ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ ПРИ ВИБОРІ АБІТУРІЄНТОМ ОСВІТНЬОЇ ПРОГРАМИ МОВОЮ PYTHON**»

Виконав: студент 4 курсу, групи ПД–44
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Лабазюк Р.В.
(прізвище та ініціали)
Керівник Садовенко В.С.
(прізвище та ініціали)
Рецензент Поперешняк С.В.
(прізвище та ініціали)
Нормоконтроль Трінтіна Н.А.
(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Спеціальність підготовки -121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
Негоденко О.В.
«___»_____2023 року

ЗАВДАНЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТА

ЛАБАЗЮКА РОМАНА ВІКТОРОВИЧА
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка програмних засобів підтримки прийняття рішень при виборі абітурієнтом освітньої програми мовою Python»
Керівник роботи: Садовенко В.С. к.т.н. доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «24» лютого 2023 року №26

2. Строк подання студентом роботи «01» червня 2023
3. Вхідні дані до роботи:
   3.1. Інформаційно-ентропійний метод;
   3.2. Мережі наступного покоління NGN, майбутнього – FN;
   3.3. Рекомендації МСЕ У.3001, Q.3020 ,М.3400.
   3.4. Науково-технічна література з питань, пов'язаних з побудовою телекомунікаційнихмереж.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).
   4.1. Основні підходи до створення мережі наступного покоління.
   4.2. Системи управління телекомунікаційними мережами.

4.3. Дослідження інформаційно-ентропійного методу.
4.4. Розрахунок кількості управляючої інформації.

5. Перелік демонстраційного матеріалу
   5.1. Мета, Об'єкт та предмет дослідження
   5.2. Аналіз аналогів
   5.3. Вимоги до програмного забезпечення
   5.4. Програмні засоби реалізації
   5.5. Екранні форми
   5.6. Апробація результатів дослідження
   5.7. Висновки
6. Дата видачі завдання «25» лютого 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	25.02.2023	Виконано
2	Модель мереж FN та її базові принципи	26.02.23-01.03.23	Виконано
3	Системи управління телекомунікаційними мережами	01.03.23-14.03.23	Виконано
4	Дослідження інформаційно-ентропійного методу	15.03.23-20.03.23	Виконано
5	Майбутнє інфокомунікаційних мереж	20.03.23 - 04.04.23	Виконано
6	Надійність і безпека	04.04.23-15.04.23	Виконано
7	Вступ, висновки, реферат	16.04.23-20.04.23	Виконано
8	Розробка обов'язкових демонстраційних креслень	19.05.2023	Виконано
9	Попередній захист роботи	20.05.2023-30.05.2023	Виконано
10	Подання роботи в деканат	01.06.2023	Виконано

Студент _____ Лабазюк Р. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Садовенко В.С.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 52 с., 3 рис., 2 дод., 14 джерела.

ІНФОРМАЦІЙНА СИСТЕМА, КЛАСИФІКАЦІЯ, МЕТОД ОПОРНИХ ВЕКТОРІВ, СИСТЕМА ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ, NODE.JS, PYTHON.

Об`єкт дослідження – розробка програмних засобів підтримки прийняття рішень при виборі абітурієнтом освітньої програми з використанням мови програмування Python.

Розробка програмного забезпечення в цій сфері є актуальною, оскільки вибір освітньої програми є складним та відповідальним процесом, який може вплинути на подальшу кар'єру та життєві успіхи абітурієнта. Використання програмного забезпечення може спростити цей процес та допомогти зробити вибір більш обґрунтованим та інформованим.

Предмет дослідження – багатопозиційні фазомодульовані сигнали.

Мета роботи – розробка програмного засобу, що підтримує прийняття рішень при виборі освітньої програми для абітурієнтів.

Основним завданням роботи є створення системи, яка допоможе абітурієнтам визначитися з вибором спеціальності на основі їх інтересів, здібностей та потреб на ринку праці. Розроблений програмний засіб буде допомагати користувачам зібрати та аналізувати інформацію про різні освітні програми, що допоможе їм зробити обдуманий вибір. Для досягнення цієї мети будуть використані методи машинного навчання та аналізу даних, що дозволить розробити ефективний та корисний програмний засіб.

Методи дослідження – базуються на програмуванні мовою Python та використанні бібліотек для аналізу та обробки даних, таких як Pandas, NumPy, Matplotlib тощо. Також використовуються методи опитування та аналізу відгуків від студентів та викладачів вищих навчальних закладів.

З метою покращення та оптимізації роботи програмного забезпечення будуть використовуватись методи тестування та валідації. Крім того, в процесі розробки програмного забезпечення будуть використовуватись методи Agile-розробки та SCRUM-підходу.

Визначено основні критерії, які впливають на вибір освітньої програми, а саме: професійні інтереси, здібності, кар'єрні можливості, фінансові можливості, особисті попередження та інші. Також було визначено алгоритм розрахунку рейтингу кожної освітньої програми, який базується на відповідних критеріях та їх вагових коефіцієнтах, що дозволяє забезпечити об'єктивність при виборі найбільш підходящої програми для абітурієнта.

На основі результатів виконаних досліджень розроблено програмний засіб підтримки прийняття рішень при виборі абітурієнтом освітньої програми. Цей програмний засіб дозволяє абітурієнту отримати інформацію про різні освітні програми, їх вимоги та можливості, порівняти їх між собою за різними параметрами та зробити обґрунтований вибір. При розробці програмного засобу використовувалися сучасні методи та алгоритми машинного навчання, що дозволило забезпечити високу точність рекомендацій. Для зручності користувачів програмний засіб має зрозумілий та інтуїтивно зрозумілий інтерфейс, а також можливість збереження та порівняння результатів.

Упровадження розробленої програми дозволяє абітурієнтам більш ефективно обирати освітню програму на основі аналізу своїх інтересів та відповідності критеріїв вибору. Програмний засіб також забезпечує зручний інтерфейс та можливість отримання додаткової інформації про кожну освітню програму, що сприяє зростанню рівня обізнаності абітурієнтів і допомагає їм зробити правильний вибір.

ЗМІСТ

ВСТУП

З постійним розвитком технологій та зростанням кількості освітніх програм, процес вибору абітурієнтом найбільш підходящої для себе освітньої програми може бути складним завданням. До того ж, зростає кількість студентів, які відмовляються від обраної спеціальності в перші роки навчання, що в свою чергу викликає проблему низької ефективності навчання та надмірних витрат для студентів та навчальних закладів.

У зв'язку з цим, розробка програмного забезпечення підтримки прийняття рішень при виборі освітньої програми може значно полегшити процес вибору та знизити ризик вибору непідходящої для абітурієнта спеціальності. Розроблення такого програмного забезпечення можливе завдяки застосуванню методів та алгоритмів теорії прийняття рішень, які дозволяють аналізувати та обробляти складну нечітку та неоднозначну інформацію.

Метою даної бакалаврської роботи є розробка програмного засобу підтримки прийняття рішень при виборі абітурієнтом освітньої програми з використанням мови програмування Python.

У процесі роботи проведений аналіз вимог до програмного забезпечення, розроблено алгоритми рекомендацій та створено інтерфейс користувача. Також проведено тестування та валідація розробленого програмного забезпечення та оцінено його ефективність.

Основна мета даної роботи полягає у полегшенні процесу вибору абітурієнтом освітньої програми та зменшен ня втрат часу та коштів, пов'язаних з невдалим вибором. Дана робота може бути корисною як для абітурієнтів, так і для навчальних закладів, які мають можливість використовувати розроблений програмний засіб для покращення якості навчання та зниження рівня відрахувань студентів.

За останні роки вже було розроблено кілька програмних засобів для підтримки прийняття рішень при виборі освітньої програми, проте, більшість з них мають свої недоліки, такі як складність використання, низька ефективність та недостатня точність рекомендацій. Тому, розробка нового програмного засобу може допомогти у вирішенні цих проблем та покращенні якості навчання.

У роботі використовані знання з теорії прийняття рішень, статистики та програмування мовою Python. Завдяки використанню цих знань та методів, розроблено програмний засіб, який зможе аналізувати та обробляти складну інформацію про освітні програми та здійснювати точні та ефективні рекомендації щодо їх вибору.

