МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Система підтримки прийняття рішень обрання дисципліни за вибором заснована на колаборативній фільтрації»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Кузіков Б.О.**

Студент групи ІН–61 **Міщенко А.Є.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую _____

Зав. кафедрою Довбиш А.С.

«_____» _____2020 р.

Завдання

до випускної роботи

Студента четвертого курсу, групи ІН-61 спеціальності «Інформатика» денної форми навчання Міщенка Антона Єгоровича.

Тема: «Система підтримки прийняття рішень обрання дисципліни за вибором заснована на колаборативній фільтрації»

Затверджена наказом по СумДУ

№_____ від _____2020 р.

Зміст пояснювальної записки: 1) огляд підходів до реалізації рекомендаційних систем; 2) постановка; 3) проектування системи підтримки прийняття рішень; 4) реалізація системи підтримки прийняття рішень та її тестування; 5) висновки.

Дата видачі завдання «_____»_____2020 г.

Керівник випускної роботи _____ Кузіков Б.О.

Завдання прийняв до виконання _____ Міщенко А.Є.

РЕФЕРАТ

Записка: 61 стор., 9 рис., 9 табл., 2 додатки, 31 джерело.

Об'єкт дослідження — Система підтримки прийняття рішень обрання дисципліни за вибором заснована на колаборативній фільтрації.

Мета роботи — розробити систему підтримки прийняття рішень обрання дисципліни за вибором. Система повинна приймати на вхід оцінки студента з його навчальних дисциплін та у відповідь надавати рекомендацію про дисципліну, яка є для нього оптимальним вибором.

Результати — проведено дослідження літератури з теми «Рекомендаційні системи». Розроблено алгоритм роботи системи підтримки прийняття рішень. Зроблено вибір інструментів реалізації розв'язання поставленої задачі. Реалізовано систему підтримки прийняття рішень мовою Java.

СИСТЕМА ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ, РЕКОМЕНДАЦІЙНА СИСТЕМА, КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ, КОСИНУС ПОДІБНОСТІ, СЕРЕДНЄ ЗВАЖЕНЕ ЗНАЧЕННЯ, JAVA, SQLITE, JAVAFX, MAVEN

ЗМІСТ

ВСТУП

Кожного року перед студентами постає питання про те, яку з-поміж десятків дисциплін за вибором обрати для вивчення у наступному семестрі. У результаті, найчастіше вибір робиться на користь тієї дисципліни, яку обирають друзі. При цьому студент не задумується, чи сподобається вона йому і чи буде корисною. У даній роботі буде зроблено спробу розв'язати цю проблему та допомогти зробити правильний вибір з допомогою системи підтримки прийняття рішень.

Система підтримки прийняття рішень (СППР) – це інтерактивна автоматизована система, що допомагає особам, які приймають рішення, використовувати дані й моделі для розв’язання неструктурованих і слабоструктурованих проблем [31]. Вона аналізує велику кількість даних, збираючи інформацію, яка може бути використана для розв'язання проблем та прийняття рішень.

СППР може бути повністю автоматизованою або отримувати інформацію від людини. У деяких випадках може бути поєднано обидва підходи. Ідеальні системи аналізують інформацію та фактично приймають рішення за користувача. Як мінімум, вони дозволяють користувачам швидше приймати більш обґрунтовані рішення.

СППР може бути розроблена для будь-якої галузі, професії чи сфери діяльності людини, включаючи медичну сферу, державні установи, сільськогосподарські організації та корпоративні операції. Наприклад, вона може бути використана для прогнозування доходів компанії на основі припущень щодо продажу продукції.

СППР повинна показувати інформацію користувачу легко зрозумілим способом. Наприклад, результат може бути виведений графічно (за допомогою графіка) або як письмовий звіт.

Для досягнення поставленої мети необхідно провести аналіз літератури з теми «Рекомендаційні системи», на основі опрацьованої інформації спроектувати та розробити СППР обрання дисципліни за вибором.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Поняття рекомендаційної системи

Рекомендаційна система – це інтелектуальна система, яка сприяє формуванню пропозицій, на основі аналізу релевантної інформації, отриманої з віртуального сховища даних, що консолідує велику кількість джерел [29]. Вона працює на основі аналізу поведінки користувача у сервісі або знань про предметну область. Основна задача рекомендаційної системи – надати користувачу інформацію про товар чи послугу, яка може бути йому цікава. Виходячи з цього, в основному рекомендаційні системи використовуються у сфері продаж.

Рекомендаційні системи використовуються у різних областях людської діяльності. Кожен з нас стикався з ними у соцмережах (Facebook, Twitter), музичних сервісах (Spotify, Apple Music), онлайн кінотеатрах (Netflix, MEGOGO), інтернет-магазинах (Rozetka.ua, Amazon, eBay), тощо.

Компанії використовують рекомендаційні системи для збільшення своїх прибутків, що досягається покращенням роботи з клієнтами через надання правильних рекомендацій. Рекомендації прискорюють пошук та допомагають користувачам отримувати пропозиції про товари та послуги, які вони, можливо, ніколи б і не побачили самостійно. Тобто, рекомендаційні системи допомагають орієнтуватися у великій кількості контенту (наприклад, за даними 2019 року, на YouTube щохвилинно завантажується 500 годин відео [1]). Також компанії можуть надсилати персоналізовані пропозиції на пошту, і, таким чином, утримувати клієнтську базу. Такі дії спонукають клієнтів частіше користуватися послугами компанії, що робить компанію більш конкурентоспроможною на ринку.

У загальному вигляді, рекомендаційну систему можна описати наступним чином:

Нехай U – множина користувачів, I – множина предметів, R – множина відомих оцінок предметів, де $u_{i,j} \in R$, $u_i \in U$, $j \in I$ – оцінка користувача u_i

предмету *j*. Необхідно розрахувати значення оцінки для пари *(u', j')*, для якої $u' \in U, j' \in I, u_{i',j'} \notin R$.

Рекомендаційні системи розрізняються за наступними характеристиками [30]:

- предмет рекомендації;
- ціль рекомендації;
- контекст рекомендації;
- джерело рекомендації;
- ступінь персоналізації;
- прозорість;
- алгоритм.

Існують різні підходи до побудови рекомендаційних систем. Серед основних можна виділити наступні:

- рекомендаційні системи, засновані на колаборативній фільтрації – працюють на припущенні, що якщо смаки користувачів співпадали раніше, то їх вибір буде однаковим й у майбутньому;
- рекомендаційні системи, засновані на змісті – працюють на порівнянні смаків користувача та певних характеристик предмета;
- рекомендаційні системи, засновані на знаннях – працюють на основі деяких заздалегідь визначених правил та обмежень предметної області рекомендаційної системи та явних вимог користувача;
- демографічні рекомендаційні системи – користувачі діляться на групи, яким і надаються спільні рекомендації;
- рекомендаційні системи, засновані на корисності – для кожного предмета за спеціальною формулою розраховується його корисність для користувача;
- гібридні рекомендаційні системи – являють собою комбінацію декількох різних підходів.

1.2 Огляд підходів до реалізації рекомендаційних систем

1.2.1 Рекомендаційні системи, засновані на колаборативній фільтрації

Рекомендаційні системи, засновані на колаборативній фільтрації роблять прогнози виходячи з попередньої реакції користувача на предмети у системі. Це можуть бути як оцінки за якоюсь шкалою, так і просто інформація про користування певними предметами. При цьому, рекомендація надається з урахуванням поведінки інших користувачів системи.

Основа колаборативної фільтрації – матриця, де по одній з осей розташовані користувачі, а по іншій – предмети. У комірках матриці міститься інформація про те, як той чи інший користувач оцінив певний предмет. При цьому, відсутність значення означає, що користувач до цього моменту ніяк не взаємодіяв з предметом. Ціллю роботи системи є заповнення цих пустих комірок.

Існує 2 підходи до побудови таких систем:

- на основі користувачів;
- на основі предметів.

Наведемо принцип їх роботи.

Колаборативна фільтрація на основі користувачів працює наступним чином: серед усіх користувачів необхідно виділити *n* найбільш схожих на ключового та на основі їх оцінок зробити прогноз. Проілюструємо цей підхід таблицею 1.1.

Таблиця 1.1 Приклад колаборативної фільтрації на основі користувачів

	Предмет 1	Предмет 2	Предмет 3	Предмет 4
Користувач 1	**3**	**3**	**5**	3
Користувач 2	1	5	2	5
Користувач 3			2	
Користувач 4	**3**	**?**	**5**	2

У таблиці 1.1 маємо матрицю оцінок між 4 користувачами та 4 предметами. При цьому необхідно зробити прогноз оцінки Користувача 1 Предмету 2. З оцінок видно, що на Користувача 4 найбільше схожий Користувач 1, тому можна зробити прогноз, що шукане значення приблизно дорівнює 3.

Колаборативна фільтрація на основі предметів працює наступним чином: серед усіх предметів необхідно виділити *n* найбільш схожих на ключовий та на основі їх оцінок зробити прогноз. Проілюструємо цей підхід таблицею 1.2

Таблиця 1.2 Приклад колаборативної фільтрації на основі предметів

	Предмет 1	Предмет 2	Предмет 3	Предмет 4
Користувач 1	3	**3**	5	**3**
Користувач 2	1	**5**	2	**5**
Користувач 3			2	
Користувач 4	3	**?**	5	**2**

З таблиці 1.2 бачимо, що на Предмет 2 найбільше схожий Предмет 4, тому можна зробити прогноз, що шукане значення приблизно дорівнює 2.

Тепер постає питання, яким чином розрахувати схожість між користувачами або предметами, а потім зробити передбачення? Для розв'язання цієї задачі існує 2 групи алгоритмів:

- алгоритми засновані на пам'яті;
- алгоритми засновані на моделі.

Алгоритми засновані на пам'яті працюють шляхом розрахунку деякої міри на основі тих даних, які наявні у системі (матриці оцінок). Оцінки кожного користувача (або оцінки кожного предмета) можна представити у вигляді векторів, до яких можна застосувати міри подібності. Після того, як буде вирахувано міру з її використанням можна буде розрахувати передбачення. Серед найбільш застосованих мір виділяють косинус

подібності [3, 4] (формула (1.1))– являє собою кут між векторами оцінок, критерій кореляції Пірсона [3, 4] (формула (1.2)), скоригований косинус подібності [3, 4] (формула (1.3)), коефіцієнт Жаккара [3] (формула (1.4)):

$$cosineSimilarity(u_i, u_j) = \cos(u_i, u_j) = \frac{\sum_{k=1}^{n} u_{i,k} u_{j,k}}{\sqrt{\sum_{k=1}^{n} u_{i,k}^2} \sqrt{\sum_{k=1}^{n} u_{j,k}^2}}, \quad (1.1)$$

$$PearsonSimilarity(u_i, u_j) = \frac{\sum_{k=1}^{n} (u_{i,k_k} - \overline{u_i})(u_{j,k_k} - \overline{u_j})}{\sqrt{\sum_{k=1}^{n} (u_{i,k} - \overline{u_i})^2} \sqrt{\sum_{k=1}^{n} (u_{j,k} - \overline{u_i})^2}}, \quad (1.2)$$

$$adjustedCosineSimilarity(u_i, u_j) = \frac{\sum_{k=1}^{n} (u_{i_k} - \overline{u_k})(u_{j_k} - \overline{u_k})}{\sqrt{\sum_{k=1}^{n} (u_{i,k} - \overline{u_k})^2} \sqrt{\sum_{k=1}^{n} (u_{j,k} - \overline{u_k})^2}}, \quad (1.3)$$

$$JaccardSimilarity(u_i, u_j) = \frac{|u_i \cap u_j|}{|u_i \cup u_j|}, \quad (1.4)$$

де u_i, u_j – вектори оцінок користувачів *i* та *j*;

n – кількість предметів;

$\overline{u_i}$, $\overline{u_j}$ – середні оцінки користувачів *i* та *j*;

$\overline{u_k}$ – середня оцінка предмета *k*.

Після цього, на основі отриманої міри подібності необхідно розрахувати прогнозоване значення оцінки. Для цього необхідно спочатку з усієї множини користувачів обрати найбільш схожих на даного. Це можна зробити, наприклад, взявши верхніх *n* користувачів за схожістю, або відфільтрувавши користувачів за якимось пороговим значенням схожості. Для прогнозу оцінки використовуються агрегуючі функції, до найчастіше уживаних належать середнє значення (формула (1.5)), середнє зважене [2] (формула (1.6)), відносне середнє зважене [2] (формула (1.7)).

$$u_{i,k} = \frac{\sum_{u_j \in U'} u_{j,k}}{|U'|}, \quad (1.5)$$

$$u_{i,k} = \frac{\sum_{u_j \in U'} similarity(u_i, u_j) u_{j,k}}{\sum_{u_j \in U'} |similarity(u_i, u_j)|}, \quad (1.6)$$

$$u_{i,k} = \overline{u_i} + \frac{\sum_{u_j \in U'} similarity(u_i, u_j)(u_{j,k} - \overline{u_{j,k}})}{\sum_{u_j \in U'} |similarity(u_i, u_j)|}, \quad (1.7)$$

де $u_{i,k}$- оцінка користувача i предмету k;

U' – множина користувачів, які найбільше схожі на даного;

u_j – вектор оцінок користувача з U;

$\overline{u_i}$ – середня оцінка користувача i серед оцінених ним предметів.

Отримане у результаті значення $u_{i,k}$ і є шуканим прогнозом.

