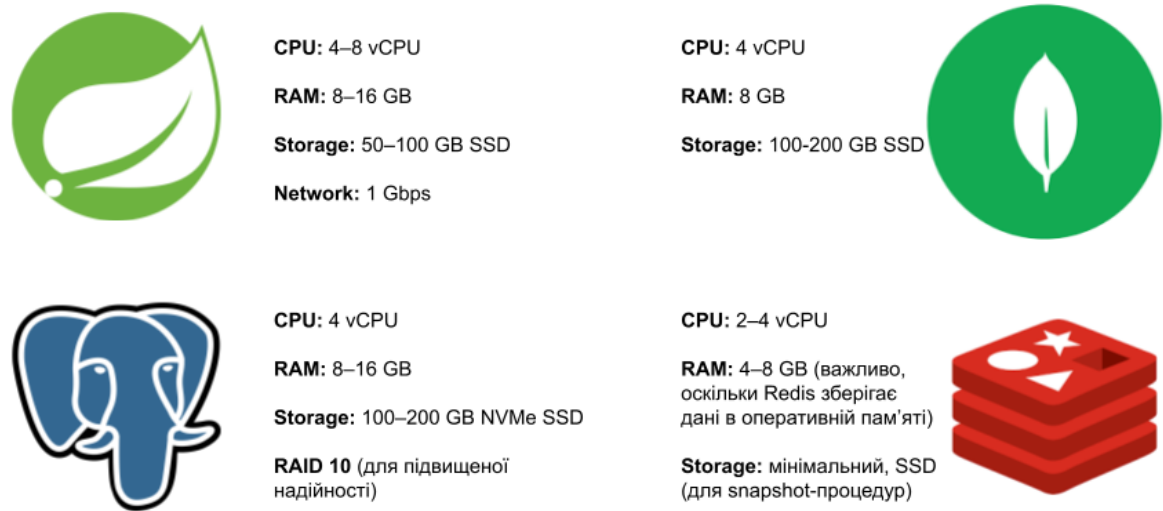


Рисунок В.12 – Слайд з апаратним розгортанням архітектури

UML діаграма розгортання ресурсів

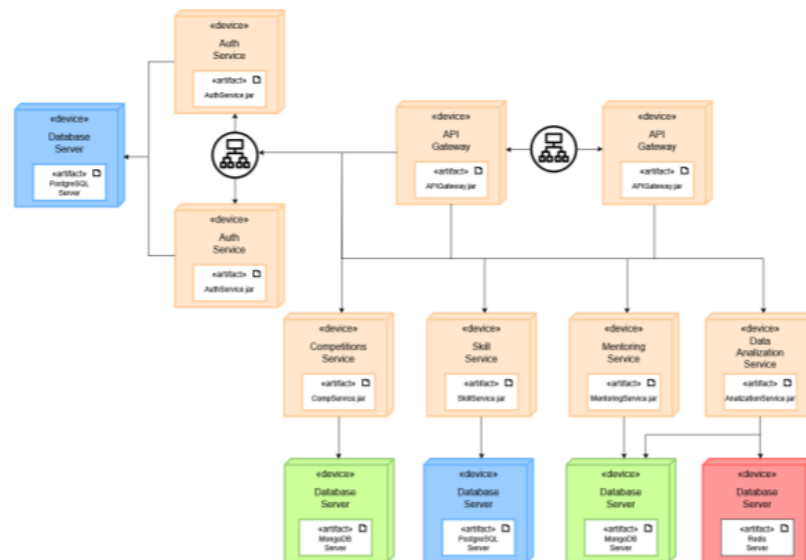


Рисунок В.13 – Слайд з діаграмою розгортання ресурсів

Діаграма послідовностей надання рекомендацій

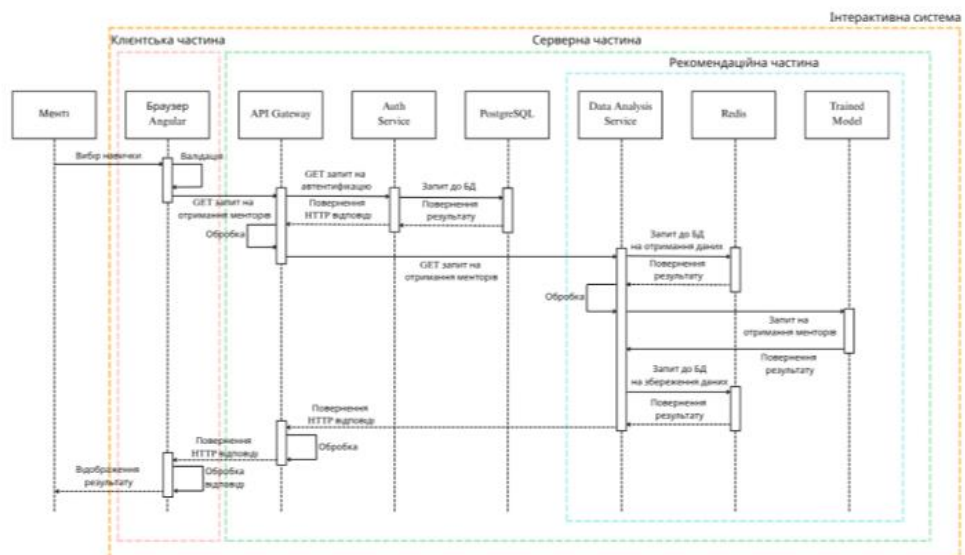


Рисунок В.14 – Слайд діаграмою послідовностей надання рекомендацій

Діаграма послідовностей збору даних

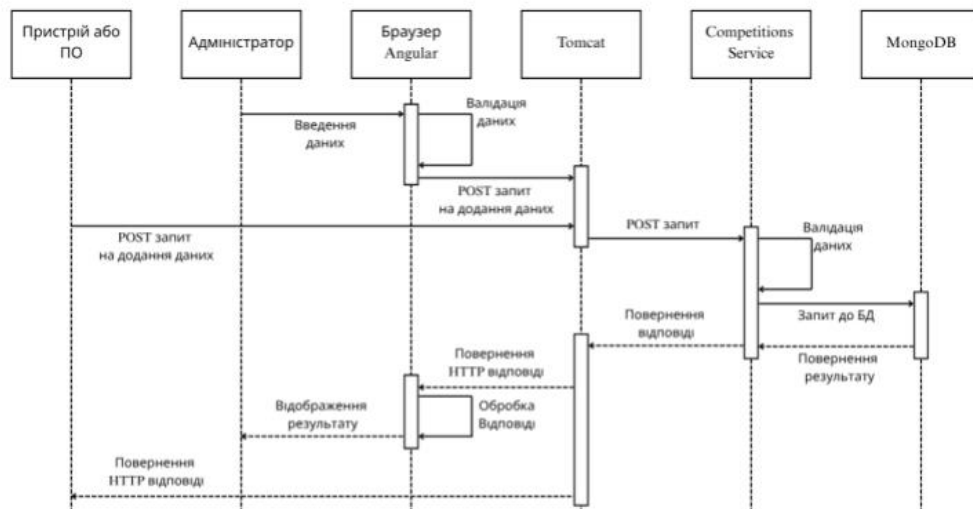


Рисунок В.15 – Слайд з діаграмою послідовностей збору даних

Результати тестування

Для моделі прогнозування improvementPoints значення **середньоквадратичної похибки** становить 0.2728, тоді як для моделі durationMonths — 0.1463, що свідчать про достатньо **високу точність** побудованих моделей.

Це дозволяє зробити **висновок**, що моделі здатні ефективно узагальнювати навчальні дані та **забезпечують адекватну якість прогнозування** у контексті рекомендаційної системи для вибору менторів.

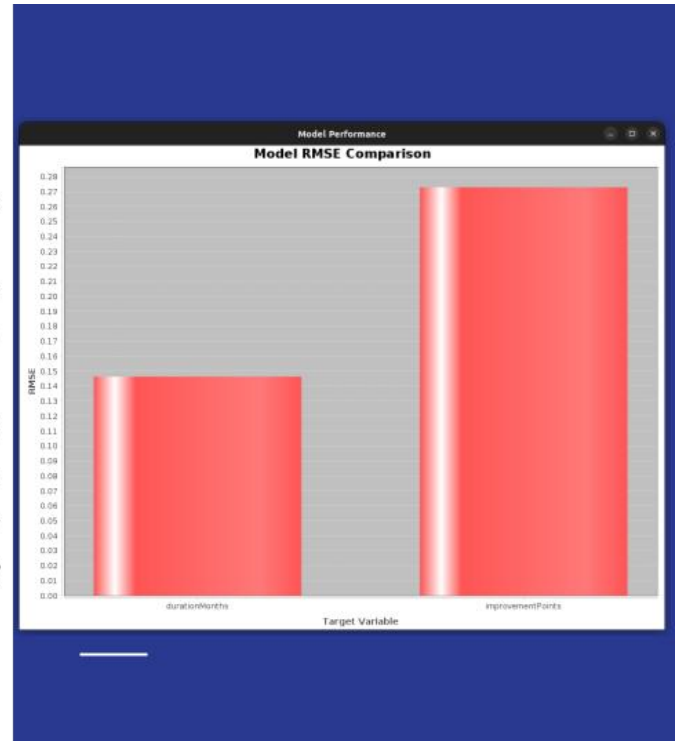


Рисунок В.16 – Слайд з результатами тестування

Висновок

Наукова новизна роботи полягає в тому, що запропоновано підхід, який забезпечує можливість автоматизованого відбору менторів відповідно до конкретних навичок і прогнозних показників їхньої ефективності. Це дозволяє мінімізувати вплив людського фактору під час прийняття рішень, який у більшості існуючих систем базується на рейтингах, популярності або суб'єктивних відгуках користувачів. Використання прогнозних характеристик дає змогу формувати більш точну оцінку потенційного результату менторингу ще до його початку, що суттєво підвищує оптимальність процесу підбору наставників.

Практична цінність результатів роботи полягає в тому, що розроблено мікросервісну систему, яка дає змогу здійснювати підбір наставників на основі визначених критеріїв та навичок, що потребують розвитку; акумулювати статистику як про діяльність менторів, так і про прогрес учня; забезпечити об'єктивність процесу навчання завдяки можливості відстеження динаміки змін у рівні компетентностей учня.

Рисунок В.19 – Слайд з висновками