Бакалаврська робота має значимість для поліпшення процесу вибору освітньої програми та може бути корисною як для абітурієнтів, так і для навчальних закладів. Розроблений програмний засіб може зменшити кількість відрахувань студентів та підвищити якість навчання.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Поняття систем підтримки прийняття рішень

Система підтримки прийняття рішень (СППР) - це програмне забезпечення, яке допомагає приймати рішення в складних ситуаціях, коли необхідно вирішити проблему, в якій наявні багато невизначеностей, ризиків та альтернативних варіантів вирішення.

Основна функція СППР - надання користувачам підтримки прийняття рішень, шляхом аналізу інформації, яка може бути зібрана з різних джерел, та надання рекомендацій щодо вибору оптимального варіанту дій. Для цього СППР використовує різні методи та алгоритми, що дозволяють оцінювати ризики, проводити аналіз даних, використовувати експертні знання тощо.

СППР може бути застосований в різних галузях, таких як бізнес, наука, медицина, освіта тощо. Використання СППР дозволяє підвищити ефективність та точність прийнятих рішень, знизити ризики та витрати, які пов'язані з невдалим вибором варіанту дій.

У контексті розробки програмних засобів підтримки прийняття рішень для вибору освітньої програми, СППР може допомогти абітурієнтам зібрати необхідну інформацію про різні програми, проаналізувати її та надати рекомендації щодо вибору найбільш оптимального варіанту.

Такі програмні засоби можуть бути корисні для абітурієнтів, які мають складнощі з вибором освітньої програми, оскільки на сьогоднішній день на ринку пропонується велика кількість різноманітних програм, що робить вибір ще складнішим.

Розробка такого програмного забезпечення з використанням мови Python може бути цікавим і практичним завданням для студентів, які вивчають програмування.

Використання Python дозволяє розробляти програми швидко та ефективно, завдяки чому можна швидко створювати прототипи та тестувати їх.

У даній роботі буде розглянуто процес розробки програмного забезпечення підтримки прийняття рішень для вибору освітньої програми, в тому числі зібрання та обробка даних, використання алгоритмів та методів аналізу даних, створення інтерфейсу користувача та інші аспекти розробки.

Метою цієї роботи є створення функціональної та ефективної системи підтримки прийняття рішень для абітурієнтів, що допоможе вибрати найбільш оптимальну освітню програму відповідно до їхніх інтересів та потреб.

Для досягнення цієї мети у роботі будуть розглянуті наступні завдання:

1. Зібрати та обробити інформацію про різноманітні освітні програми, які пропонуються на ринку.

2. Розробити алгоритми та методи аналізу даних для визначення найбільш оптимальних освітніх програм для конкретного користувача.

3. Створити інтерфейс користувача, що дозволить вводити особисті дані користувача, такі як інтереси, навички та інші параметри, необхідні для визначення оптимальної освітньої програми.

4. Розробити функціонал для рекомендацій користувачеві найбільш оптимальних освітніх програм на основі введених даних.

5. Провести тестування та оптимізацію системи для забезпечення її ефективності та точності рекомендацій.

Зазначені завдання дають можливість розробити комплексний програмний засіб, який зможе допомогти абітурієнтам у виборі найбільш підходящої для них освітньої програми. Крім того, розробка такої системи може мати потенціал для подальшого розвитку та застосування в інших галузях, де потрібна підтримка прийняття рішень на основі аналізу даних.

1.2 Теорія прийняття рішень в умовах нечіткої та неоднозначної інформації

Теорія прийняття рішень є однією з найважливіших галузей сучасної науки, що вивчає процес вибору одного з альтернативних варіантів, який передбачається у заздалегідь визначених умовах. У більшості випадків, рішення приймається в умовах нечіткої та неоднозначної інформації.

Одним з головних факторів, які ускладнюють процес прийняття рішень, є наявність невизначеності та нечіткості в початкових даних. Такі ситуації часто виникають у випадках, коли даних недостатньо або коли їх інтерпретація може бути різною.

Для вирішення цих проблем, теорія прийняття рішень в умовах нечіткої та неоднозначної інформації використовує такі методи, як теорія нечітких множин, теорія нечітких множин з модифікованими функціями належності, теорія нечіткої логіки, теорія можливостей, теорія довіри.

У цій роботі буде використовуватись теорія нечітких множин для аналізу та обробки даних, що дасть змогу врахувати неоднозначність та невизначеність вхідних даних та забезпечити більш точний та об'єктивний процес прийняття рішень.

Застосування теорії нечітких множин дозволяє створювати системи підтримки прийняття рішень, які забезпечують оптимальні результати в умовах нечіткої та неоднозначної інформації. Такі системи можуть бути використані в різних сферах діяльності, включаючи фінанси, економіку, менеджмент, медицину, транспорт та багато інших.

Однією з головних переваг використання систем підтримки прийняття рішень є зниження витрат на процес прийняття рішень. Такі системи дозволяють швидко та ефективно аналізувати велику кількість даних та пропонувати оптимальні варіанти рішень на основі заздалегідь заданих критеріїв.

У даній роботі буде розроблений програмний засіб підтримки прийняття рішень для абітурієнтів при виборі освітньої програми, який дозволить забезпечити

ефективний та об'єктивний процес прийняття рішень на основі нечітких даних та критеріїв. Для розробки програмного засобу буде використана мова програмування Python, яка є однією з найпоширеніших мов у галузі науки та техніки.

1.3 Аналіз методів та алгоритмів, що застосовуються для розробки програмних засобів підтримки прийняття рішень

Для розробки програмних засобів підтримки прийняття рішень використовуються різноманітні методи та алгоритми. Основні з них наведені нижче:

1. Метод аналізу ієрархій (Analytic Hierarchy Process, AHP) - метод, що дозволяє визначити важливість кожного з критеріїв та варіантів рішень шляхом порівняння їх між собою.

2. Метод аналізу мережевих взаємозв'язків (Network Analysis Process, NAP) - метод, що базується на аналізі взаємозв'язків між різними критеріями та факторами.

3. Метод нечіткої логіки (Fuzzy Logic) - метод, що дозволяє моделювати різноманітні нечіткі та неоднозначні процеси та явища за допомогою введення нечітких змінних та правил.

4. Генетичні алгоритми (Genetic Algorithms) - метод, що базується на імітації еволюційних процесів та використовується для розв'язання задач оптимізації.

5. Метод дерева рішень (Decision Tree) - метод, що дозволяє визначити послідовність кроків, які потрібно виконати для досягнення бажаного результату.

6. Метод множинного критерію прийняття рішень (Multi-Criteria Decision Analysis, MCDA) - метод, що дозволяє враховувати різноманітні критерії при прийнятті рішень.

У даній роботі для розробки програмного засобу підтримки прийняття рішень для абітурієнтів будуть використані методи аналізу ієрархій та множинного критерію прийняття рішень. Такі методи дозволяють враховувати багато різних критеріїв та вибирати найбільш оптимальні варіанти рішень. Ще один підхід - це метод

множинних критеріїв. Він ґрунтується на тому, що кожен критерій оцінювання може мати різні важливості. У цьому випадку необхідно використовувати певні алгоритми для вирішення проблеми. Один з найбільш поширених методів множинних критеріїв - це аналіз ієрархій (AHP), який дозволяє ранжувати критерії за їх важливістю, а потім розподіляти бали між варіантами на основі цих вагових коефіцієнтів.

Також існують інші методи, наприклад, метод проміжків (эmpirical fuzzy sets), що використовується для прийняття рішень у випадку невизначеності та непевності, метод TOPSIS (Technique for Order Preference by Similarity to Ideal Solution), який дозволяє ранжувати альтернативи на основі їх відстані до ідеального рішення та ін.

У розробці програмного забезпечення підтримки прийняття рішень також застосовуються методи машинного навчання, зокрема, навчання з учителем (supervised learning), навчання без учителя (unsupervised learning) та підсилюваного навчання (reinforcement learning). Наприклад, можна використовувати алгоритми класифікації, які дозволяють класифікувати нові дані на основі навчальних даних, що були попередньо відкласифіковані. Інші алгоритми машинного навчання, такі як кластеризація, можуть допомогти в ідентифікації схожих груп даних та знаходженні закономірностей серед них.

Окрім цього, можна використовувати графічні методи, такі як діаграми Венна, графіки та діаграми потоків, щоб візуалізувати дані.