Також до алгоритмів колаборативної фільтрації заснованих на пам'яті належать алгоритми сімейства Slope One. Вони є простими та ефективними [5]. У основі методу лежить предиктор, який являє собою середню різницю оцінок двох предметів [6, 7] (формула (1.8)). Після цього, предиктор застосовується для прогнозування оцінки. Очікуване значення оцінки розраховується за формулою [7] (1.9). Для більш точного прогнозування також може застосовуватися середньо зважене значення [7] (формула (1.10)).

$$dev_{k,m} = \frac{\sum_{u_i \in U_{k,m}} (u_{j,k} - u_{j,m})}{|U_{k,m}|}, \quad (1.8)$$

$$u_{i,k} = \frac{\sum_{m \in S_i} (dev_{k,m} + u_{i,m})}{|S_i|}, \quad (1.9)$$

$$u_{i,k} = \frac{\sum_{m \in u_i} (dev_{k,m} + u_{i,m}) |U_{k,m}|}{\sum_{m \in u_i} |U_{k,m}|}, \quad (1.10)$$

де $dev_{k,m}$ – предиктор для предмета k на основі предмета m;

$U_{k,m}$ – множина користувачів, які оцінили як предмет k, так і предмет m;

u_i – оцінки користувача i;

S_i – множина предметів, які були оцінені користувачем i та були оцінені будь-яким іншим користувачем з множини спільно з предметом k.

У колаборативній фільтрації заснованій на моделі застосовуються різні алгоритми добування даних та машинного навчання для прогнозування оцінок

користувачів. Їх можна розбити на 3 групи: непараметричні алгоритми (наприклад, метод k-найбличжих сусідів), алгоритми факторизації матриць (наприклад, SVD) та методи глибинного навчання (наприклад, багатошарові нейромережі) [8].

Факторизація матриць належить до одних з найпоширеніших підходів побудови рекомендаційних систем заснованих на моделі. Факторизація – це операція розкладу об'єкта на його прості складові [9]. Спочатку необхідно виділити певний набір прихованих факторів предметної області. Нехай таких факторів буде *r*. Вони являють собою якісь характеристики предметів. Після цього необхідно за одним з алгоритмів розкласти матрицю оцінок – *X* – у 2 матриці: матрицю відношення користувачів до прихованих факторів (матриця представлення користувачів – *U*), та матрицю відношення предметів до прихованих факторів (матриця представлення предметів – *V*). До найбільш популярних алгоритмів факторизації матриць належать SVD, TruncatedSVD, PCA, NMF. У результаті отримаємо, що $X \approx U * V$ – формула 1.11 [9].

$$\begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix} \approx \begin{pmatrix} u_{11} & \cdots & u_{1r} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mr} \end{pmatrix} \begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{r1} & \cdots & v_{rn} \end{pmatrix} \tag{1.11}$$

Після цього, щоб отримати рекомендації – тобто заповнити пропуски у початковій матриці *X* – необхідно виконати множення *U* матриці на матрицю *V*, у результаті отримаємо матрицю $X' \approx X$, яка міститиме числа замість пропусків. Проте ці числа не будуть являти собою реальний прогноз оцінки – це лише певні значення, які тепер можна використовувати для рекомендацій. Наприклад, можна інтерполювати оцінки предметів у певний діапазон, а потім рекомендувати користувачам ті предмети, де отримане значення вище заданого порогу [9].

1.2.2 Рекомендаційні системи, засновані на змісті

Колаборативні системи засновані на змісті працюють з описом предмета та профілем уподобань користувачів. У таких системах кожен предмет представляється у вигляді набору ознак. Потім ці ознаки порівнюються зі смаками користувача або використовуються для пошуку предметів, схожих на ті, які користувач високо оцінив.

Ознаки предметів залежать від конкретної предметної області, у якій застосовується рекомендаційна система. Наприклад, для фільмів це можуть бути жанр, рік випуску, режисер, для одягу – колір, матеріал, розмір.

Профіль користувача може формуватися двома шляхами: явним (наприклад, анкетування при реєстрації), або неявним (наприклад, у результаті аналізу історії користувача). При цьому необхідно мати спосіб отримання релевантних даних про смаки користувача. Для цього можна використовувати алгоритми добування даних, інтелектуального аналізу тексту, тощо.

Набір ознак предмета та профіль користувача можна представити у вигляді вектора. А за допомогою, наприклад, наївного баєсового класифікатору, кластерного аналізу чи дерева ухвалення рішень ці вектори можна порівнювати та шукати предмети зі схожими ознаками.

Одним з поширених підходів, який застосовується у рекомендаційних системах, заснованих на змісті є TF-IDF. Найчастіше він застосовується при аналізі текстів, наприклад, при рекомендації книг. Розглянемо його детальніше.

Спочатку для кожного значення ознак предметів необхідно розрахувати міру TF (Term Frequency) – частоту входження цього значення до вектора ознак [11]. Для цього можна застосувати декілька підходів, наприклад, бінарний [10] (формула (1.12)), простий підрахунок [10] (формула (1.13)), частоту входження [10] (формула (1.14)), нормалізацію логарифмуванням [10] (формула (1.15)), подвійну нормалізацію K [10] (формула (1.16)):

$$TF(t,d) = 1 \; if \; t \; occurs \; in \; d \; else \; 0, \tag{1.11}$$

$$TF(t,d) = n_t, \tag{1.12}$$

$$TF(t,d) = \frac{n_t}{\sum t'}, \tag{1.13}$$

$$\mathrm{TF(t,d)} = \log(1 + n_t), \tag{1.14}$$

$$TF(t,d) = K + (1-K)\frac{n_t}{\max\{n_{t'} : k \in d\}}, \tag{1.15}$$

де t – значення ознаки,

d – вектор ознак,

n_t – кількість входження ознаки t до вектора d,

K – фактор нормалізації, зазвичай $K = 0.5$.

Потім необхідно розрахувати значення міри IDF (Inverse Document Frequency) – інверсія частоти, з якою слово зустрічається у векторах ознак [11]. Чим рідше слово зустрічається у різних векторах, тим більше інформації воно несе. Через це його цінність збільшується. Це і показує міра IDF. Загальний вигляд розрахунку цієї міри можна побачити на формулі (1.16) [11]. У чисельнику маємо загальну кількість векторів, а у знаменнику – кількість векторів, у яких зустрічається ознака.

$$IDF(t,D) = \log\frac{N}{|\{d \in D : t \in d\}|}. \tag{1.16}$$

Маючи міри TF та IDF для кожного значення ознаки можемо розрахувати міру TF-IDF, яка дорівнює добутку TF та IDF [11]. У результаті отримали вектори ознак предметів зі значенням міри TF-IDF. Тепер можна розрахувати значення TF-IDF для вхідного вектора ознак, для якого робимо прогноз (наприклад, профілю користувача). Отримані вектори можна порівнювати, наприклад, із застосуванням косинусу подібності та робити рекомендації.

1.2.3 Рекомендаційні системи, засновані на знаннях

Рекомендаційні системи, засновані на знаннях працюють на основі знань про предметну область. Як і у рекомендаційних системах заснованих на контенті, кожен предмет представлений у вигляді вектора ознак. Далі, знаючи як ці ознаки співвідносяться з інтересами користувача, можна побудувати базу знань, яка їх описує. На основі цих знань за вибором користувача можна розрахувати рекомендацію для нього. При цьому база знань може бути гнучко налаштована під кожну предметну область. Такі системи мають високу точність, бо вони не оцінюють схожість між предметами чи користувачами для прогнозування, а пропонують користувачу те, що йому потрібно.

Рекомендаційні системи, засновані на знаннях добре підходять для предметних областей, де предмети не купуються регулярно (наприклад, будинки). При цьому оцінки таких предметів зібрати важко, або з часом вони стають неактуальними. У таких випадках користувач може чітко описати свої вимоги, а система – правильно їх обробити. При цьому у користувача є контроль вибору, система підбере йому той предмет, який найбільше підходить до його явних вимог.

Виділяють 2 основних типи рекомендаційних систем заснованих на знаннях: на основі обмежень та на основі конкретних випадків [12].

На основі обмежень – користувач задає вимоги або обмеження до певних ознак предмета [12]. Наприклад, це може бути ціна або рік випуску автомобіля. Також при такому підході, обмеження можуть бути закладені на рівні бази знань системи. Причому вони можуть стосуватися як тільки предметів (наприклад, новобудови не мають ремонту), так і взаємодії ознак користувача та ознак предмета (наприклад, старі інвестори не будуть інвестувати у проекти з високим ризиком [12]). У процесі роботи з системою, користувач може змінювати обмеження, видаляти їх, або додавати нові до тих пір, поки результат його не задовольнить.

На основі конкретних випадків – користувач визначає для чого йому потрібна рекомендація [12]. При цьому до ознак предметів можуть входити й

цілі, для яких він призначений. Система шукає ті предмети, які підходять введеним користувачем цілям та рекомендує їх. Як і у підході на основі обмежень користувач, отримавши рекомендацію, може скоригувати свій запит.

Взаємодія з користувачем у таких системах може відбуватися декількома шляхами [12]:

- розмовний підхід – уподобання користувача отримуються системою у циклі зворотного зв'язку;
- пошуковий підхід – уподобання користувача отримуються системою з відповідей на задану послідовність запитань;
- навігаційний підхід – користувач вказує, що б він хотів змінити у товарі, який рекомендується.

1.2.4 Демографічні рекомендаційні системи

У демографічних рекомендаційних системах рекомендації надаються за належністю користувача до певної групи [13]. Спочатку збирається вся наявна інформація про користувача: наприклад, стать, вік, рівень освіти, тощо. Потім за цими ознаками користувачі діляться на певні групи. Рекомендації розраховуються одразу для цілої групи. Це може бути як нова рекомендація для групи так і рекомендації від одних користувачів іншим всередині груп. При появі нового користувача щоб почати рекомендувати для нього предмети досить лише з'ясувати, до якої групи він відноситься.

Такі системи формуються співвідношення між людьми як і системи колаборативної фільтрації, але використовують для цього різні дані. При цьому вони не потребують наявності оцінок для розділення по групах [13].

1.2.5 Рекомендаційні системи, засновані на корисності

Рекомендаційні системи, засновані на корисності працюють за принципом розрахунку корисності предмета для користувача. Кожен з

предметів задається у вигляді вектора ознак [13]. Для кожного користувача визначається власна функція корисності. Вона може бути задана системою на основі певних правил або ним самим. Ця функція визначає важливість для користувача того чи іншого атрибуту. При цьому, важливість атрибутів може динамічно змінюватись, наприклад, залежно від поточної дати або місця розташування користувача. Після розрахунку значення цієї функції для кожного предмета, можна рекомендувати ті предмети, для яких значення найбільше.

У таких рекомендаційних системах можуть застосовуватись підходи MAUT (Multi-Attribute Utility Theory – теорія корисності мультиатрибутів) [14]. У теорії рішень функція корисності мультиатрибутів використовується для кількісної оцінки відносної привабливості альтернатив з кількома атрибутами [15].

1.2.6 Гібридні рекомендаційні системи

Гібридні рекомендаційні системи являють собою об'єднання двох або більше підходів до побудови рекомендаційних систем. Це дозволяє в рамках однієї системи переваги підходу кожної підсистеми та зменшити (або повністю нівелювати) їх недоліки. При цьому існує багато варіантів побудови гібридних систем, розглянемо найбільш популярні.

Зважена комбінація – кожній з систем призначається ваговий коефіцієнт, які використовуються для прогнозування шляхом розрахунку лінійної комбінації оцінок об'єктів [16]. Також може бути застосована схема голосування: якщо підсистеми, що використовуються у гібридній системі, генерують бінарні оцінки, то можна рекомендувати лише ті предмети, які були позитивно оцінені кожною з підсистем. Приклад – система P-Tango. У ній підсистеми засновані на змісті та колаборативній фільтрації спочатку мають однакові вагові коефіцієнти, які з часом змінюються в залежності від успішності прогнозів [13].

Перемикання – вибір кращої підсистеми на льоту, під впливом певних наперед заданих критеріїв [13]. Ці критерії можуть бути задані з урахуванням слабких сторін підсистем, щоб не враховувати рекомендації у тих ситуаціях, де підсистема може проявити себе погано. Такий підхід добре проявляє себе у розв'язанні проблеми "холодного старту" для колаборативних систем. Наприклад, якщо у предмета ще недостатньо оцінок для коректної роботи колаборативної підсистеми, замість неї можна використати рекомендації підсистеми, заснованої на змісті. Як приклад такої системи можна навести DailyLearner: у ній спочатку прогноз намагається дати підсистема заснована на змісті, а якщо немає впевненості у його правильності, то застосовуються підсистему засновану на колаборативній фільтрації [13].

Змішані рекомендації – кожна підсистема незалежно генерує свій список рекомендацій і на їх основі формується спільний список [16]. При цьому рекомендації однієї підсистеми доповнюються рекомендаціями іншої. Цей підхід зручно застосовувати коли необхідно зробити одразу багато рекомендацій. Прикладом може слугувати PTV – система рекомендацій телевізійних програм, яка формує загальний список змішуванням разом рекомендацій від двох підсистем, одна з яких заснована на змісті, а інша – на колаборативній фільтрації [13].

Комбінування ознак – розширення підходу на основі змісту шляхом додавання до нього ознак, отриманих через інші підходи [16]. Найчастіше це відбувається через колаборативну фільтрацію: до ознак, застосованих у підсистемі заснованій на змісті додається ще одна ознака – оцінки схожих користувачів. Це дозволяє використовувати оцінки предметів не покладаючись на них повністю [13]. Наприклад, були експерименти з системою машинного навчання на основі правил Ripper, у яких система давала рекомендації фільмів на основі поєднання їх ознак та оцінок користувачів [13].

Посилення ознак – вихідні дані однієї підсистеми використовуються як вхідні дані для іншої. Це дозволяє не змінюючи власної системи підключати до неї сторонні системи [16]. Наприклад, у рекомендаційних системах

заснованих на змісті для певних значень ознак предмета можна визначити пов'язані значення, як це зроблено у системі рекомендацій книг Libra: дані, на основі яких робляться рекомендації, містять інформацію про "пов'язаних авторів" та "пов'язаних заголовків", яка приходить від рекомендаційної системи сайту Amazon.com [13].