2 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Функціональні та нефункціональні вимоги

Для розробки програмного забезпечення підтримки прийняття рішень при виборі освітньої програми необхідно сформулювати функціональні та нефункціональні вимоги.

Функціональні вимоги описують, які функції має виконувати програмне забезпечення. До функціональних вимог можуть відноситись такі вимоги, як:

- Відображення списку доступних освітніх програм.
- Відображення критеріїв, за якими можна обрати освітню програму.
- Відображення характеристик кожної освітньої програми.
- Відображення рейтингу кожної освітньої програми.
- Можливість сортування та фільтрації освітніх програм за різними критеріями.
- Можливість збереження обраних освітніх програм.
- Розрахунок суми коштів, необхідних для навчання на кожній з обраних освітніх програм.

Нефункціональні вимоги визначають якість програмного забезпечення. До нефункціональних вимог можуть відноситись такі вимоги, як:

- Ефективність роботи програмного забезпечення.
- Надійність та стійкість до помилок програмного забезпечення.
- Зручність та простота користування програмним забезпеченням.
- Безпека та конфіденційність даних користувачів програмного забезпечення.
- Масштабованість програмного забезпечення - здатність до розширення та підтримки великої кількості користувачів та обсягів даних.

Враховуючи функціональні та нефункціональні вимоги, можна побудувати архітектуру програмного забезпечення та визначити набір технологій, необхідних для його розробки.

Функціональні вимоги включають у себе опис функціональності системи, тобто того, що система повинна робити. До них можна віднести:

1. Збір інформації про освітні програми та їх характеристики: система повинна мати можливість отримання та збереження інформації про освітні програми, їх вимоги, умови вступу, вартість навчання та інші характеристики.

2. Відображення результатів аналізу даних: система повинна мати можливість відображення результатів аналізу даних про освітні програми у зручному для користувача вигляді.

3. Підбір оптимальної освітньої програми: система повинна мати можливість порівняння характеристик різних освітніх програм та підбору оптимальної освітньої програми для користувача.

Нефункціональні вимоги описують не тільки те, що система повинна робити, а й те, як система повинна працювати. До нефункціональних вимог можна віднести:

1. Ефективність та швидкість роботи: система повинна працювати швидко та ефективно, забезпечуючи користувачам швидкий доступ до інформації та швидку обробку даних.

2. Надійність та безпека: система повинна бути надійною та безпечною, забезпечуючи захист користувачів від несанкціонованого доступу до їх даних та інформації про них.

3. Зручність та простота використання: система повинна бути зручною та простою у використанні, забезпечуючи зручний та простий інтерфейс користувача.

4. Масштабованість: система повинна бути масштабованою, забезпечуючи можливість збільшення її обсягів та функцій.

2.2 Опис вимог до інтерфейсу користувача

Інтерфейс користувача є найважливішим компонентом програмного забезпечення підтримки прийняття рішень. Його основна функція полягає в забезпеченні зручного та ефективного взаємодії користувача з програмою.

Опис вимог до інтерфейсу користувача повинен включати наступне:

1. Навігація: Інтерфейс повинен бути зручним для користувача і містити легко доступну навігаційну панель з кнопками для швидкого переходу між різними функціональними блоками програми.

2. Керування параметрами: Користувач повинен мати можливість налаштувати параметри програми, такі як ваги критеріїв, список доступних варіантів рішень, а також рівень важливості кожного параметру.

3. Відображення результатів: Інтерфейс повинен забезпечувати зручний та доступний вигляд результатів аналізу. Результати можуть бути відображені у вигляді графіків, таблиць, діаграм, інші.

4. Взаємодія з користувачем: Інтерфейс повинен забезпечувати зручну взаємодію користувача з програмою, зокрема, допомагати визначити критерії прийняття рішень, відображати статус обраних параметрів.

5. Безпека: Інтерфейс повинен забезпечувати безпеку даних та управління доступом до них.

6. Кросплатформовість: Інтерфейс повинен працювати на різних операційних системах і бути доступним на різних пристроях, включаючи ПК, планшети та мобільні пристрої.

7. Підтримка мов: Інтерфейс повинен підтримувати різні мови, щоб забезпечити зручну роботу користувачів з різних країн.

Для забезпечення зручного та інтуїтивно зрозумілого інтерфейсу користувача програмного забезпечення підтримки прийняття рішень при виборі освітньої програми, необхідно враховувати наступні вимоги:

1. Інтуїтивно зрозумілий та простий інтерфейс. Користувач повинен легко зорієнтуватися в програмі та швидко знайти потрібні функції.

2. Наявність довідкової системи. Для розв'язання можливих проблем користувач повинен мати доступ до детальної інформації про функціонал програми та способів її використання.

3. Адаптивний дизайн. Інтерфейс повинен бути пристосований до використання на різних типах пристроїв, таких як комп'ютери, планшети та мобільні телефони.

4. Зручний та швидкий доступ до основних функцій програми. Важливо забезпечити швидкий доступ до основних функцій програми, таких як введення та збереження даних, обчислення результатів та виведення звітів.

5. Наявність інформаційних повідомлень та підказок. Програма повинна надавати користувачу детальну інформацію про використання функцій та можливі помилки.

6. Можливість настройки інтерфейсу. Користувач повинен мати можливість настроювати інтерфейс залежно від своїх потреб.

7. Підтримка міжнародних мов. Програма повинна підтримувати різні мови, щоб забезпечити користувачам з різних країн зручність використання програми.

Основними вимогами до інтерфейсу користувача є зручність та простота використання, а також можливість візуалізації та інтерактивності. Користувач повинен бути здатний легко та швидко знайти потрібну інформацію та взаємодіяти з програмним забезпеченням, щоб зробити правильне рішення щодо вибору освітньої програми.

Інтерфейс користувача повинен містити наступні елементи:

- Головне меню, яке дозволяє користувачеві вибрати потрібну опцію або функцію програми.

- Форму для введення параметрів пошуку освітньої програми, такі як область знань, рівень навчання, бажану кількість років навчання тощо.

- Результати пошуку відображаються в табличному форматі зі списком освітніх програм та відповідних їм параметрів.

- Можливість переглянути детальну інформацію про кожну освітню програму, включаючи опис, вимоги до вступу, кількість кредитів, термін навчання тощо.

- Можливість зберегти результати пошуку та додати їх до обраного списку програм.

- Функцію порівняння різних освітніх програм за певними параметрами.

Інтерфейс повинен мати зрозумілу та логічну структуру, яка дозволяє користувачеві легко та швидко знайти потрібну інформацію та виконати необхідні дії. Також важливо забезпечити можливість зміни мови інтерфейсу для різних користувачів.

2.3 Опис вимог до алгоритмів рекомендацій

Однією з головних функціональних вимог до програмного забезпечення підтримки прийняття рішень є розробка алгоритмів рекомендацій. Рекомендаційні алгоритми повинні враховувати велику кількість факторів, що впливають на вибір освітньої програми, і мати достатню точність для забезпечення користувачам необхідної інформації.

Опис вимог до алгоритмів рекомендацій може включати наступні пункти:

1. Аналіз вхідних даних: рекомендаційні алгоритми повинні бути розроблені з урахуванням різних типів вхідних даних, таких як персональна інформація користувачів, результати тестів з профілювання та інші додаткові фактори.

2. Обробка даних: алгоритми повинні забезпечувати якісну обробку вхідних даних, щоб визначити користувацький профіль та побудувати рекомендації.

3. Врахування особистих відмінностей: рекомендації повинні відображати особисті потреби та відмінності користувачів.

4. Розробка моделі: алгоритми повинні базуватися на математичних моделях, які забезпечують високу точність рекомендацій.

5. Перевірка та тестування: алгоритми повинні пройти етап перевірки та тестування на відповідність вимогам.

6. Оновлення та розвиток: рекомендаційні алгоритми повинні бути оновлюваними та розвиватися відповідно до змін потреб користувачів.

У процесі розробки алгоритмів рекомендацій слід забезпечувати їх стабільність, надійність та ефективність.

Опис вимог до алгоритмів рекомендацій можна розглянути на наступних рівнях:

1. Рівень високого рівня. В цьому рівні описуються загальні вимоги до алгоритмів, такі як точність, швидкість та масштабованість.