Конвеєр – послідовне застосування декількох підсистем для уточнення рекомендацій. При такому підході, спочатку перша підсистема робить грубий відбір, відсіюючи ті предмети, які не попадуть до рекомендацій, а друга – робить більш точне прогнозування. При цьому робота другої підсистеми стає більш ефективною, бо вона не витрачає свої ресурси на предмети, які точно не підійдуть для рекомендації (наприклад, через низьку оцінку). Наприклад, рекомендаційна система ресторанів EntreeC використовує конвеєр з підсистемами заснованими на знаннях та колаборативній фільтрації [13].

Мета-рівень – модель, створена однією підсистемою стає вхідною моделлю для іншої [13]. Цей підхід схожий на посилення ознак, але на відміну від нього, вся модель, а не окремі її частини, подається на вхід до іншої. Наприклад, підсистема заснована на змісті може згенерувати модуль, що міститиме стислі дані про інтереси користувача, а потім ці дані будуть використовуватися іншою підсистемою. Як приклад можна навести систему LaboUr, яка спочатку формує профілі користувачів, а потім порівнює їх [13].

1.2.7 Переваги та недоліки різних підходів до побудови рекомендаційних систем

Розглянемо переваги та недоліки основних підходів до побудови рекомендаційних систем.

Рекомендаційні системи, засновані на колаборативній фільтрації:

а) Переваги:

1) Можна робити крос-жанрові рекомендації;
2) Не потребують знань предметної області;

3) Правильність рекомендацій покращується з часом;
4) Достатньо мати лише інформацію про оцінки;
5) Можливість використання неявної інформації взаємодії користувача з системою.

б) Недоліки:

1) Проблема холодного старту для користувачів;
2) Проблема холодного старту для предметів;
3) Проблема сірої вівці;
4) Проблема популярних товарів;
5) Правильні рекомендації можливі лише за наявності великого набору даних;
6) Необхідно працювати з матрицями з високим рівнем розрідженості;
7) Дилема стабільності-пластичності.

Рекомендаційні системи, засновані на змісті:

а) Переваги:

1) Правильність рекомендацій покращується з часом;
2) Достатньо мати лише профіль користувача та інформацію про предмети;
3) Можливість використання неявної інформації взаємодії користувача з системою;
4) Легко пояснити чому була зроблена рекомендація, бо вони є персоналізованими.

б) Недоліки:

1) Проблема холодного старту для користувачів;
2) Необхідно правильно визначити набір ознак предмета;
3) Правильні рекомендації можливі лише за наявності великого набору даних;

4) Рекомендації прив'язані до поточних інтересів користувача і не можуть вийти за їх рамки;
5) Дилема стабільності-пластичності.

Рекомендаційні системи, засновані на знаннях:

а) Переваги:
   1) Відсутність проблеми холодного старту;
   2) Гарантована якість рекомендацій;
   3) Чутлива до змін уподобань;
   4) Можуть включати ознаки, які не належать предметам;
   5) Рекомендації надаються згідно з потребами користувача.
б) Недоліки:
   1) Відсутність навчання;
   2) Потрібно розробляти базу знань конкретної предметної області.

Демографічні рекомендаційні системи:

а) Переваги:
   1) Можна робити крос-жанрові рекомендації;
   2) Не потребують знань предметної області;
   3) Правильність рекомендацій покращується з часом.
б) Недоліки:
   1) Проблема холодного старту для користувачів;
   2) Проблема сірої вівці;
   3) Правильні рекомендації можливі лише за наявності великого набору даних;
   4) Дилема стабільності-пластичності;
   5) Потрібно збирати інформацію для поділу користувачів на групи.

Рекомендаційні системи, засновані на корисності:

а) Переваги:
   1) Відсутність проблеми холодного старту;
   2) Гарантована якість рекомендацій;
   3) Чутливі до змін уподобань;
   4) Можуть працювати з ознаками, які не належать предметам.

б) Недоліки:
   1) Необхідно визначати функцію корисності для кожного користувача;
   2) Відсутність навчання.

Гібридні рекомендаційні системи:

а) Переваги:
   1) Об'єднують в собі переваги підсистем;
   2) Нівелюють вплив певних недоліків підсистем.

б) Недоліки:
   1) Потребують хорошої точності рекомендацій від підсистем;
   2) Необхідно добре знати підходи побудови кожної підсистеми;
   3) Важко пояснити чому була зроблена рекомендація;
   4) Працюють повільніше та потребують більше обчислювальних потужностей ніж окремі рекомендаційні системи.

Описані переваги та недоліки потрібно брати до уваги при виборі потрібного підходу для використання рекомендаційної системи у певній предметній області з урахуванням її особливостей.

1.3 Постановка задачі

Необхідно розробити систему підтримки прийняття рішень обрання дисципліни за вибором. Система повинна приймати на вхід оцінки студента з його навчальних дисциплін та у відповідь надавати рекомендацію про

дисципліну, яка найбільше йому підходить. Для досягнення поставленої мети необхідно:

1. Обрати найбільш оптимальний підхід реалізації рекомендаційної системи виходячи з переваг та недоліків різних підходів та особливостей предметної області.
2. Розробити алгоритм роботи СППР на основі обраного підходу.
3. Обрати методи програмної реалізації.
4. Виконати програмну реалізацію СППР та провести її тестування.

2. ПРОЕКТУВАННЯ СИСТЕМИ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ ОБРАННЯ ДИСЦИПЛІНИ ЗА ВИБОРОМ

2.1 Вибір підходу до побудови рекомендаційної системи

Предметною областю роботи системи є результати студентів з навчальних дисциплін. Під час аналізу предметної області було виявлено такі її особливості:

- Предметна область містить оцінки студентів з навчальних дисциплін;
- Навчальні дисципліни є двох видів: обов'язкові та на вибір;
- Кожен студент (включаючи того, для якого буде робитися рекомендація) має оцінку за кожну обов'язкову навчальну дисципліну і навпаки;
- Кожен студент (виключаючи того, для якого буде робитися рекомендація) має оцінку з однієї дисципліни за вибором;
- Всі оцінки є позитивними (тобто, кожна оцінка належить інтервалу [60; 100]).

Кожного студента можна представити у вигляді вектора, який складатиметься з його оцінок з обов'язкових навчальних дисциплін. Тоді ці вектори можна об'єднати у матрицю оцінок. При цьому, оскільки кожен студент має оцінку з кожної обов'язкової дисципліни, то розрідженість такої матриці буде дорівнювати 0.

Виходячи з цього аналізу та переваг та недоліків підходів до побудови рекомендаційних систем, наведених у пункті 1.2.7, можна зробити висновок, що колаборативна фільтрація найкраще підходить для СППР, що розробляється. Доцільно використати підхід «користувач-користувач». Він має високу точність за умови великого набору даних та їх низької розрідженості – саме те, що наявне у нашій предметній області. При цьому, також відсутня проблема холодного старту для користувачів, тобто будь-який студент зможе отримати рекомендацію. Вплив проблеми популярних товарів буде мінімізовано завдяки алгоритму роботи системи.

2.2 Розробка рекомендаційного алгоритму

Для даної предметної області було розроблено алгоритм колаборативної системи рекомендацій заснованої на сусідстві за принципом «користувач-користувач»:

Для вхідних даних «дисципліна-оцінка за дисципліну» (i, x) (тобто $u_i = x$, де u – вектор оцінок студента):

1. Розрахувати схожість студентів на даного студента на основі їх векторів оцінок;
2. Відсортувати студентів за їх схожістю на даного;
3. Для кожної дисципліни за вибором знайти n студентів, які найбільше схожі на даного;
4. Розрахувати середньо зважену оцінку кожної з дисциплін серед дисциплін на вибір на основі оцінок студентів, яких було обрано на кроці 3;
5. Відсортувати дисципліни за вибором відповідно до їх середньо зважених оцінок, знайдених на кроці 4;
6. Рекомендувати студенту m верхніх дисциплін за вибором.

Для оцінювання схожості студентів будемо використовувати косинус подібності (формула (1.2)), для розрахунку прогнозованої оцінки – середньо зважене значення (формула (1.6)).

Для спрощення обрахунків та отримання більш коректних результатів, для знаходження коефіцієнту схожості оцінки будемо нормалізовувати в інтервал [0.01; 1.01] за формулою (2.1):

$$normS(S) = \frac{S - S_{min}}{S_{max} - S_{min}} + 0.01. \quad (2.1)$$

де S – вхідна оцінка;

S_{min} – мінімально можлива оцінка (у нашому випадку 60);

S_{max} – максимально можлива оцінка (у нашому випадку 100).

Нехай u_1, u_2 – вектори оцінок двох студентів, u'_1, u'_2 – вектори скоригованих оцінок. Тоді їх схожість можна подати у вигляді $similarity(u_1,u_2) = \cos(u'_1,u'_2)$.

Якщо $\cos(u'_1,u'_2) = 0$, це означає, що вектори перпендикулярні, тобто студенти абсолютно не схожі один на одного. Якщо $\cos(u'_1,u'_2) = 1$, то вектори або збігаються, або паралельні – студенти однакові за своїми оцінками.

Наприклад, $u_1 = (60, 100, 60)$, $u_2 = (68, 60, 100)$, після нормалізації отримуємо $u'_1 = (0.01, 1.01, 0.01)$, $u'_2 = (0.21, 0.01, 1.01)$, отже $similarity(u_1,u_2) = \cos((0.01, 1.01, 0.01), (0.21, 0.01, 1.01) \approx 0.09$ – студенти майже не схожі один на одного;

$u_1 = (68, 76, 100)$, $u_2 = (76, 68, 76)$, після нормалізації отримуємо $u'_1 = (0.21, 0.41, 1.01)$, $u'_2 = (0.41, 0.21, 0.41)$, отже $similarity(u_1,u_2) = \cos((0.21, 0.41, 1.01), (0.41, 0.21, 0.41) \approx 0.86$ – студенти схожі;

$u_1 = (92, 76, 76)$, $u_2 = (76, 76, 68)$, після нормалізації отримуємо $u'_1 = (0.81, 0.41, 0.41)$, $u'_2 = (0.41, 0.21, 0.21)$, отже $similarity(U_1,U_2) = \cos((0.81, 0.41, 0.41), (0.41, 0.21, 0.21)) \approx 1$ – студенти однакові за своїми оцінками.

Розглянемо роботу алгоритму на прикладі. Значення n для кроку 3 алгоритму встановимо рівним 5, значення m для кроку 6 рівним 1.

Нехай маємо 5 студентів (u_1, …, u_5), 5 обов'язкових дисциплін (j_1, …, j_5), 2 дисципліни за вибором (j_6, …, u_7), оцінки студентів з цих дисциплін та інформацію про те, яку дисципліну за вибором обрав кожен із студентів та яку оцінку за неї отримав. Побудуємо матрицю оцінок «студент – оцінка за певну дисципліну» – таблиця 2.1

Таблиця 2.1 Матриця оцінок

	j_1	j_2	j_3	j_4	j_5	j_6	j_7
u_1	90	100	80	79	93	91	
u_2	81	67	78	97	100		99
u_3	93	83	70	81	79		80
u_4	100	87	72	70	63	67	
u_5	80	62	74	91	67		89

Також маємо студента u_0 для якого необхідно зробити рекомендацію. Його оцінки наведено у таблиці 2.2:

Таблиця 2.2 Вхідний вектор оцінок

	j_1	j_2	j_3	j_4	j_5
u_0	75	80	63	71	95

Нормалізуємо оцінки з дисциплін j_1, …, j_5. Наприклад, нормалізація оцінки першого студента з першої дисципліни:

$$normS(S_{1,1}) = \frac{90 - 60}{100 - 60} + 0.01 = 0.76$$

Результат нормалізації показано у таблиці 2.3

Таблиця 2.3 Результат нормалізації оцінок з таблиць 1.1 та 1.2

	j_1	j_2	j_3	j_4	j_5
u_1	0.76	1.01	0.51	0.485	0.835
u_2	0.535	0.185	0.46	0.935	1.01
u_3	0.835	0.585	0.26	0.535	0.485
u_4	1.01	0.685	0.31	0.26	0.085
u_5	0.51	0.06	0.36	0.785	0.185
u_0	0.385	0.51	0.085	0.285	0.885

Знайдемо схожість ключового студента u_0 з іншими. Наприклад, його схожість на студента u_1 буде розрахована наступним чином:

$$similarity(U_1, U_0) =$$

$$= \cos(0.76, 1.01, 0.51, 0.485, 0.835), (0.385, 0.51, 0.085, 0.285, 0.885) =$$

$$= \frac{0.76 * 0.385 + 1.01 * 0.51 + 0.51 * 0.085 + 0.485 * 0.285 + 0.835 * 0.885}{\sqrt{0.76^2 + 1.01^2 + 0.51^2 + 0.485^2 + 0.835^2}\sqrt{0.385^2 + 0.51^2 + 0.085^2 + 0.285^2 + 0.885}}$$

$$= \frac{1.670404}{1.670404 * 1.131371} = 0.914492$$

Дані про схожість ключового студента на інших наведено у таблиці 2.4.

Таблиця 2.4 Схожість студента u_0 на інших

	u_1	u_2	u_3	u_4	u_5
u_0	0,914492	0,851011	0,847519	0,626895	0,558029

Розрахуємо середньо зважену оцінку для дисциплін за вибором на основі схожості студентів. Наприклад, для j_6:

$$u_{0,6} = \frac{91 * 0.914492 + 67 * 0.626895}{0.914492 + 0..626895} = \frac{124.132}{1.526338} = 81.239$$

Середньо зважені оцінки для дисциплін за вибором наведено у таблиці 2.5

Таблиця 2.5 Середньо зважені оцінки дисциплін за вибором

	j_6	j_7
u_0	81,239	89,39105

Отже, можемо рекомендувати j_7 як дисципліну за вибором для цього студента.

2.3 Вибір інструментів реалізації

Згідно з постановкою задачі, необхідно виконати програмну реалізацію СППР. Для досягнення цієї цілі було обрано мову програмування Java.

Java – це об'єктно-орієнтована мова програмування загального призначення. Вона була розроблена за принципом WORA - «Write Once, Run Anywhere» («написати один раз, запускати де завгодно») [17]. Тобто, скомпільований код Java може запускатися на всіх платформах з підтримкою Java без необхідності перекомпіляції.

Для розробки на Java необхідно встановити на свій комп'ютер JRE та JDK. JRE (Java Runtime Environment) – це середовище, яке здатне запускати Java програми без компіляції. Воно складається з JVM та бібліотек класів. JVM (Java Virtual Machine) – виконує байт-код Java, який було скомпільовано з вихідних файлів. Саме JRE робить Java крос-платформеною мовою та дозволяє працювати Java програмам однаково на різних ОС. JDK (Java Development Kit) – набір, який дозволяє розробляти програми мовою Java. До нього входять компілятор, стандартні бібліотеки, документація та JRE. Зв'язок між JVM, JRE та JDK зображено на рисунку 2.1.

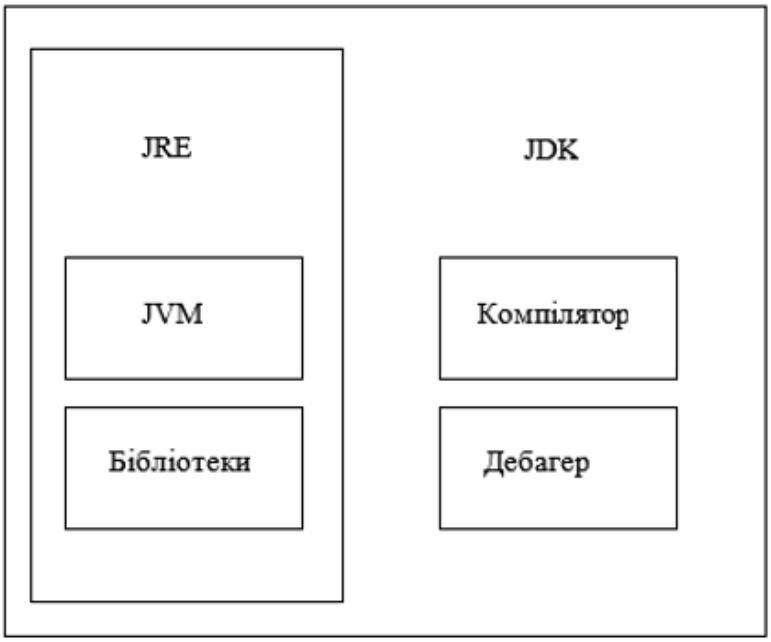

Рисунок 2.1 – Залежність між JVM, JRE та JDK

Основні переваги Java полягають у наступному [18, 19]:

- Об'єктно-орієнтована мова;
- Крос-платформена мова;
- Є безпечною мовою;
- Підтримка багатопотоковості;
- Підтримка автоматичного керування пам'яттю;
- Легко вивчити;

- Наявність великої активної спільноти.

До недоліків можна віднести [18, 19]:

- Низька продуктивність;
- Відсутність нативного дизайну;
- Важкий для розуміння код, особливо у великих проектах;
- Проблеми з управлінням пам'яттю.

Система повинна мати графічний інтерфейс. Для його реалізації було обрано бібліотеку JavaFX.

JavaFX – це програмна платформа, що складається з набору графічних та медіа пакетів, що дозволяє розробникам Java створювати настільні та web додатки. Вона пропонує багатий набір API для графіки та медіа. Також ця бібліотека має інтерфейси, за допомогою яких розробники можуть комбінувати графічну анімацію та управління інтерфейсом. За допомогою JavaFX можна створювати додатки для різних ОС та пристроїв. Такі додатки працюватимуть всюди, де встановлено JRE.

JavaFX повністю замінила собою стару бібліотеку Swing як стандартну бібліотеку Java для розробки інтерфейсів. Починаючи з Java 8 вона за замовчуванням йде у комплекті з JRE та JDK. JavaFX можна використовувати з технологіями на базі JVM, такими як Java, Groovy та JRuby.

Основні особливості JavaFX:

- Використовується FXML як мова розмітки інтерфейсу;
- Підтримка стилізації за допомогою CSS;
- Підтримка прив'язки даних;
- Підтримка спеціальних ефектів (тіні, відбиття, розмиття, тощо) та анімацій;
- Підтримка сенсорних дисплеїв;
- Підтримка HTLM5, аудіо- та відеоконтенту;
- Сумісність з Swing.

Для зберігання даних необхідно використовувати базу даних. Виходячи з того, що об'єм даних не буде надто великим та є потреба у тому, щоб додаток міг запускатися будь-де без встановлення додаткового програмного забезпечення, бази даних на кшталт Oracle Database чи MySQL нам не підходять. Необхідно використати базу даних, яка займає мало місця і може бути вбудована у додаток. Тому вирішено було використати SQLite.

SQLite – це база даних, яку можна вбудовувати в додатки. Вона написана мовою програмування C та її вихідний код відкритий кожному для будь-яких цілей. SQLite займає дуже мало місця, тому її можна використовувати на будь-яких пристроях. Вона працює дуже швидко: завантажуються лише потрібні дані, а при редагуванні невеликих частин файлу замінюються лише змінені частини. SQLite не потрібно встановлювати та налаштовувати. Потрібно лише створити базу даних (наприклад, з використанням DB Browser for SQLite) і вона одразу готова до роботи. Однією з важливих особливостей SQLite є її надійність. Її вихідний код на 100% покритий тестами та випуск кожної версії ретельно тестується [20].

SQLite підтримує мінімальний достатній набір SQL функцій. Наприклад, не підтримуються такі функції як: редагування чи видалення стовпця у таблиці (можна лише перейменувати стовпець чи додати новий) [21], використання процедур, контроль доступу до таблиць [22].

SQLite є базою даних з динамічною типізацією даних. При вставці нового значення до таблиці, база даних намагається привести його до типу даних стовпця. І якщо це неможливо, то воно вставляється як є. В такому випадку тип стовпця є лише рекомендацією. Це називається «type affinity» («спорідненість типів») та використовується для забезпечення сумісності SQLite з іншими базами даних [23].

Для збирання проекту будемо використовувати Maven. Apache Maven - фреймворк для автоматизації збирання проектів. Для цього він використовує опис структури проекту в файлах на мові POM. Ці файли містять інформацію про модуль, до якого вони відносяться, про проект, налаштування збирання

проекту, налаштування оточення, список бібліотек (залежностей) та плагінів, які використовуються цим модулем [24].

Maven був розроблений для розв'язання проблем, з якими стикалися розробники при використанні Ant. В Ant розробникам доводиться самостійно задавати всі етапи збирання додатку. Maven автоматизує більшість етапів шляхом застосування більш жорстких стандартів організації коду. Як результат, легше розпочати роботу над стандартними проектами. Maven стандартизує та спрощує процес збирання [25]. Він побудований і працює за принципом «convetion over configuration» («угода головніша за конфігурацію»). З Maven легко працювати, поки проект дотримується стандартів.

До ключової особливості Maven належить автоматичне завантаження залежностей та управління їх версіями [25]. Також Maven підтримує багатомодульні проекти та додавання потрібних функцій через плагіни.

3. РЕАЛІЗАЦІЯ СИСТЕМИ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ ОБРАННЯ ДИСЦИПЛІНИ ЗА ВИБОРОМ

3.1 Створення бази даних та реалізація запитів

База даних необхідна для зберігання оцінок студентів. При цьому, нам неважливо, якій саме людині ці оцінки належать. Виходячи з цих вимог було спроектовану ER-діаграму, яку зображено на рисунку 3.1.

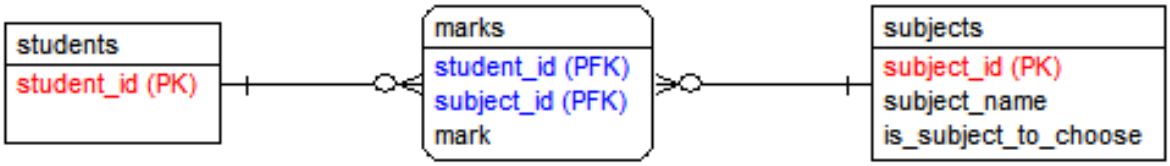

Рисунок 3.1 – ER-діаграма бази даних системи

На основі ER-діаграми та аналізу предметної області, виконаного у пункті 2.1, було визначено необхідні таблиці, стовпці, типи даних, їх призначення та обмеження. Їх можна побачити у таблиці 3.1.

Таблиця 3.1 – Логічна реалізація бази даних

Таблиця	Поле	Зміст	Тип	Ключі	Обмеження
students	student_id	Номер студента	INTEGER	PK	NOT NULL
subjects	subject_id	Номер предмета	INTEGER	PK	NOT NULL
	subject_name	Назва предмета	TEXT		NOT NULL
	is_subject_to_choose	Тип предмета	INTEGER		NOT NULL, IN (0, 1)
marks	student_id	Номер студента	INTEGER	PFK	NOT NULL
	subject_id	Номер предмета	INTEGER	PFK	NOT NULL
	mark	Оцінка	INTEGER		NOT NULL, BETWEEN 60 and 100

Таблиця *students* містить список студентів. Вона необхідна для підтримки цілісності даних. Таблиця subjects містить список дисциплін, їх назву та помітку про те, до якого виду належить дисципліна. Таблиця *marks* містить дані про те, яку оцінку отримав певний студент за певну дисципліну.

На основі логічної реалізації бази даних необхідно створити її фізичну реалізацію (скрипти створення таблиць). Для додавання таблиць у SQLite використовується команда CREATE TABLE. Його синтаксис [26]:

```
CREATE TABLE [IF NOT EXISTS][schema_name].table_name (
    column_1 data_type PRIMARY KEY,
    column_2 data_type NOT NULL,
    column_3 data_type DEFAULT 0,
    ...,
    table_constraints
) [WITHOUT ROWID];
```

Аргументи команди:

- table_name – назва таблиці;
- IF NOT EXISTS – створення таблиці з тим же іменем без цієї опції призведе до помилки;
- schema_name – назва схеми, до якої належить таблиця;
- column_1, column_2, column_3, … - назви стовпців;
- data_type – тип даних стовпця (INTEGER, BLOB, TEXT, NUMERIC, REAL);
- DEFAULT 0 – задає значення за замовчуванням;
- table_constraints – обмеження таблиці (PRIMARY KEY, FOREIGN KEY, UNIQUE, та CHECK);
- WITHOUT ROWID – вказує, що таблицю треба створити без прихованої колонки rowid, яка зберігає унікальний ідентифікатор рядка таблиці.

Для вставки даних до таблиць використовується команда INSERT. Її синтаксис [27]:

```
INSERT INTO table_name (column_1,column_2 ,...) VALUES( value_1, value_2 ,...);
```

Аргументи команди:

- table_name – назва таблиці, до якої заносяться дані;
- column_1, column_2, … – назви стовпців, до яких заносяться дані;
- value_1, value_2, … – дані, які заносяться до таблиці. Їх кількість повинна співпадати з кількістю стовпців, вказаних у списку стовпців.

Для функціонування системи необхідно буде отримувати з бази даних інформацію про список дисциплін та оцінки студентів. Для цього використовується команда SELECT. Її синтаксис [28]:

```
SELECT [DISTINCT] column_1, column_2, ...
FROM table_1
[JOIN table_2 ON join_condition]
[WHERE row_filter]
[ORDER BY column]
[LIMIT count OFFSET offset]
[GROUP BY column
[HAVING group_filter];
```

Аргументи команди:

- column_1, column_2, ... – назви стовпців, з яких необхідно отримати дані;
- table_1 назва таблиці, з якої необхідно отримати дані
- DISTINCT – вказує на те, що непотрібно виводити дублікати у результатах запиту;

- JOIN table_2 ON join_condition – використовується для приєднання таблиць для одночасного пошуку чи фільтрації у декількох таблицях;
- WHERE row_filter – використовується для фільтрації даних;
- ORDER BY column – використовується для сортування результатів запиту:
- LIMIT count OFFSET offset – використовується для обмеження кількості повернених запитом рядків;
- GROUP BY column – використовується для групування результатів запиту;
- HAVING group_filter – використовується для фільтрації результатів запиту при групуванні.

Фізичну реалізацію бази даних та запити, які використовуються для отримання інформації з бази даних наведено у додатку А.

3.2 Створення додатку системи підтримки прийняття рішень

Додаток було вирішено створювати на мові Java. Виходячи з вимог, було визначено наступні класи:

- *Main* – точка входу у програму;
- *Student* – містить інформацію про студента;
- *PossibleSubject* – містить інформацію про дисципліну за вибором;
- *Controller* – містить обробники подій;
- *RecommendedSystem* – містить основну логіку рекомендаційної системи;
- *PossibleSubject* – містить інформацію про дисципліну за вибором;
- *InvalidInputException* – клас для обробки виняткових ситуацій;
- *DBUtils* – клас для підключення до бази даних;

- *RecommendedSystemDAO* – інтерфейс який описує необхідні методи доступу до даних з бази даних;
- *RecommendedSystemDAOImpl* – клас який реалізує методи, описані в *RecommendedSystemDAO*.