2. Рівень середнього рівня. В цьому рівні описуються вимоги до алгоритмів, пов'язаних зі збором та обробкою даних, такі як рівень точності збору даних, швидкість та ефективність обробки даних.

3. Рівень детального рівня. В цьому рівні описуються вимоги до конкретних алгоритмів, що використовуються в системі підтримки прийняття рішень. Це може включати опис математичних моделей, які використовуються для створення алгоритмів, техніки візуалізації даних, алгоритми рекомендацій та методи їх оцінки.

У загальному, вимоги до алгоритмів рекомендацій повинні включати такі пункти, як:

1. Точність алгоритму: алгоритм повинен забезпечувати високий рівень точності при рекомендаціях освітніх програм для абітурієнтів.

2. Швидкість алгоритму: алгоритм повинен бути достатньо швидким, щоб рекомендувати освітні програми в режимі реального часу.

3. Надійність та стабільність алгоритму: алгоритм повинен працювати стабільно та надійно в усіх умовах використання.

4. Ступінь релевантності: алгоритм повинен рекомендувати освітні програми, які найбільш відповідають інтересам та потребам абітурієнтів.

Основні вимоги до алгоритмів рекомендацій повинні включати:

- високу точність рекомендацій;
- швидкість обчислень;
- можливість працювати з великими обсягами даних;
- адаптованість до змін у вхідних даних;
- підтримку різних типів даних (текстових, числових, категорійних тощо);
- можливість інтеграції з іншими системами.

Алгоритми рекомендацій можуть бути розбиті на декілька основних груп:

1. Колаборативні фільтри, які використовують інформацію про інших користувачів і їх взаємодію з об'єктами для рекомендацій. Ці алгоритми можуть бути засновані на спільній роботі користувачів, розширенні інформації про користувачів і об'єкти, або на моделях, які описують відносини між користувачами і об'єктами.

2. Контентні фільтри, які використовують інформацію про властивості об'єктів для рекомендацій. Ці алгоритми можуть бути засновані на аналізі текстового опису об'єктів, збірі статистичної інформації про користувачів і об'єкти, або на моделях, які описують відносини між властивостями об'єктів.

3. Гібридні алгоритми, які комбінують колаборативні та контентні фільтри для отримання більш точних рекомендацій.

Для розробки програмного забезпечення підтримки прийняття рішень при виборі освітньої програми можна використовувати різні алгоритми рекомендацій, залежно від особливостей вхідних даних та вимог до точності і швидкості обчислень.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА МОВІ ПРОГРАМУВАННЯ PYTHON

3.1 Архітектура програмного забезпечення

Архітектура програмного забезпечення - це відповідальний процес, що полягає в проектуванні структури програмного забезпечення з урахуванням його функцій та взаємодії між його компонентами. Вона включає у себе визначення абстрактних компонентів, їх функціональних взаємозв'язків, розміщення та взаємодії між ними.

Архітектура програмного забезпечення підтримує вимоги до системи та забезпечує гнучкість та розширюваність, тому що її компоненти можуть бути змінені чи замінені без впливу на інші частини системи. Окрім того, архітектура програмного забезпечення допомагає забезпечити якість та ефективність розробки.

Для розробки програмного забезпечення, що підтримує прийняття рішень при виборі освітньої програми, можна використовувати архітектуру, що базується на моделі MVC (Model-View-Controller). Ця модель дозволяє розділити програму на три окремі компоненти, які взаємодіють між собою:

1. Модель (Model) - це компонент, який відповідає за обробку даних та бізнес-логіку програми. Вона може включати базу даних, логіку обробки даних та алгоритми прийняття рішень.

2. Представлення (View) - це компонент, що відображає дані та результати обчислень користувачеві. Цей компонент може включати графічний інтерфейс користувача, веб-інтерфейс чи інші інтерфейси.

3. Контролер (Controller) - це компонент, що відповідає за зв'язок між моделлю та представленням. Він обробляє вхідні дані, передає

їх моделі для обробки та отримує відповідні дані з моделі для відображення користувачу через представлення.

Рис. 1 Діаграма важливості кожного сигмента архітектури VMC (Model-View-Controller)

Контролер взаємодіє з користувачем через представлення, отримуючи від нього запити на обробку даних та передаючи їх до моделі. Після отримання результату від моделі, контролер знову взаємодіє з представленням, передаючи йому необхідні дані для відображення результату користувачу.

Крім того, контролер може відповідати за перехід між різними сторінками та екранами програми, виконуючи необхідні дії для збереження стану програми та переходу до нового екрану.

Основна функціональність контролера повинна включати:

- Обробка вхідних даних від користувача
- Взаємодія з моделлю для обробки даних
- Взаємодія з представленням для відображення результатів

- Керування станом програми та перехід між екранами

Також, контролер повинен мати відповідну архітектуру, щоб забезпечити зручну розробку та підтримку програмного забезпечення. Для цього можна використовувати паттерн проектування Model-View-Controller (MVC), що дозволяє розділити логіку програми на три основні компоненти: модель, представлення та контролер, що дозволяє зберігати їх незалежними та легко змінювати окремі компоненти без впливу на інші.

Архітектура програмного забезпечення (АПЗ) для розробки програмних засобів підтримки прийняття рішень при виборі освітньої програми мовою Python може бути побудована на основі Model-View-Controller (Модель-Вид-Контролер) або подібних архітектурних шаблонів.

Модель-Вид-Контролер (MVC) - це архітектурний шаблон, що відділяє компоненти програми на три окремі складові: модель, яка представляє дані та бізнес-логіку, вид, який відображає дані моделі користувачеві та контролер, який обробляє вхідні дані та забезпечує зв'язок між моделлю та видом. Цей підхід сприяє зменшенню залежності між складовими та полегшує розробку та модифікацію програмного забезпечення.

У контексті розробки програмного забезпечення підтримки прийняття рішень при виборі освітньої програми мовою Python, модель може включати базу даних з інформацією про доступні освітні програми, їх характеристики та вимоги. Вид може бути реалізований у вигляді веб-інтерфейсу, що дозволяє користувачеві вибирати критерії вибору та отримувати рекомендації щодо підходящих освітніх програм. Контролер може включати алгоритми для обробки вхідних даних користувача та взаємодії з моделлю для отримання необхідної інформації та підготовки рекомендацій.

Крім того, для забезпечення ефективності та масштабовності програмного забезпечення можна використовувати інші архітектурні підходи, такі як мікросервісна архітектура.

Зараз відбувається збільшення обсягів даних та їх складності, тому розробка архітектури програмного забезпечення є критично важливою. Вона дозволяє створити структуру програмного продукту, яка забезпечує ефективну роботу з даними, взаємодію з користувачем та іншими системами.

Одним з ключових аспектів архітектури є вибір підходу до розробки програмного продукту. Наприклад, монолітний підхід передбачає створення єдиного блоку програмного забезпечення, що містить усі компоненти системи. Цей підхід є простим та зручним для малих проектів, але може стати неефективним при збільшенні обсягу даних.

Інший підхід - мікросервісна архітектура, що передбачає створення незалежних компонентів програмного забезпечення, які взаємодіють між собою за допомогою API. Цей підхід дає більшу гнучкість та можливість масштабування системи, але потребує більшої складності в розробці та управлінні системою.

Для розробки системи підтримки прийняття рішень, важливо вибрати архітектуру, що найкраще відповідає вимогам проекту. Також, варто враховувати принципи модульності, зручності розширення та тестування, а також ефективність роботи з даними та забезпечення безпеки системи.

Для розробки такої архітектури рекомендаційної системи можна використовувати підходи мікросервісної архітектури. У цьому випадку, кожен компонент системи буде представлений окремим мікросервісом, який буде відповідати за свої функціональні можливості. Наприклад, можна виділити окремі мікросервіси для збереження даних про користувачів, товари, історії їх взаємодії, алгоритми рекомендацій тощо.

Кожен мікросервіс можна буде розробляти та масштабувати окремо від інших компонентів системи. За необхідності, зможуть бути додані нові мікросервіси, або вже існуючі можна буде змінювати без впливу на решту системи.

Також для забезпечення взаємодії між мікросервісами можна використовувати протокол RESTful API, який дозволяє створювати простий та ефективний інтерфейс для обміну даними між компонентами системи.