Взаємодію між класами зображено на діаграмі класів на рисунку 3.2

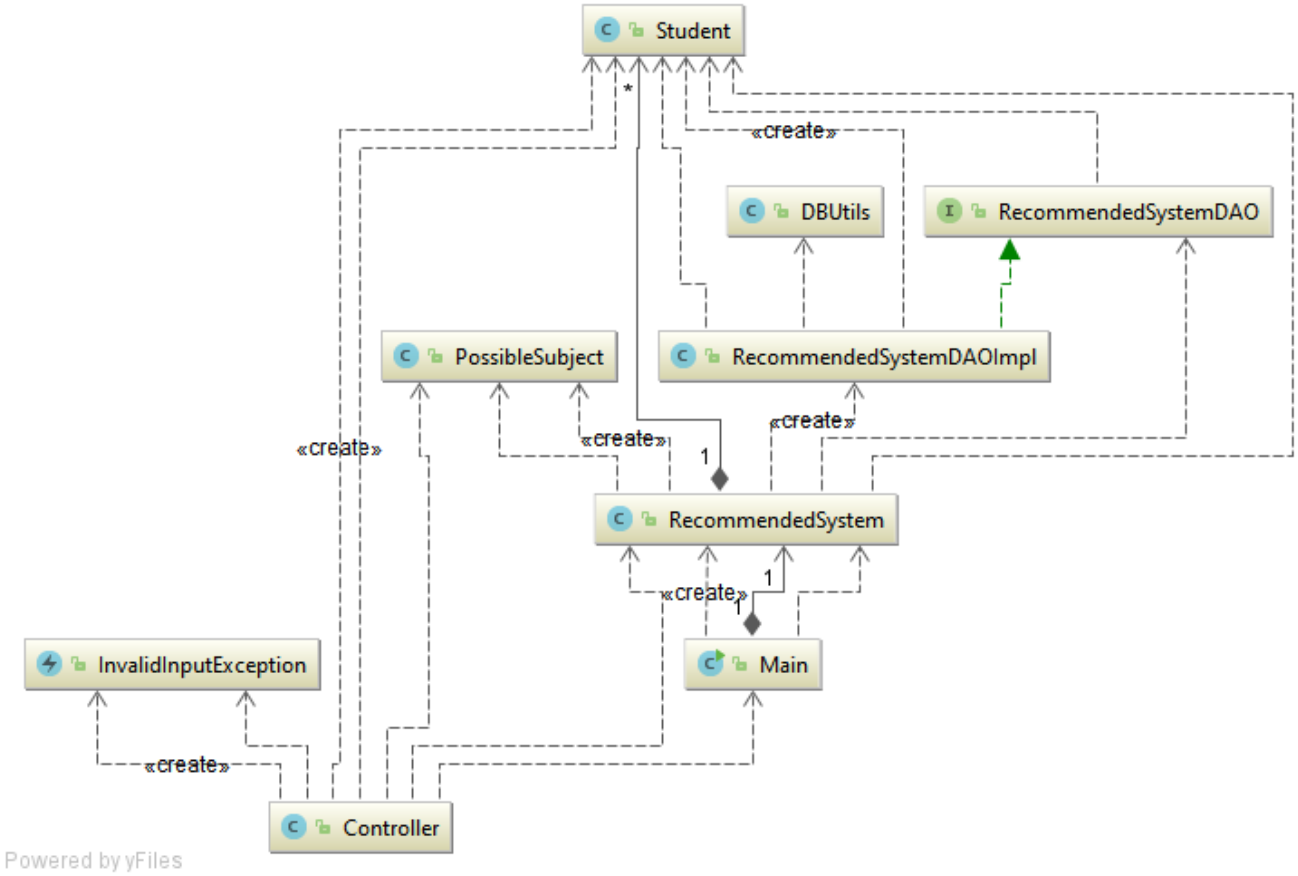

Рисунок 3.2 – ER-діграма бази даних системи

Розглянемо реалізацію основних методів додатку. Повний код додатку наведено у Додатку Б.

Метод пошуку косинусу подібності з класу RecommendedSystem:

```
private static double cosineSimilarity(Student a, Student b) {
    double sum = 0;
    double sumA = 0;
    double sumB = 0;
    for (String subject : subjects) {
        sum += (((a.getMarks().get(subject) - MIN_MARK) / DIFF) + 0.01) *
(((b.getMarks().get(subject) - MIN_MARK) / DIFF) + 0.01);
```

```
        sumA += (((a.getMarks().get(subject) - MIN_MARK) / DIFF) + 0.01) *
(((a.getMarks().get(subject) - MIN_MARK) / DIFF) + 0.01);
        sumB += (((b.getMarks().get(subject) - MIN_MARK) / DIFF) + 0.01) *
(((b.getMarks().get(subject) - MIN_MARK) / DIFF) + 0.01);
    }

    double res = Math.sqrt(sumA) * Math.sqrt(sumB);

    if (res == 0) {
        return 0;
    } else {
        return sum / res;
    }
}
```

Аргументами цього методу є 2 об'єкти класу Student, для яких необхідно розрахувати схожість за допомогою формули (1.2).

Метод пошуку рекомендованих дисциплін з класу RecommendedSystem:

```
public List<PossibleSubject> makeRecommendations(Student student) {
    List<PossibleSubject> possibleSubjects = new LinkedList<>();
    Map<String, List<Student>> studentsBySubject = new LinkedHashMap<>();
    for (String subject: subjectsToChoose) {
        studentsBySubject.putIfAbsent(subject, new LinkedList<>());
    }
    for (Student s : students) {
        s.setSimilarity(cosineSimilarity(student, s));
        System.out.println(cosineSimilarity(student, s));
        studentsBySubject.get(s.getChosenSubject().getKey()).add(s);
    }
    for (Map.Entry<String, List<Student>> entry : studentsBySubject.entrySet()) {
        System.out.println(entry.getKey() + " : " );
        for (Student s: entry.getValue()) {
            System.out.println(s.getChosenSubject() + " : " + s.getSimilarity());
        }
        System.out.println("**************************");
        List<Student> resultStudents = entry.getValue().stream()
                .sorted(comparing(Student::getSimilarity,
comparing(Math::abs)).reversed())
                .limit(LIMIT_OF_SIMILAR_STUDENTS)
                .collect(toList());
        double sumOfSimilarities = 0;
        double sumOfMarks = 0;
        for (Student s: resultStudents) {
            sumOfMarks += s.getChosenSubject().getValue() * s.getSimilarity();
            sumOfSimilarities += s.getSimilarity();
            System.out.println(s.getChosenSubject() + " : " + s.getSimilarity());
        }
        System.out.println(sumOfSimilarities);
        System.out.println(sumOfMarks);
        PossibleSubject possibleSubject = new PossibleSubject(entry.getKey());
        possibleSubject.setSumOfMarks(sumOfMarks);
        possibleSubject.setSumOfSimilarities(sumOfSimilarities);
        possibleSubjects.add(possibleSubject);
    }
    List<PossibleSubject> resultSubjects = possibleSubjects.stream()
```

```
            .sorted(comparing(PossibleSubject::getResult,
comparing(Math::abs)).reversed())
            .limit(LIMIT_OF_SUBJECTS_TO_CHOOSE)
            .collect(toList());
    System.out.println(possibleSubjects);
    return resultSubjects;
}
```

Аргументом цього методу є об'єкт класу Student, для якого необхідно зробити рекомендацію. Спочатку обчислюється схожість цього студента на інших, потім з для кожної дисципліни за вибором серед студентів які її обрали береться декілька студентів найбільш схожих на ключового (їх кількість зберігається у змінній LIMIT_OF_SIMILAR_STUDENTS) та на основі їх оцінок з цієї дисципліни за формулою (1.6) обчислюється середнє зважене значення оцінки з дисципліну. Потім список дисциплін сортується за цим значенням та з декілька кращих (їх кількість зберігається у змінній LIMIT_OF_SUBJECTS_TO_CHOOSE).

3.3 Збирання та тестування додатку

Для збирання проекту було використано Maven з наступними плагінами:

- maven-compiler-plugin – потрібен для компіляції вихідних файлів проекту;
- maven-assembly-plugin – потрібен для збирання проекту у jar-файл, який міститиме у собі файли ресурсів (розташовані у директорії \src\main\resources).

Для генерації jar-файлу застосуємо команду clean compile assembly:single, де clean – видаляє файли, згенеровані під час попереднього збирання проекту, complie – компілює вихідні файли проекту, assembly:single – збирає скомпільовані класи та ресурси у jar-файл.

Після виконання команди отримуємо згенерований jar-файл у директорії target. Всередині він містить скомпільовані класи, файли ресурсів та файл

MANIFEST.MF, який містить додаткову інформацію, таку як версія додатка, ким він був створений, точка входу у програму, тощо (рисунок 3.3)

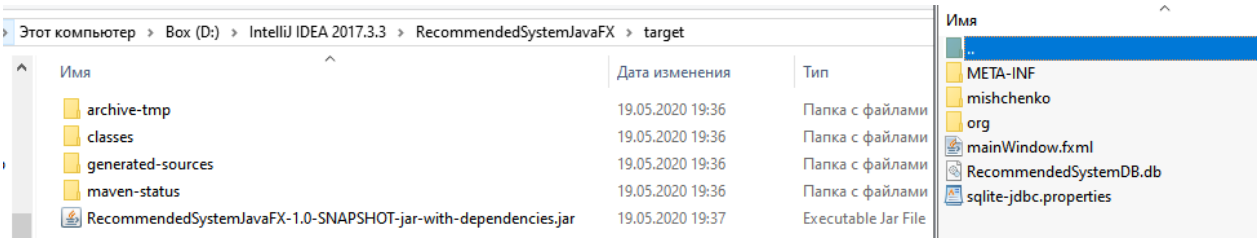

Рисунок 3.3 – Згенерований jar-файл

Запустимо додаток. Початкове вікно зображено на рисунку 3.4

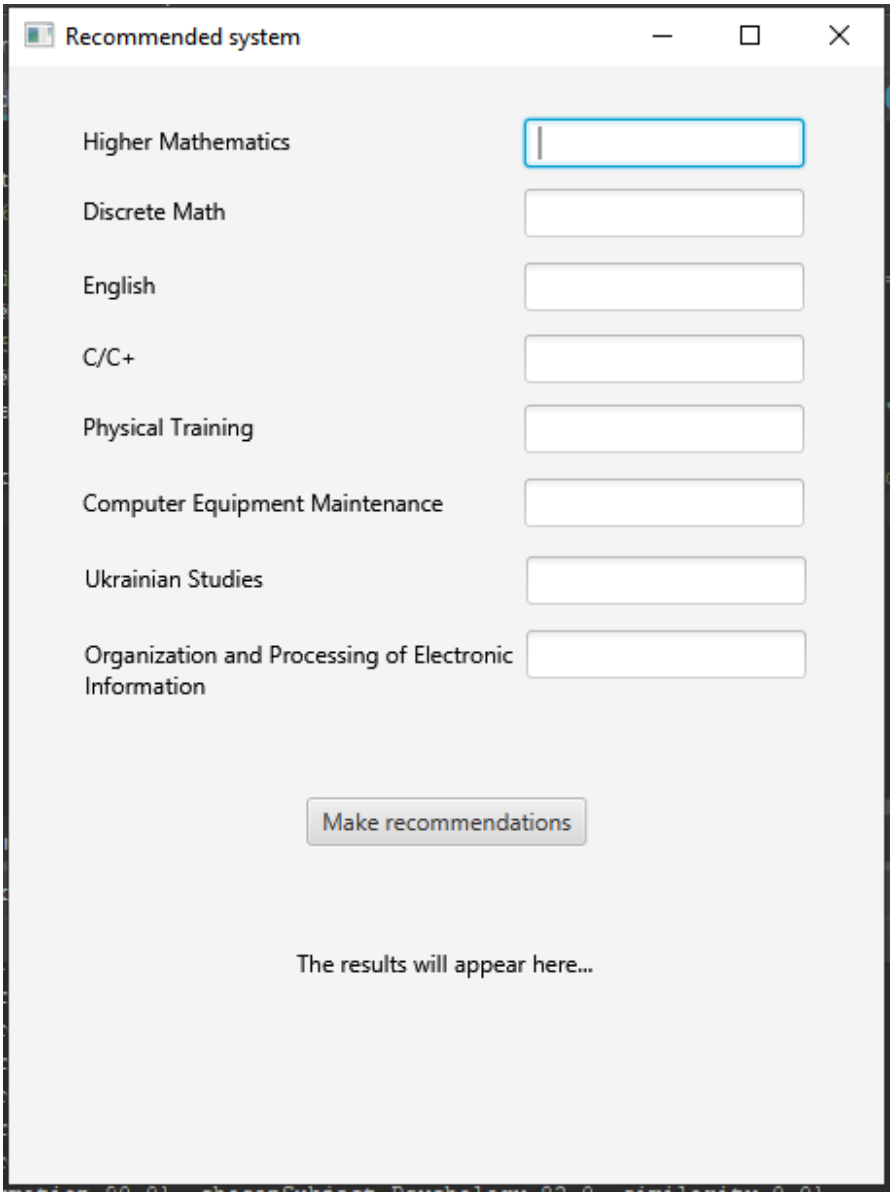

Рисунок 3.4 – Вікно додатку після запуску

Протестуємо рекомендаційний алгоритм для різних оцінок.