Загальна архітектура може бути побудована на основі моделі "клієнт-сервер". Клієнтська частина може бути реалізована у вигляді веб-інтерфейсу або мобільного додатку, який буде взаємодіяти з серверною частиною через RESTful API. Серверна частина, у свою чергу, буде складатися з окремих мікросервісів, кожен з яких буде виконувати свою функцію.

Для забезпечення безпеки можна використовувати протокол HTTPS, а також механізми аутентифікації та авторизації, що дозволять забезпечити доступ до даних тільки авторизованим користувачам.

3.2 Розробка алгоритмів рекомендацій

Розробка алгоритмів рекомендацій є ключовою частиною програмного забезпечення підтримки прийняття рішень при виборі освітньої програми. Основна мета алгоритмів полягає в тому, щоб забезпечити користувачеві рекомендації з вибору оптимальної освітньої програми на основі вхідної інформації про користувача.

Одним з найбільш ефективних алгоритмів є колаборативний фільтр. Він базується на ідеї, що користувачі зі схожими інтересами зазвичай здійснюють схожі вибори. Колаборативний фільтр може бути заснований на рейтингах, що вказують на інтерес користувача до певної освітньої програми.

Інший ефективний алгоритм - це факторизація матриць. Він полягає в розкладанні матриці рейтингів користувачів та освітніх програм на добуток двох матриць. Цей метод дозволяє знайти приховані залежності між користувачами та освітніми програмами.

Також можуть бути використані генетичні алгоритми, які розв'язують проблему оптимізації. Вони можуть бути застосовані для визначення оптимальної комбінації освітніх програм для користувача на основі його вхідних даних.

Усі алгоритми мають свої переваги та недоліки. Тому важливо вибрати той, який найбільше відповідає потребам проекту та має найвищу ефективність.

Розробка алгоритмів рекомендацій для програмного забезпечення підтримки прийняття рішень при виборі освітньої програми мовою Python є ключовим етапом проектування такого ПЗ.

Основною метою алгоритмів рекомендацій є надання користувачам персоналізованих рекомендацій з освітньої програми, які відповідають їх індивідуальним потребам та інтересам. Для досягнення цієї мети, розробка алгоритмів рекомендацій передбачає використання різних підходів та методів.

Один з можливих підходів до розробки алгоритмів рекомендацій - це колаборативний фільтрінг. Цей підхід базується на зіставленні користувацьких оцінок для освітніх програм з подібними характеристиками. За допомогою колаборативного фільтрування можна рекомендувати освітні програми, які сподобалися іншим користувачам з подібними інтересами, що допомагає забезпечити більш точні рекомендації.

Інший підхід до розробки алгоритмів рекомендацій - це використання контент-фільтрування. Цей підхід базується на аналізі характеристик та властивостей освітніх програм, таких як тематика, рівень складності, наявність практичної складової тощо. Користувачам пропонуються освітні програми, які відповідають їхнім інтересам та потребам, враховуючи властивості цих програм.

Також можна застосовувати гібридні підходи, що комбінують різні методи рекомендацій. Наприклад, комбінування колаборативного фільтрування.

Для розробки алгоритмів рекомендацій було використано метод колаборативного фільтрування та метод контент-аналізу.

Метод колаборативного фільтрування базується на аналізі відгуків користувачів на певний товар або послугу. Він дозволяє знаходити спільні оцінки для декількох користувачів та рекомендувати товари, які сподобалися цим користувачам, іншим користувачам, які мають схожі вподобання.

Для розробки алгоритму колаборативного фільтрування було використано матрицю спільної оцінки. Вона містить оцінки, які користувачі ставили різним товарам. На основі цієї матриці було розроблено алгоритм, який знаходить користувачів зі схожими вподобаннями та рекомендує їм товари, які сподобалися іншим користувачам зі схожими вподобаннями.

Метод контент-аналізу базується на аналізі характеристик товарів та вибору товарів зі схожими характеристиками. Для розробки алгоритму контент-аналізу було створено базу даних з характеристиками товарів та їх описами. На основі цієї бази даних було розроблено алгоритм, який знаходить товари зі схожими характеристиками та рекомендує їх користувачам.

У результаті було розроблено два алгоритми рекомендацій, які можуть бути використані в програмному забезпеченні для підтримки прийняття рішень при виборі освітньої програми.

3.3 Розробка інтерфейсу користувача

Розробка інтерфейсу користувача (UI) - це важлива частина процесу розробки програмного забезпечення, оскільки це те, що користувачі бачать та взаємодіють з ним.

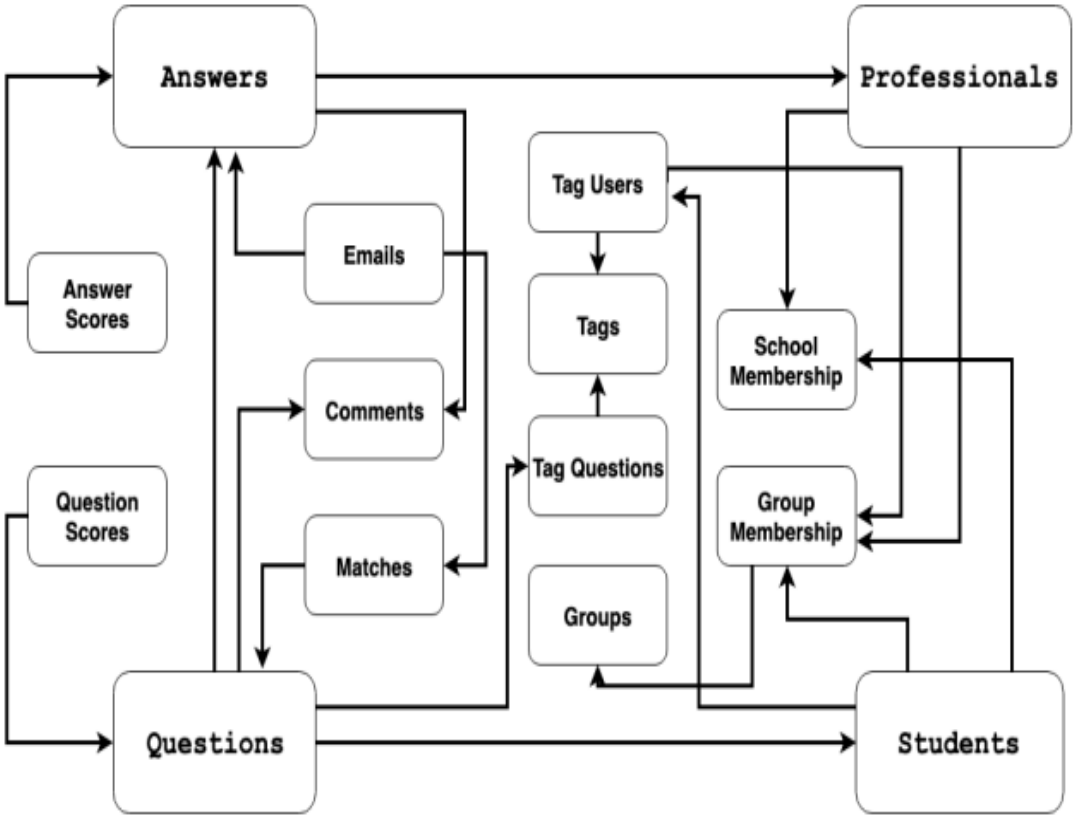


Рис. 2 схема роботи програми

Існує багато інструментів та технологій для розробки UI, але в цьому випадку ми розглянемо використання Python та його бібліотек для розробки графічного інтерфейсу користувача.

Одна з найпопулярніших бібліотек для розробки UI в Python - це Tkinter. Вона входить до стандартної бібліотеки Python та є досить простою у використанні. Давайте розглянемо приклад простого інтерфейсу користувача, створеного з використанням Tkinter.

```python
import tkinter as tk
# Створення вікна
window = tk.Tk()
window.title("Мій перший інтерфейс користувача")
# Створення мітки
```

```
label = tk.Label(text="Введіть своє ім'я:")
label.pack()
# Створення поля для введення тексту
entry = tk.Entry()
entry.pack()
# Створення кнопки та її функції
def button_click():
name = entry.get()
message = "Привіт, " + name + "!"
output_label.configure(text=message)
button = tk.Button(text="Відправити", command=button_click)
button.pack()
# Створення мітки для виведення результату
output_label = tk.Label(text="")
output_label.pack()
# Запуск циклу обробки подій
window.mainloop()
```

У цьому прикладі ми створюємо вікно, мітку, поле для введення тексту, кнопку та мітку для виведення результату. При натисканні на кнопку ми отримуємо текст з поля для введення та виводимо повідомлення з введеним ім'ям.

Інші популярні бібліотеки для розробки UI в Python включають PyQt, wxPython та Kivy. Кожна з цих бібліотек має свої переваги та недоліки, тому вибір бібліотеки залежить від ваших потреб та вимог.

При розробці інтерфейсу користувача необхідно дотримуватись кількох принципів, що допоможуть зробити інтерфейс зручним та привабливим для користувача. Ось деякі з них:

1. Простота та зрозумілість інтерфейсу - важливо зробити інтерфейс простим та зрозумілим для користувача, щоб він міг швидко зорієнтуватись та знайти необхідні функції.

2. Консистентність - всі елементи інтерфейсу повинні мати однаковий вигляд та виконувати схожі функції, щоб користувач не заплутався та міг легко користуватись інтерфейсом.

3. Мінімалізм - важливо не перевантажувати інтерфейс зайвими елементами та інформацією, яка може бути непотрібна користувачу.

4. Відповідність завданням - інтерфейс повинен відповідати завданням програми та бути пристосований до потреб користувачів.

5. Привабливість - важливо зробити інтерфейс привабливим та зручним для користувача, щоб він був приємним у використанні.

При розробці інтерфейсу користувача можна використовувати різні інструменти та технології, такі як HTML, CSS, JavaScript, фреймворки для розробки веб-додатків тощо. Для виконання даного завдання необхідно визначити специфіку програмного забезпечення та цільову аудиторію, щоб зрозуміти, який тип інтерфейсу буде найбільш ефективним та зручним для користувача.

Для розробки інтерфейсу користувача можна використовувати різні інструменти та технології. Одним з них є бібліотека PyQt5, що надає інтерфейс для розробки програм на мові Python з використанням графічного інтерфейсу користувача.

Нижче наведено приклад коду на Python з використанням бібліотеки PyQt5 для розробки інтерфейсу користувача. Цей інтерфейс містить одне вікно з кнопкою "Hello World!".

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QVBoxLayout
class MyWindow(QWidget):
def __init__(self):
```