Нехай вектор оцінок студента $u_1=(100; 100; 100; 100; 100; 100; 100; 100)$. Вводимо дані до системи, тиснемо «Make recommendations» та отримуємо результат, який зображено на рисунку 3.5:

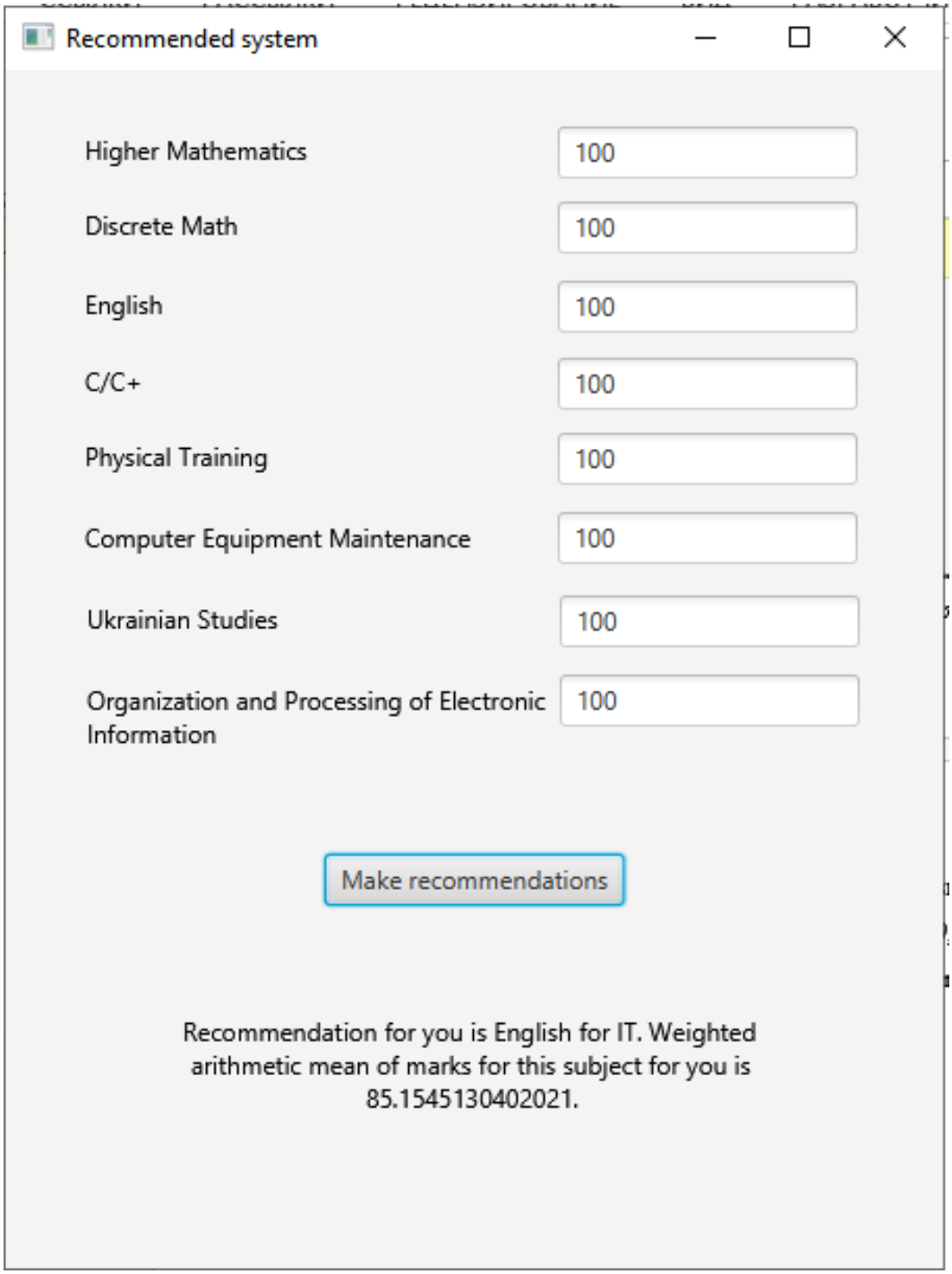

Рисунок 3.5 – Рекомендація для студента u_1

Цьому студенту краще обрати дисципліну English for IT.

Перевіримо систему для вектора u_2=*(78; 87; 91; 84; 71; 90; 100; 89)* (див. рисунок 3.6):

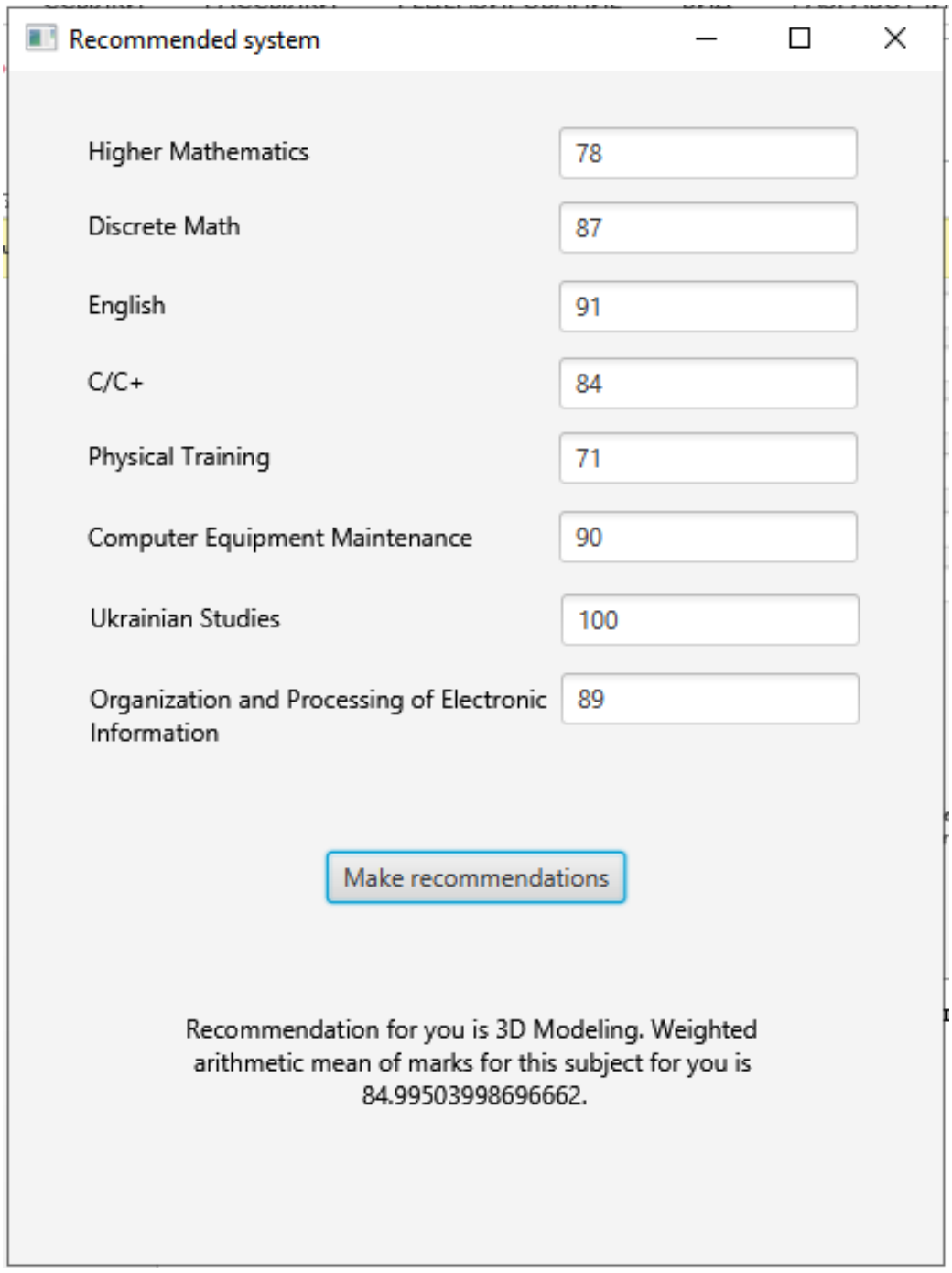

Рисунок 3.6 – Рекомендація для студента u_2

Цьому студенту рекомендується дисципліна 3D Modeling.

Нехай вектор оцінок студента $u_3=(74; 91; 100; 79; 60; 78; 83; 84)$. Тоді отримаємо рекомендацію, зображену на рисунку 3.7

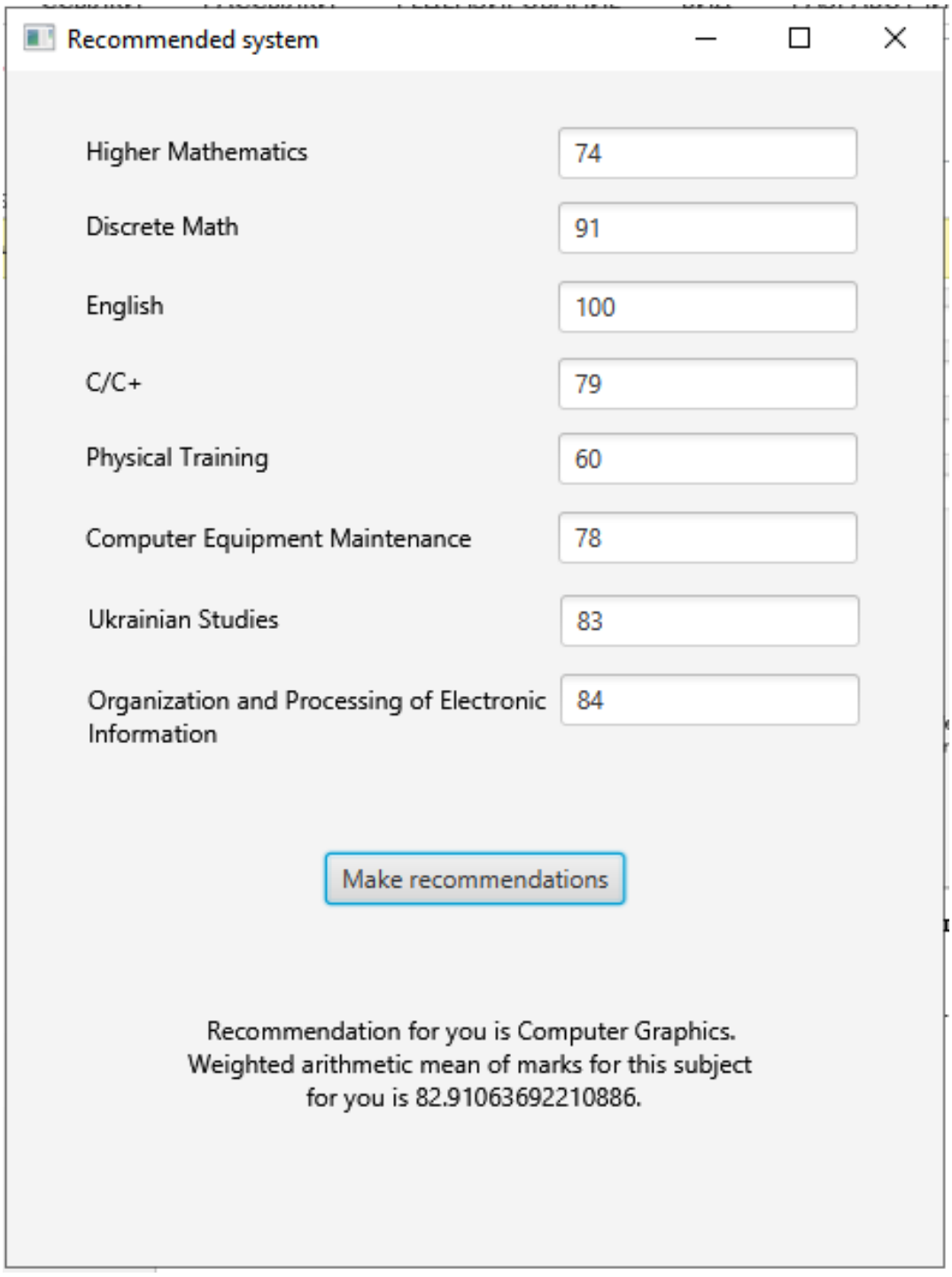

Рисунок 3.7 – Рекомендація для студента u_3

Рекомендовано було дисципліну Computer Graphics.

Перевіримо систему для вектора u_4=*(60; 60; 60; 60; 60; 60; 60; 60)*. Результат можна побачити на рисунку 3.8.

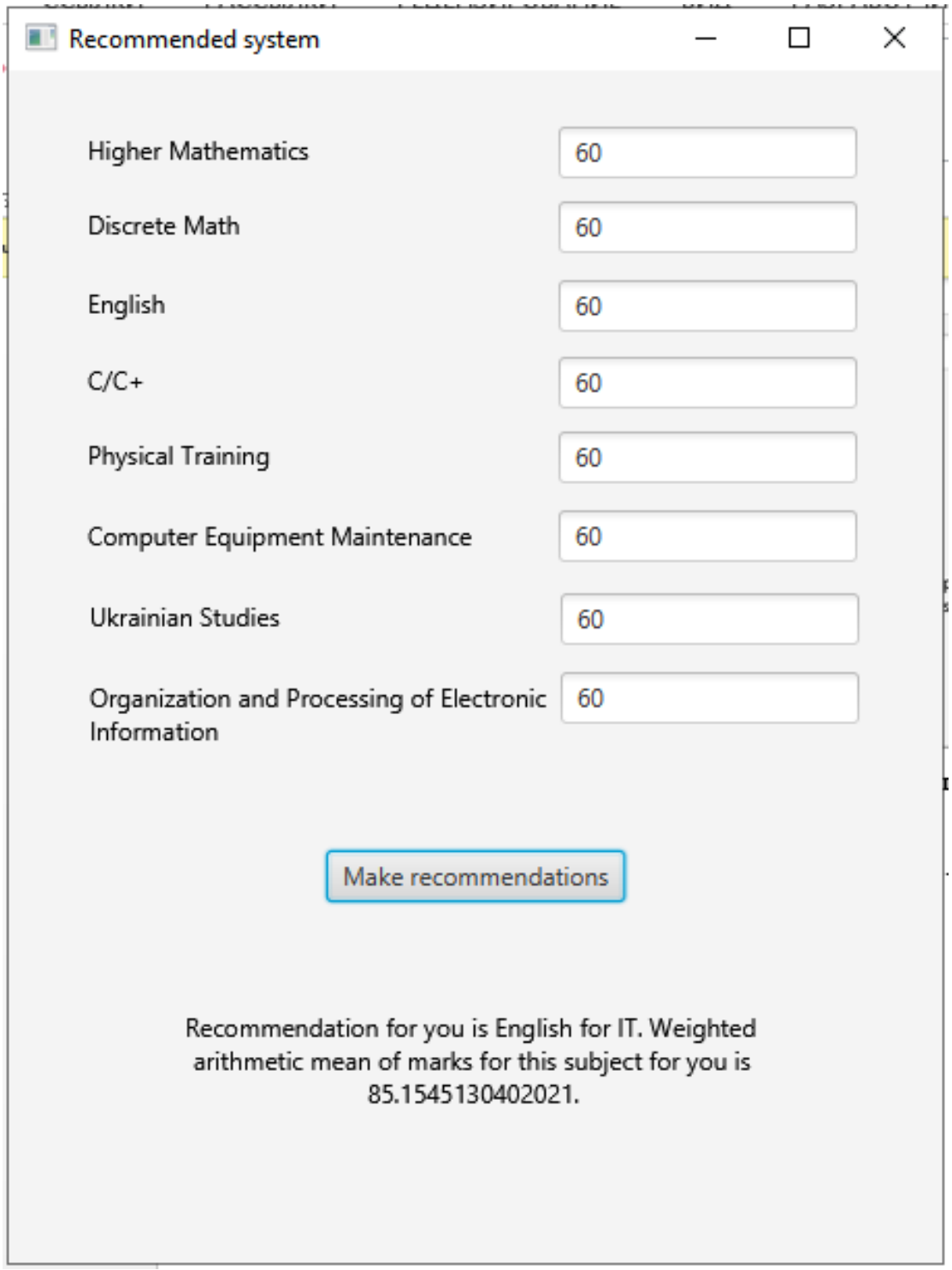

Рисунок 3.8 – Рекомендація для студента u_4

У якості рекомендації було обрано дисципліну English for IT.