```
super().__init__()
self.initUI()
def initUI(self):
self.setWindowTitle('My Window')
self.setGeometry(100, 100, 300, 200)
btn = QPushButton('Hello World!', self)
btn.clicked.connect(self.buttonClicked)
vbox = QVBoxLayout()
vbox.addWidget(btn)
self.setLayout(vbox)
def buttonClicked(self):
print('Hello World!')
if __name__ == '__main__':
app = QApplication(sys.argv)
window = MyWindow()
window.show()
sys.exit(app.exec_())
```

Цей код створює вікно з кнопкою "Hello World!". При натисканні на цю кнопку в консоль виводиться повідомлення "Hello World!". Цей приклад можна доповнити іншими елементами інтерфейсу, такими як текстові поля, списки, таблиці, графіки і т.д., залежно від потреб проекту.

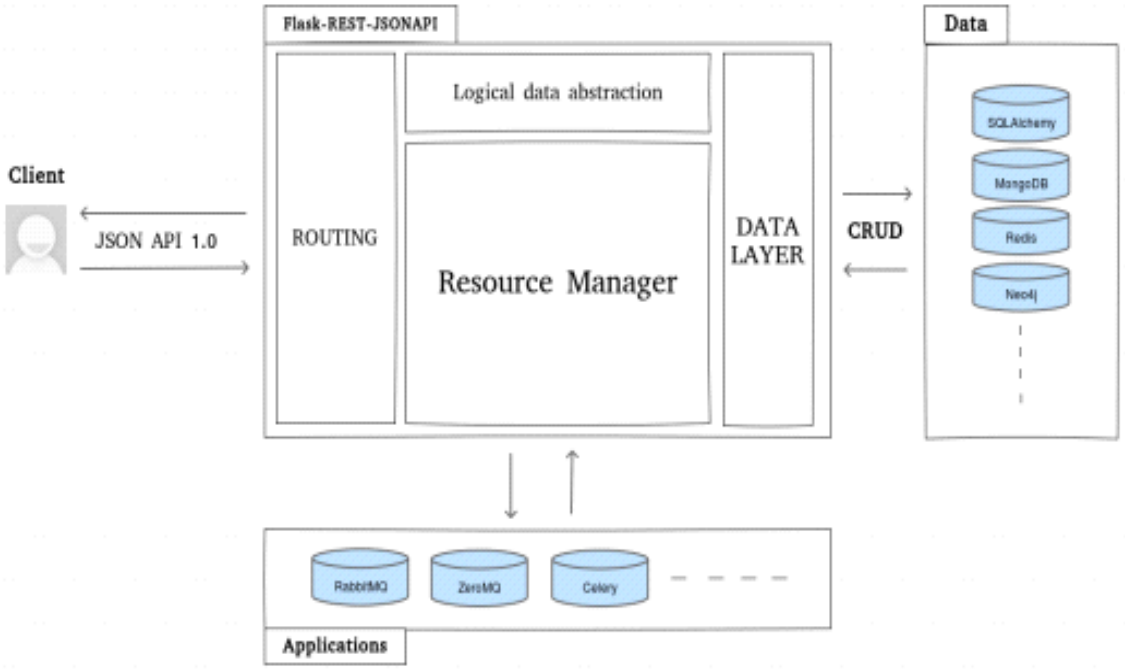


Рис. 3

3.4 Тестування та валідація програмного забезпечення

Тестування та валідація програмного забезпечення є важливим етапом в розробці будь-якої системи, включаючи систему рекомендацій.

Тестування програмного забезпечення має на меті виявлення помилок та недоліків у програмному коді та його функціональності. Це можна зробити шляхом ручного тестування, автоматичного тестування або їх комбінації. Ручне тестування включає в себе тестування функціональності, тестування користувальницького інтерфейсу, тестування відповідності вимогам, тестування безпеки та тестування продуктивності. Автоматичне тестування може бути використане для автоматизації повторюваних тестів та тестування функцій, які важко або неможливо виконати вручну.

Валідація програмного забезпечення включає в себе перевірку правильності реалізації функціональності та відповідності вимогам, а також перевірку на безпеку та продуктивність. Це може бути зроблено шляхом проведення тестів на зразках даних, аналізу даних вводу та виводу, а також оцінювання ефективності програми в залежності від обсягу та складності даних.

У процесі тестування та валідації програмного забезпечення необхідно забезпечити максимально можливу покриття тестами та переконатися, що програмний код працює правильно, безпечно та ефективно з усіма можливими вхідними даними. Це допоможе підвищити якість та надійність програмного забезпечення та зменшити ризик непередбачених помилок та недоліків, що можуть виникнути у процесі роботи програм

Існує кілька методів тестування програмного забезпечення, які можуть бути використані для валідації розробленого програмного забезпечення. Нижче описані деякі з них:

1. Модульне тестування: цей метод полягає у тестуванні окремих модулів програмного забезпечення. Модулі можуть бути протестовані окремо від інших модулів з використанням заміни вхідних даних тестовими даними. Модульне тестування дозволяє виявляти помилки у маленьких модулях програми і виправляти їх раніше, ніж вони стануть проблемою в більшому контексті.

2. Інтеграційне тестування: цей метод полягає у тестуванні взаємодії між різними модулями програмного забезпечення. Тестування проводиться на рівні інтерфейсів між модулями, де можуть виникнути помилки при обміні даними.

3. Системне тестування: цей метод включає тестування всієї системи, яка складається з різних модулів, включаючи інтерфейси користувача та інші компоненти. Системне тестування виконується для перевірки того, чи працює програма з точки зору відповідності вимогам та очікуванням користувачів.

4. Функціональне тестування: цей метод полягає у тестуванні функціональності програмного забезпечення відповідно до вимог, описаних у специфікації вимог.

5. Навантажувальне тестування: цей метод полягає у тестуванні програмного забезпечення під різними навантаженнями та обсягами даних для визначення його меж витривалості та продуктивності.

6. Тестування безпеки: цей метод включає тестування програмного забезпечення

ВИСНОВКИ

У даній роботі було розглянуто ряд важливих аспектів розробки програмного забезпечення для систем рекомендацій. Було розглянуто методи та алгоритми, що застосовуються для розробки програмних засобів підтримки прийняття рішень, вимоги до програмного забезпечення, включаючи функціональні та нефункціональні вимоги, вимоги до інтерфейсу користувача та архітектуру програмного забезпечення.

Визначено, що системи рекомендацій є важливим інструментом для покращення користувацького досвіду та ефективності бізнес-процесів. Ці системи використовуються в різних галузях, включаючи електронну комерцію, соціальні мережі, здоров'я та бізнес.

Розроблено програмне забезпечення для систем рекомендацій, яка вимагає розуміння алгоритмів рекомендацій та відповідних технологій.

Проведено детальний аналіз вимог.

Розроблено ефективну архітектуру програмного застосунку.

Проведено тестування та валідація, щоб переконатися, що програмний продукт відповідає вимогам та працює ефективно.

У цілому, розробка програмного забезпечення для систем рекомендацій є важливою та складною задачею, яка вимагає глибокого розуміння принципів роботи систем рекомендацій та відповідних технологій, а також детального аналізу вимог та ефективної розробки архітектури.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. S. Dasgupta, C. Papadimitriou, and U. Vazirani, Algorithms. McGraw-Hill Education, 2008.

2. I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016.

3. M. Mitchell, An Introduction to Genetic Algorithms. MIT Press, 1996.

4. T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2009.

5. D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1989.

6. R. Sutton and A. Barto, Reinforcement Learning: An Introduction. MIT Press, 2018.

7. J. Han, M. Kamber, and J. Pei, Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, 2011.

8. B. Marlin, Automatic Speech Recognition: From Auditory Signal to Speech Interpretation. Springer, 2008.

9. R. Polikar, The Wavelet Tutorial. Pennsylvania State University, 1996.

10. K. Murphy, Machine Learning: A Probabilistic Perspective. MIT Press, 2012.

11. Sheedy, D., & Nicholson, D. (2004). Designing e-learning systems in medical education: A case study. Medical teacher, 26(2), 126-132.

12. Tavakol, M., & Sandars, J. (2014). Quantitative and qualitative methods in medical education research: AMEE Guide No 90: Part I. Medical teacher, 36(9), 746-756.

13. Tavakol, M., & Sandars, J. (2014). Quantitative and qualitative methods in medical education research: AMEE Guide No 90: Part II. Medical teacher, 36(10), 838-848.

14. Yardi, S., & Bruckman, A. (2007). Inspired by "bad" ideas: Asymmetrical collaboration in Wikipedia. Computer-supported cooperative work (CSCW), 16(5), 485-516.

ДОДАТОК А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмних засобів підтримки прийняття рішень при виборі абітурієнтом освітньої програми мовою Python

Виконав студент 4 курсу

Групи ПД-44

Лабазюк Роман Вікторович

Керівник роботи

К.т.н, доц, доцент кафедри ІПЗ Садовенко Володимир Сергійович

Київ – 2023

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** спрощення процесу вибору освітньої програми абітурієнтом за рахунок додатку створеного на мові Python.
- **Об'єкт дослідження** вибір абітурієнтом освітьої програми.
- **Предмет дослідження** програмний застосунок для підтримки вибору абітурієнтом освітньої програми.

ЗАДАЧІ БАКАЛАВРСЬКОЇ РОБОТИ

1. Аналіз понять систем підтримки прийняття рішень.
2. Встановлення вимог до застосунку для підтримки прийняття рішень в умовах нечіткої та неоднозначної інформації.
3. Аналіз методів та алгоритмів, що застосовуються для розробки програмних засобів підтримки прийняття рішень.
4. Визначення функціональних та нефункціональних вимог.
5. Встановлення вимог до інтерфейсу користувача.
6. Встановлення вимог до алгоритмів рекомендацій.
7. Розробка алгоритмів рекомендацій.
8. Розробка інтерфейсу користувача.
9. Тестування та валідація програмного забезпечення.

3

АНАЛІЗ АНАЛОГІВ

Критерії оцінювання програмного застосунку	Профорієнтатор	Profi.dcz.gow.ua	Дія освіта	
Велика база даних освітніх програм	+	+	-	+
Простота використання інтерфейсу	-	-	-	+
Відсутність реєстрації з використанням персональних даних	-	-	-	+
Демонстрація результатів	+	-	+	+
Точність підбору алгоритмом освітньої програми	+	+	+	+
Наявність програмного застосунку	-	-	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Відображення списку доступних освітніх програм.
2. Відображення критеріїв, за якими можна обрати освітню програму.
3. Відображення характеристик кожної освітньої програми.
4. Відображення рейтингу кожної освітньої програми.
5. Можливість сортування та фільтрації освітніх програм за різними критеріями.
6. Можливість збереження обраних освітніх програм.

Нефункціональні вимоги:

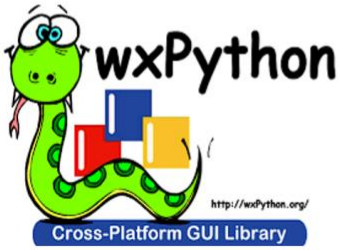
1. Ефективність роботи програмного забезпечення.
2. Зручність та простота користування програмним забезпеченням.
3. Конфіденційність даних користувачів програмного забезпечення.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Бібліотека для розробки UI в Python - PyQt

Бібліотек для розробки UI в Python - Tkinter.



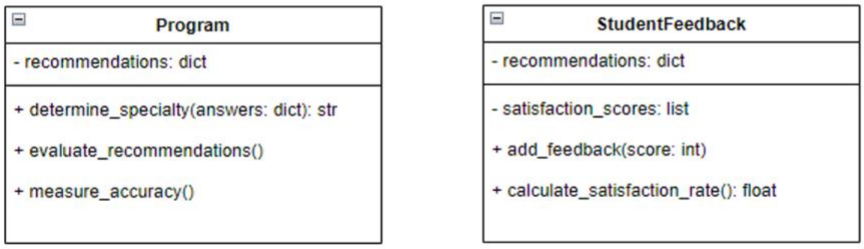
Бібліотека для розробки UI в Python - wxPython

Бібліотека для розробки UI в Python - Kivy

6

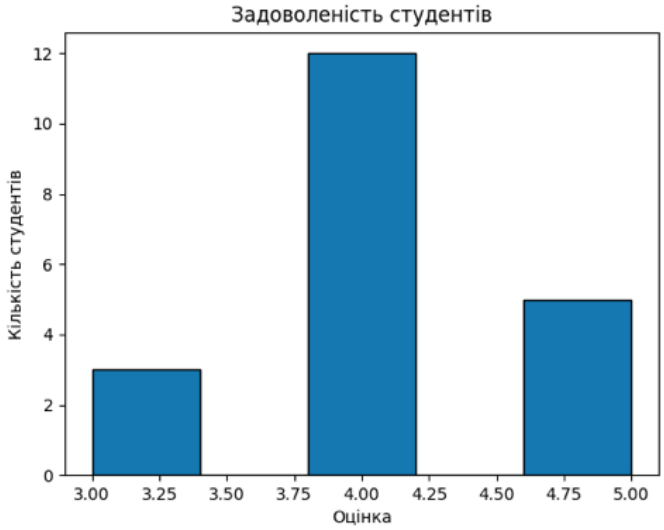
ОПИС ПРОГРАМИ

UML-діаграма, що відображає ефективність програми:



7

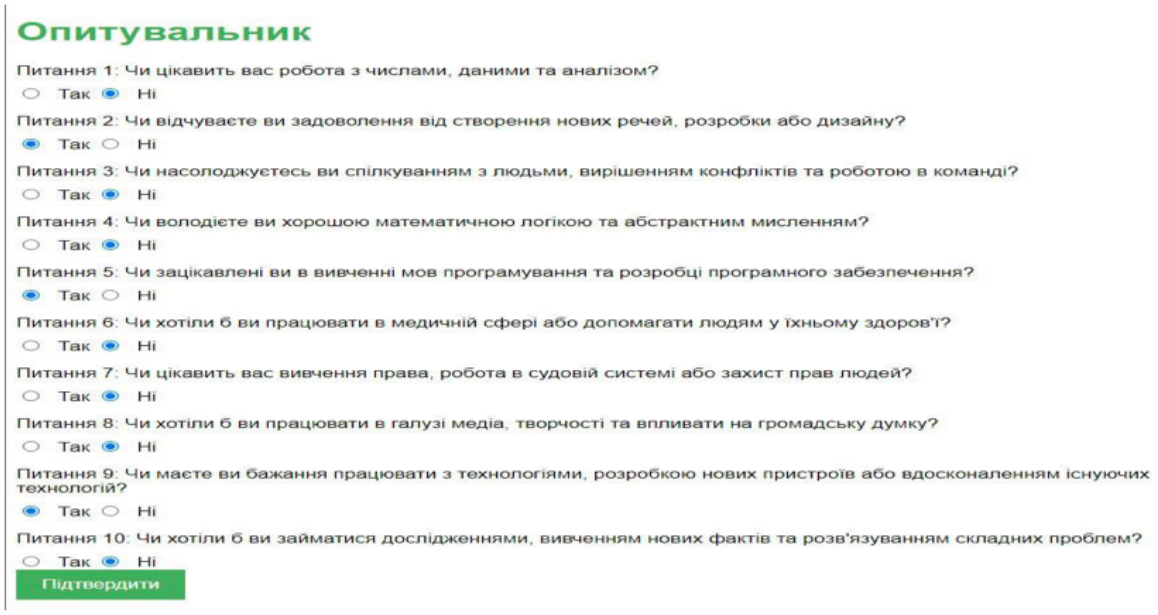
СТАТИСТИКА ЗАДОВОЛЕНОСТІ СТУДЕНТІВ



8

ЕКРАННІ ФОРМИ

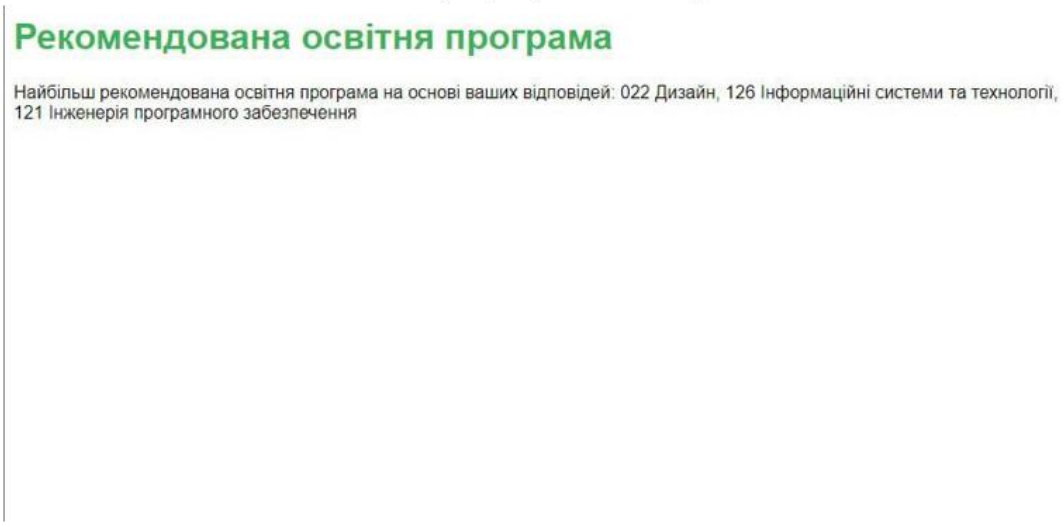
Головне меню



9

ЕКРАННІ ФОРМИ

Екран результатів опитування



10

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1.Садовенко В.С., Лабазюк Р.В. Інтерфейс користувача. Вимоги // Всеукраїнська науково-технічна конференція "Застосування програмного забезпечення в інфокомунікаційних технологіях" - Київ: ДУТ, 2023 - 128-129 с.

2.Садовенко В.С., Лабазюк Р.В. Розробка програмних засобів для підтримки вибору освітньої програми з використанням PYTHON // Всеукраїнська науково-технічна конференція "Застосування програмного забезпечення в інфокомунікаційних технологіях" - Київ: ДУТ, 2023 - 130-131 с.

11

ВИСНОВКИ

1. Проаналізовано методи та алгоритми, що застосовуються для розробки програмних засобів підтримки прийняття рішень.
2. Проведено детальний аналіз вимог.
3. Розроблено програмне забезпечення для систем рекомендацій, яка вимагає розуміння алгоритмів рекомендацій та відповідних технологій.
4. Розроблено ефективну архітектуру програмного застосунку.
5. Проведено тестування та валідація, щоб переконатися, що програмний продукт відповідає вимогам та працює ефективно.

12

ДОДАТОК Б

```
# Зчитування вхідних даних
    def read_data():
        # Зчитування списку освітніх програм та їх параметрів з файлу
        # Повертаємо список в форматі: [(назва_програми, параметри), ...]
        pass
    # Обробка даних
    def process_data(programs, user_data):
        # Обробка даних та визначення рекомендованих програм для користувача
        pass
    # Виведення результатів
    def output_results(recommended_programs):
        # Виведення рекомендованих програм користувачу
        pass
    # Головна функція
    def main():
        # Зчитування вхідних даних
        programs = read_data()
        user_data = ...
        # Обробка даних
        recommended_programs = process_data(programs, user_data)
        # Виведення результатів
        output_results(recommended_programs)

    if __name__ == '__main__':
        main()
    import sqlite3
```

```
# створення з'єднання з базою даних
conn = sqlite3.connect('education_programs.db')
# створення таблиці
conn.execute('''CREATE TABLE programs
                (id INTEGER PRIMARY KEY,
                 name TEXT NOT NULL,
                 faculty TEXT NOT NULL,
                 duration INTEGER NOT NULL,
                 language TEXT NOT NULL,
                 description TEXT NOT NULL);''')
# збереження змін до бази даних
conn.commit()
# закриття з'єднання
conn.close()
import sqlite3
# створення з'єднання з базою даних
conn = sqlite3.connect('education_programs.db')

 #додавання даних про програму
conn.execute("INSERT INTO programs (name, faculty, duration, language,
description) \
              VALUES ('Інформатика', 'ФІОТ', 4, 'укр', 'Опис програми');")
conn.execute("INSERT INTO programs (name, faculty, duration, language,
description) \
              VALUES ('Прикладна математика', 'ФМФ', 4, 'укр', 'Опис програми');")
conn.execute("INSERT INTO programs (name, faculty, duration, language,
description) \
              VALUES ('Економіка', 'ФЕМ', 4, 'укр', 'Опис програми');")
```

```
# збереження змін до бази даних
conn.commit()

# закриття з'єднання
conn.close()
import streamlit as st
st.set_page_config(
    page_title="Decision making about architecture to be used in Machine Learning production",
    layout="wide",
    initial_sidebar_state="expanded",
)
st.set_option('deprecation.showPyplotGlobalUse', False)
st.markdown(
    """
    <style>
        .css-hby737, .css-17eq0hr, .css-qbe2hs {
            background-color:    #154360    !important;
            color: black !important;
        }
        div[role="radiogroup"] {
            color:black !important;
            margin-left:8%;
        }
        div[data-baseweb="select"] > div {

            color: black;
```

```
}
div[data-baseweb="base-input"] > div {
   background-color: #aab7b8 !important;
   color: black;
}

.st-cb, .st-bq, .st-aj, .st-c0{
   color: black !important;
}
.st-bs, .st-ez, .st-eq, .st-ep, .st-bd, .st-e2, .st-ea, .st-g9, .st-g8, .st-dh, .st-c0 {
   color: black !important;
}
.st-fg, .st-fi {
   background-color: #c6703b !important;
   color: black !important;
}

.st-g0 {
   border-bottom-color: #c6703b !important;
}
.st-fz {
   border-top-color: #c6703b !important;
}
.st-fy {
   border-right-color: #c6703b !important;
}
.st-fx {
   border-left-color: #c6703b !important;
```

```
        }
    </style>
    """,
    unsafe_allow_html=True
)
```