Об'єднаємо отримані результати у таблиці 3.2

Таблиця 3.2 – Зведені результати тестування

Вектор оцінок	Рекомендована дисципліна	Середньозважена оцінка
u_1=(100; 100; 100; 100; 100; 100; 100; 100)	English for IT	~85.15
u_2= (78; 87; 91; 84; 71; 90; 100; 89)	3D Modeling	~85
u_3= (74; 91; 100; 79; 60; 78; 83; 84)	Computer Graphics	~82.91
u_4= (60; 60; 60; 60; 60; 60; 60; 60)	English for IT	~85.15

Можна помітити, що результати для u_1 та u_4 ідентичні, попри те, що їх оцінки абсолютно різні. Але це правильне поводження системи при використанні косинусу схожості (бо вектори їх оцінок є колінеарними) і його можна пояснити логічно: ці студенти знають всі предмети на одному рівні, без перекосів у яку-небудь область, тому і рекомендація для них однакова.

ВИСНОВКИ

У ході роботи було проведено дослідження літератури з теми «Рекомендаційні системи». У результаті аналізу отримано інформацію щодо особливостей роботи підходів до побудови рекомендаційних систем, їх переваг та недоліків. Було проведено аналіз предметної області та виходячи з них обрано колаборативну фільтрацію на основі користувачів як найкращий підхід у даній ситуації.

На основі обраного підходу та виходячи з вимог до системи підтримки прийняття рішень було розроблено алгоритм її роботи. У якості мови програмування для реалізації було обрано Java, для зберігання даних про оцінки було використано СУБД SQLite, для розробки користувацького інтерфейсу обрано JavaFX. Для збирання проекту вирішено обрати Maven. Було спроектовано базу даних та структуру класів додатку. На їх основі з використанням перелічених інструментів було розроблено додаток, який було успішно протестовано.

СПИСОК ЛІТЕРАТУРИ

1. Пфанштиль И. Статистика YouTube 2019. Инфографика [Електронний ресурс] / Ия Пфанштиль – Режим доступу до ресурсу: https://exlibris.ru/news/statistika-youtube-2019-infografika/.Mining of Massive Datasets
2. Rating Aggregation in Collaborative Filtering Systems / F. Garcin, B. Faltings, R. Jurca, N. Joswig // RecSys '09: Proceedings of the third ACM conference on Recommender systems, ACM, New York, NY, USA (2009) / / F. Garcin, B. Faltings, R. Jurca, N. Joswig. – New York, 2009. – С. 349–352.
3. Agarwal A. Similarity Measures used in Recommender Systems: A Study / A. Agarwal, M. Chauhan. // International Journal of Engineering Technology Science and Research IJETSR. – Volume 4, Issue 6, 2017.
4. Item-based collaborative filtering [Електронний ресурс] // cs.carleton.edu – Режим доступу до ресурсу: http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/itembased.html.
5. Ghisloti R. Slope One [Електронний ресурс] / Renata Ghisloti // girlincomputerscience. – 2010. – Режим доступу до ресурсу: http://girlincomputerscience.blogspot.com/2010/11/slope-one.html.
6. Espinosa A. Slope One Predictors for Online Rating-Based Collaborative Filtering [Електронний ресурс] / Andrés Espinosa // @andresespinosapc. – 2017. – Режим доступу до ресурсу: https://medium.com/@andresespinosapc/today-im-writing-about-slope-one-predictors-for-online-rating-based-collaborative-filtering-by-3c06677f75c6.
7. Lemire D. Slope One Predictors for Online Rating-Based Collaborative Filtering / D. Lemire, A. Maclachlan // Proceedings of the SIAM Data Mining (SDM'05), Newport Beach, California, April 21-23, 2005 / D. Lemire, A. Maclachlan. – Newport Beach, 2002.
8. Grover P. Various Implementations of Collaborative Filtering [Електронний ресурс] / Prince Grover // @pgrover3. – 2017. – Режим доступу до ресурсу:

https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0.

9. Bondarenko K. Факторизация в рекомендательных системах. Что такое и с чем её едят ? [Електронний ресурс] / Kirill Bondarenko // @bond.kirill.alexandrovich. – 2019. – Режим доступу до ресурсу: https://link.medium.com/tG9rbxNbJ6.
10. Understanding TF-IDF (Term Frequency-Inverse Document Frequency) in python [Електронний ресурс] // @azunan3. – 2019. – Режим доступу до ресурсу: https://medium.com/@azunan3/understanding-tf-idf-term-frequency-inverse-document-frequency-in-python-373070acb895.
11. Raman V. Recommender Engine — Under The Hood [Електронний ресурс] / Venkat Raman // @venksaiyan. – 2017. – Режим доступу до ресурсу: https://towardsdatascience.com/recommender-engine-under-the-hood-7869d5eab072.
12. Aggarwal C. Knowledge-Based Recommender Systems / Charu Aggarwal // Recommender Systems: The Textbook / Charu Aggarwal. – Cham: Springer, 2016. – С. 167–197.
13. Burke R. Hybrid Recommender Systems: Survey and Experiments / Robin Burke. // User Modeling and User-Adapted Interaction. – 2002. – №12. – С. 331–370.
14. Feng D. Utility-based Recommender Systems Using Implicit Utility and Genetic Algorithm / Deng Feng // Proceedings of the International Conference on Mechatronics, Electronic, Industrial and Control Engineering (MEIC 2015) / Deng Feng. – Shenyang, 2015.
15. Sarin R. Multi-attribute Utility Theory / Rakesh Sarin // Encyclopedia of Operations Research and Management Science / Rakesh Sarin. – Boston: Springer, 2013.
16. Князева А. А. СПОСОБЫ ПОСТРОЕНИЯ ГИБРИДНОЙ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ НА ОСНОВЕ ДАННЫХ О ЗАКАЗАХ БИБЛИОТЕКИ / А. А. Князева, О. С. Колобов, И. Ю.

Турчановчкий // РАСПРЕДЕЛЕННЫЕ ИНФОРМАЦИОННО-ВЫЧИСЛИТЕЛЬНЫЕ РЕСУРСЫ. ЦИФРОВЫЕ ДВОЙНИКИ И БОЛЬШИЕ ДАННЫЕ. (DICR-2019) / А. А. Князева, О. С. Колобов, И. Ю. Турчановчкий. – Новосибирск: Институт вычислительных технологий Сибирского отделения РАН, 2019. – С. 96–101.

17. Rouse M. write once, run anywhere (WORA) [Електронний ресурс] / Margaret Rouse // whatis.com. – 2013. – Режим доступу до ресурсу: https://whatis.techtarget.com/definition/write-once-run-anywhere-WORA.
18. DATAFLAIR TEAM. Pros and Cons of Java | Advantages and Disadvantages of Java [Електронний ресурс] / DATAFLAIR TEAM // DataFlair. – 2019. – Режим доступу до ресурсу: https://data-flair.training/blogs/pros-and-cons-of-java/.
19. The Good and the Bad of Java Programming [Електронний ресурс] // alefsoft. – 2018. – Режим доступу до ресурсу: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-java-programming/
20. How SQLite Is Tested [Електронний ресурс] // sqlite.org – Режим доступу до ресурсу: https://www.sqlite.org/testing.html.
21. SQL As Understood By SQLite: ALTER TABLE [Електронний ресурс] // sqlite.org – Режим доступу до ресурсу: https://www.sqlite.org/lang_altertable.html.
22. About Databases and Performances [Електронний ресурс] // MSNoise – Режим доступу до ресурсу: http://msnoise.org/doc/about_db_performances.html.
23. Datatypes In SQLite Version 3 [Електронний ресурс] // sqlite.org – Режим доступу до ресурсу: https://www.sqlite.org/datatype3.html.
24. POM Reference [Електронний ресурс] // Apache Maven Project – Режим доступу до ресурсу: https://maven.apache.org/pom.html.

25. Zak H. Ant vs Maven vs Gradle [Електронний ресурс] / Zak H // linuxhint. – 2018. – Режим доступу до ресурсу: https://linuxhint.com/ant-vs-maven-vs-gradle/.
26. SQLite Create Table [Електронний ресурс] // SQLite Tutorial – Режим доступу до ресурсу: https://www.sqlitetutorial.net/sqlite-create-table/.
27. SQLite Insert [Електронний ресурс] // SQLite Tutorial – Режим доступу до ресурсу: https://www.sqlitetutorial.net/sqlite-insert/
28. SQLite Select [Електронний ресурс] // SQLite Tutorial – Режим доступу до ресурсу: https://www.sqlitetutorial.net/sqlite-select/
29. Ржеуський А. Рекомендаційна система інформаційного обслуговування користувачів бібліотек / А. Ржеуський, Н. Кунанець, М. Стахів // Матеріали V науково-технічної конференції «Інформаційні моделі, системи та технології», 1-2 лютого 2018 року. — Т. : ТНТУ, 2018. — С. 37. — (Секція 2. Інформаційні системи).
30. Коточигов К. Анатомия рекомендательных систем. Часть первая [Електронний ресурс] / Константин Коточигов // ГК ЛАНИТ. – 2018. – Режим доступу до ресурсу: https://habr.com/ru/company/lanit/blog/420499/.
31. Гринчуцька С. В. Конспект лекцій з курсу «Системи прийняття рішень» для студентів напряму підготовки 6.030502 «Економічна кібернетика» , спеціальності 051 «Економіка» / Світлана Вікторівна Гринчуцька. – Тернопіль: ТНТУ імені І. Пулюя, 2017. – 130 с.

ДОДАТКИ

Додаток А

Скрипти для створення таблиць бази даних

```
CREATE TABLE IF NOT EXISTS `Subjects` (
    `subject_id`  INTEGER NOT NULL PRIMARY KEY
AUTOINCREMENT,
    `subject_name`    TEXT NOT NULL,
    `is_subject_to_choose` INTEGER NOT NULL CHECK
(`is_subject_to_choose` IN (0, 1))
);
CREATE TABLE IF NOT EXISTS `Students` (
    `student_id`  INTEGER NOT NULL,
    PRIMARY KEY(`student_id`)
);
CREATE TABLE IF NOT EXISTS `Marks` (
    `student_id`  INTEGER,
    `subject_id`  INTEGER,
    `mark`   INTEGER NOT NULL CHECK (`mark` BETWEEN 60 and
100),
    FOREIGN KEY(`subject_id`) REFERENCES
`Subjects`(`subject_id`),
    PRIMARY KEY(`student_id`,`subject_id`),
    FOREIGN KEY(`student_id`) REFERENCES
`Students`(`student_id`)
);
```

Запити, які використовуються для отримання інформації з бази даних

Запит для отримання списку навчальних дисциплін. У якості аргумента передається 0 для отримання обов'язкових дисциплін, або 1 для отримання дисциплін за вибором.

```
select * from subjects where is_subject_to_choose = ?
```

Запит для отримання інформації про оцінки студентів.

```
select * from subjects natural join marks
```

Додаток Б

Повний код програмної реалізації додатку

```
package mishchenko;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
import mishchenko.recommendedsystem.RecommendedSystem;

public class Main extends Application {
    private static RecommendedSystem recommendedSystem;

    public static RecommendedSystem getRecommendedSystem() {
        return recommendedSystem;
    }

    @Override
    public void start(Stage primaryStage) throws Exception{
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(getClass().getResource("/mainWindow.fxml"));
        Parent root = loader.load();
        primaryStage.setTitle("Recommended system");
        primaryStage.setScene(new Scene(root, 450, 575));
        primaryStage.setResizable(false);
        primaryStage.show();
        recommendedSystem = new RecommendedSystem();
        recommendedSystem.start();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

```
package mishchenko.controller;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Alert;
import javafx.scene.control.TextField;
import javafx.scene.text.Text;
import mishchenko.Main;
import mishchenko.model.PossibleSubject;
import mishchenko.model.Student;
import mishchenko.recommendedsystem.RecommendedSystem;
import java.net.URL;
import java.util.LinkedList;
import java.util.List;
import java.util.ResourceBundle;

public class Controller implements Initializable {
    @FXML
    Text recommendationText;

    @FXML
    TextField higherMathematicsField;

    @FXML
    TextField discreteMathField;

    @FXML
    TextField englishField;

    @FXML
    TextField computerEquipmentMaintenanceField;

    @FXML
    TextField physicalTrainingField;

    @FXML
    TextField programmingField;

    @FXML
    TextField ukrainianStudiesField;

    @FXML
    TextField electronicInformationField;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
    }

    public void buttonAction(ActionEvent actionEvent) {
        try {
            List<Double> marks = new LinkedList<>();
            marks.add(getInput(higherMathematicsField.getText().trim()));
            marks.add(getInput(discreteMathField.getText().trim()));
            marks.add(getInput(englishField.getText().trim()));
            marks.add(getInput(computerEquipmentMaintenanceField.getText().trim()));
            marks.add(getInput(physicalTrainingField.getText().trim()));
            marks.add(getInput(programmingField.getText().trim()));
            marks.add(getInput(ukrainianStudiesField.getText().trim()));
            marks.add(getInput(electronicInformationField.getText().trim()));
```

```
            Student student = new Student();
            for (int i = 0; i < marks.size(); i++) {
                student.getMarks().put(RecommendedSystem.getSubjects().get(i),
marks.get(i));
            }
            System.out.println(student);
            List<PossibleSubject> recommendations =
Main.getRecommendedSystem().makeRecommendations(student);
            StringBuilder result = new StringBuilder();
            for (PossibleSubject possibleSubject: recommendations) {
                result.append("Recommendation for you is ")
                        .append(possibleSubject.getName())
                        .append(". Weighted arithmetic mean of marks for this subject
for you is ")
                        .append(possibleSubject.getResult())
                        .append(".\n");
            }
            recommendationText.setText(result.toString());
        } catch (InvalidInputException e) {
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Error!");
            alert.setHeaderText("Check input!");
            alert.setContentText(e.getMessage());
            alert.showAndWait();
        }
    }

    private double getInput(String str) throws InvalidInputException {
        if (str == null) {
            throw new InvalidInputException("One of the fields is empty");
        }

        double input;
        try {
            input = Double.parseDouble(str);
        } catch (NumberFormatException | NullPointerException e) {
            throw new InvalidInputException("One of the fields is filled
incorrectly!");
        }

        if ((input < RecommendedSystem.getMinMark() || input >
RecommendedSystem.getMaxMark())) {
            throw new InvalidInputException("One of the fields is filled
incorrectly!");
        }

        return input;
    }
}
```

```
package mishchenko.controller;

public class InvalidInputException extends Exception {
    public InvalidInputException(String message) {
        super(message);
    }
}
```

```
package mishchenko.db;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBUtils {
    private static Connection connection;
    private static final String PATH_TO_DB =
"jdbc:sqlite::resource:RecommendedSystemDB.db";
    public static synchronized Connection getConnection() {
        if (connection == null) {
            try {
                Class.forName("org.sqlite.JDBC");
                connection = DriverManager.getConnection(PATH_TO_DB);
            } catch (SQLException | ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
        return connection;
    }
}
```

```
package mishchenko.db;

import mishchenko.model.Student;
import java.util.List;

public interface RecommendedSystemDAO {
    List<String> getSubjects(int type);
    List<Student> getStudents();
}
```

```
package mishchenko.db;

import mishchenko.model.Student;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.LinkedList;
import java.util.List;

public class RecommendedSystemDAOImpl implements RecommendedSystemDAO {
    private static final String GET_SUBJECTS = "select * from subjects where
is_subject_to_choose = ?";
    private static final String GET_STUDENTS = "select * from subjects natural join
marks";

    @Override
    public List<String> getSubjects(int type) {
        Connection connection = DBUtils.getConnection();
        List<String> subjects = new LinkedList<>();
        try {
            PreparedStatement statement = connection.prepareStatement(GET_SUBJECTS);
            statement.setInt(1, type);
            ResultSet resultSet = statement.executeQuery();
            while (resultSet.next()) {
```

```
                System.out.println(resultSet.getString("subject_name"));
                subjects.add(resultSet.getString("subject_name"));
            }
            statement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return subjects;
    }

    @Override
    public List<Student> getStudents() {
        Connection connection = DBUtils.getConnection();
        List<Student> students = new LinkedList<>();
        try {
            PreparedStatement statement = connection.prepareStatement(GET_STUDENTS);
            ResultSet resultSet = statement.executeQuery();
            while (resultSet.next()) {
                int id = resultSet.getInt("student_id");
                Student student = students.stream()
                        .filter(s -> id == s.getId())
                        .findAny()
                        .orElse(null);
                if (student == null) {
                    student = new Student(id);
                    students.add(student);
                }
                if (resultSet.getInt("is_subject_to_choose") == 1) {
                    student.setChosenSubject(resultSet.getString("subject_name"),
resultSet.getDouble("mark"));
                } else {
                    student.getMarks().put(resultSet.getString("subject_name"),
resultSet.getDouble("mark"));
                }
            }
            System.out.println("//////////////");
            for (Student s: students) {
                System.out.println(s);
            }
            System.out.println("//////////////");
            statement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return students;
    }
}


package mishchenko.model;

public class PossibleSubject {
    private String name;
    private double sumOfMarks;
    private double sumOfSimilarities;

    public PossibleSubject(String name) {
        this.name = name;
```

```
        this.sumOfMarks = 0;
        this.sumOfSimilarities = 0;
    }

    public String getName() {
        return name;
    }

    public void setSumOfMarks(double sumOfMarks) {
        this.sumOfMarks = sumOfMarks;
    }

    public void setSumOfSimilarities(double sumOfSimilarities) {
        this.sumOfSimilarities = sumOfSimilarities;
    }

    public double getResult() {
        return sumOfSimilarities == 0 ? 0 : sumOfMarks / sumOfSimilarities;
    }

    @Override
    public String toString() {
        return "PossibleSubject{" +
                "name='" + name + '\'' +
                ", sumOfMarks=" + sumOfMarks +
                ", sumOfSimilarities=" + sumOfSimilarities +
                ", result=" + getResult() +
                '}';
    }
}
```

```
package mishchenko.model;

import javafx.util.Pair;
import java.util.LinkedHashMap;
import java.util.Map;

public class Student {
    private int id;
    private Map<String, Double> marks;
    private Pair<String, Double> chosenSubject;
    private double similarity;

    public Student(int id) {
        this.id = id;
        this.marks = new LinkedHashMap<>();
    }

    public Student() {
        id = -1;
        this.marks = new LinkedHashMap<>();
    }

    public double getSimilarity() {
        return similarity;
    }

    public void setSimilarity(double similarity) {
        this.similarity = similarity;
```

```
    }

    public int getId() {
        return id;
    }

    public Map<String, Double> getMarks() {
        return marks;
    }

    public Pair<String, Double> getChosenSubject() {
        return chosenSubject;
    }

    public void setChosenSubject(String chosenSubject, Double mark) {
        this.chosenSubject = new Pair<>(chosenSubject, mark);
    }

    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", marks=" + marks +
                ", chosenSubject=" + chosenSubject +
                ", similarity=" + similarity +
                '}';
    }
}


package mishchenko.recommendedsystem;

import mishchenko.db.RecommendedSystemDAO;
import mishchenko.db.RecommendedSystemDAOImpl;
import mishchenko.model.PossibleSubject;
import mishchenko.model.Student;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import static java.util.Comparator.comparing;
import static java.util.stream.Collectors.toList;

public class RecommendedSystem {
    private static final int MIN_MARK = 60;
    private static final int MAX_MARK = 100;
    private static final int DIFF = MAX_MARK - MIN_MARK;
    private static final int LIMIT_OF_SIMILAR_STUDENTS = 5;
    private static final int LIMIT_OF_SUBJECTS_TO_CHOOSE = 1;
    private static List<Student> students;
    private static List<String> subjects;
    private static List<String> subjectsToChoose;

    public static int getMinMark() {
        return MIN_MARK;
    }

    public static int getMaxMark() {
        return MAX_MARK;
    }
```

```
    public static List<String> getSubjects() {
        return subjects;
    }

    private static double cosineSimilarity(Student a, Student b) {
        double sum = 0;
        double sumA = 0;
        double sumB = 0;
        for (String subject : subjects) {
            sum += (((a.getMarks().get(subject) - MIN_MARK) / DIFF) + 0.01) *
(((b.getMarks().get(subject) - MIN_MARK) / DIFF) + 0.01);
            sumA += (((a.getMarks().get(subject) - MIN_MARK) / DIFF) + 0.01) *
(((a.getMarks().get(subject) - MIN_MARK) / DIFF) + 0.01);
            sumB += (((b.getMarks().get(subject) - MIN_MARK) / DIFF) + 0.01) *
(((b.getMarks().get(subject) - MIN_MARK) / DIFF) + 0.01);
        }

        double res = Math.sqrt(sumA) * Math.sqrt(sumB);

        if (res == 0) {
            return 0;
        } else {
            return sum / res;
        }
    }

    public void start() {
        RecommendedSystemDAO recommendedSystemDAO = new RecommendedSystemDAOImpl();
        subjectsToChoose = recommendedSystemDAO.getSubjects(1);
        System.out.println(subjectsToChoose);
        subjects = recommendedSystemDAO.getSubjects(0);
        System.out.println(subjects);
        students = recommendedSystemDAO.getStudents();
    }

    public List<PossibleSubject> makeRecommendations(Student student) {
        List<PossibleSubject> possibleSubjects = new LinkedList<>();
        Map<String, List<Student>> studentsBySubject = new LinkedHashMap<>();
        for (String subject: subjectsToChoose) {
            studentsBySubject.putIfAbsent(subject, new LinkedList<>());
        }
        for (Student s : students) {
            s.setSimilarity(cosineSimilarity(student, s));
            System.out.println(cosineSimilarity(student, s));
            studentsBySubject.get(s.getChosenSubject().getKey()).add(s);
        }
        for (Map.Entry<String, List<Student>> entry : studentsBySubject.entrySet()) {
            System.out.println(entry.getKey() + " : " );
            for (Student s: entry.getValue()) {
                System.out.println(s.getChosenSubject() + " : " + s.getSimilarity());
            }
            System.out.println("**************************");
            List<Student> resultStudents = entry.getValue().stream()
                    .sorted(comparing(Student::getSimilarity,
comparing(Math::abs)).reversed())
                    .limit(LIMIT_OF_SIMILAR_STUDENTS)
                    .collect(toList());
            double sumOfSimilarities = 0;
            double sumOfMarks = 0;
            for (Student s: resultStudents) {
```

```
                sumOfMarks += s.getChosenSubject().getValue() * s.getSimilarity();
                sumOfSimilarities += s.getSimilarity();
                System.out.println(s.getChosenSubject() + " : " + s.getSimilarity());
            }
            System.out.println(sumOfSimilarities);
            System.out.println(sumOfMarks);
            PossibleSubject possibleSubject = new PossibleSubject(entry.getKey());
            possibleSubject.setSumOfMarks(sumOfMarks);
            possibleSubject.setSumOfSimilarities(sumOfSimilarities);
            possibleSubjects.add(possibleSubject);
        }
        List<PossibleSubject> resultSubjects = possibleSubjects.stream()
                .sorted(comparing(PossibleSubject::getResult,
comparing(Math::abs)).reversed())
                .limit(LIMIT_OF_SUBJECTS_TO_CHOOSE)
                .collect(toList());
        System.out.println(possibleSubjects);
        return resultSubjects;
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<?import java.lang.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.text.*?>
<Pane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-
Infinity" prefHeight="575.0" prefWidth="450.0" xmlns="http://javafx.com/javafx/8"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="mishchenko.controller.Controller">
    <children>
        <Text fx:id="higherMathematicsLabel" layoutX="38.0" layoutY="43.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Higher Mathematics"
wrappingWidth="162.662109375" />
        <Text fx:id="discreteMathLabel" layoutX="38.0" layoutY="79.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Discrete Math"
wrappingWidth="162.662109375" />
        <Text fx:id="ratingLabel" layoutX="38.0" layoutY="117.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="English" wrappingWidth="162.662109375" />
        <Text fx:id="ratingLabel1" layoutX="38.0" layoutY="229.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Computer Equipment Maintenance"
wrappingWidth="223.662109375" />
        <Text fx:id="itemLabel1" layoutX="38.0" layoutY="190.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Physical Training" wrappingWidth="162.662109375" />
        <Text fx:id="userLabel1" layoutX="38.0" layoutY="154.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="C/C+ " wrappingWidth="162.662109375" />
        <Text fx:id="itemLabel11" layoutX="39.0" layoutY="268.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Ukrainian Studies" wrappingWidth="162.662109375" />
        <Text fx:id="ratingLabel11" layoutX="39.0" layoutY="307.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Organization and Processing of
Electronic Information" wrappingWidth="223.662109375" />
        <TextField fx:id="higherMathematicsField" layoutX="265.0" layoutY="27.0"
prefHeight="25.0" prefWidth="144.0" />
        <TextField fx:id="discreteMathField" layoutX="265.0" layoutY="63.0"
prefHeight="25.0" prefWidth="144.0" />
        <TextField fx:id="englishField" layoutX="265.0" layoutY="101.0"
prefHeight="25.0" prefWidth="144.0" />
        <TextField fx:id="computerEquipmentMaintenanceField" layoutX="265.0"
layoutY="138.0" prefHeight="25.0" prefWidth="144.0" />
```

```
        <TextField fx:id="physicalTrainingField" layoutX="265.0" layoutY="174.0"
prefHeight="25.0" prefWidth="144.0" />
        <TextField fx:id="programmingField" layoutX="265.0" layoutY="212.0"
prefHeight="25.0" prefWidth="144.0" />
        <TextField fx:id="ukrainianStudiesField" layoutX="266.0" layoutY="252.0"
prefHeight="25.0" prefWidth="144.0" />
        <TextField fx:id="electronicInformationField" layoutX="266.0" layoutY="290.0"
prefHeight="25.0" prefWidth="144.0" />
        <Button layoutX="153.0" layoutY="376.0" mnemonicParsing="false"
onAction="#buttonAction" text="Make recommendations" />
        <Text fx:id="recommendationText" layoutX="85.0" layoutY="466.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="The results will appear here..."
textAlignment="CENTER" wrappingWidth="279.13671875" />
    </children>
</Pane>


<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>groupId</groupId>
    <artifactId>RecommendedSystemJavaFX</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc -->
        <dependency>
            <groupId>org.xerial</groupId>
            <artifactId>sqlite-jdbc</artifactId>
            <version>3.31.1</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.5.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>mishchenko.Main</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```