

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Використання методів нечіткої логіки в системах тестування
знань»**

Виконав студент групи ПЗм-23-1
ЗАХАРЧУК Максим Сергійович

Керівник роботи:
ЧИЖМОТРИЯ Олексій Володимирович

Рецензент:
БАКАЛЮК Тетяна Анатоліївна

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«1» жовтня 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЗАХАРЧУК Максим Сергійович**

Керівник роботи: **ЧИЖМОТРЕ Олексій Володимирович**

Тема роботи: **«Використання методів нечіткої логіки в системах тестування знань»**, затверджена Наказом закладу вищої освіти від **«1» жовтня 2024 р.**, №520/с

Вихідні дані для роботи: **системи тестування знань, застосування методів нечіткої логіки для оцінювання результатів тестування.**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Керівник / Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О. В.	01.10.2024р.	01.10.2024р.
2	Чижмотря О. В.	01.10.2024р.	01.10.2024р.
3	Чижмотря О. В.	01.10.2024р.	01.10.2024р.

РЕФЕРАТ

Кваліфікаційна робота освітнього ступеня «магістр» складається з програмного комплексу онлайн сервісу системи тестування знань з використанням нечіткої логіки та пояснювальної записки. Пояснювальна записка до випускної роботи містить 88 сторінок, 24 ілюстрації та 5 таблиць.

Метою роботи є дослідження способів використання нечіткої логіки в тестуванні знань та розробки веб додатка з реалізацією даного функціоналу, що включає модель оцінювання на базі нечіткої логіки.

В роботі визначені головні завдання для розробки додатку, проаналізовано методи розв'язання поставлених задач, обґрунтовано вибір структури додатку, наведено алгоритм його роботи, а також архітектуру та взаємодію його основних компонентів. Також була описана програмна реалізація програми.

КЛЮЧОВІ СЛОВА: НЕЧІТКА ЛОГІКА, ТЕСТУВАННЯ ЗНАНЬ, ВЕБ-ДОДАТОК, PYTHON, ASP.NET.

ABSTRACT

The master's thesis consists of a software complex for an online knowledge testing system using fuzzy logic and an explanatory report. The explanatory report contains 88 pages, 24 illustrations, and 5 tables.

The purpose of the thesis is to explore methods of applying fuzzy logic in knowledge testing and to develop a web application implementing this functionality, including an evaluation model based on fuzzy logic.

The thesis defines the main tasks for application development, analyzes methods for solving the specified problems, justifies the chosen application structure, and presents the algorithm of its operation, as well as the architecture and interaction of its main components. The software implementation of the application is also described.

KEYWORDS: FUZZY LOGIC, KNOWLEDGE TESTING, WEB APPLICATION, PYTHON, ASP.NET.

ЗМІСТ

РЕФЕРАТ.....	3
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ ВИКОРИСТАННЯ НЕЧІТКОЇ ЛОГІКИ В СИСТЕМАХ ТЕСТУВАННЯ ЗНАНЬ.....	9
1.1 Поняття нечіткої логіки: основи теорії та практичні аспекти	9
1.2 Особливості застосування нечіткої логіки в інтелектуальних системах ..	12
1.3 Аналіз можливостей використання нечіткої логіки в системах тестування знань	14
1.4 Постановка задачі та вибір технологій для розробки	18
Висновки до першого розділу.....	21
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА МОДЕЛІ ТЕСТУВАННЯ ЗНАНЬ ІЗ ВИКОРИСТАННЯМ НЕЧІТКОЇ ЛОГІКИ.....	22
2.1 Розробка моделі оцінювання результатів тестування за допомогою нечіткої логіки	22
2.2 Проектування веб-орієнтованої системи тестування знань	27
2.3 Розробка веб-орієнтованої системи тестування знань з використанням методів нечіткої логіки	37
Висновки до другого розділу	45
РОЗДІЛ 3. ПОРЯДОК РОБОТИ З ДОДАТКОМ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ	46
3.1 Порядок роботи з додатком	46
3.2. Оцінка роботи системи та перспективи використання	53
Висновки до третього розділу.	57
ВИСНОВКИ	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	60
ДОДАТКИ.....	63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

JS – JavaScript (мова програмування)

БД – база даних

ПЗ – програмне забезпечення

UI – інтерфейс користувача

ПК – персональний комп'ютер

MVC – models views controllers (патерн проектування)

ВСТУП

Сучасний світ ставить перед суспільством нові виклики, що стосуються освіти та підвищення ефективності навчального процесу. Одним із ключових аспектів освітньої діяльності є об'єктивне оцінювання знань, умінь і навичок учнів, студентів та інших суб'єктів навчального процесу. Традиційні методи тестування нерідко мають обмеження, пов'язані з надмірною жорсткістю критеріїв оцінювання, що може не враховувати індивідуальні особливості тестованих осіб.

У цьому контексті актуальним стає використання нечіткої логіки, яка дозволяє моделювати процеси, що мають нечітко визначені чи розмиті критерії. Методологія нечіткої логіки дозволяє створювати інтелектуальні системи оцінювання, які здатні гнучко адаптуватися до різних ситуацій та надавати точніші й об'єктивніші результати.

Мета цієї роботи полягає у дослідженні можливостей застосування методів нечіткої логіки в системах тестування знань, розробці моделі оцінювання, заснованої на цих методах, та створенні програмного продукту, що дозволяє демонструвати її практичну ефективність.

У рамках роботи будуть вирішені наступні завдання:

- Проведено аналіз існуючих рішень із застосуванням методів нечіткої логіки в системах тестування.
- Розроблено модель оцінювання знань із використанням нечіткої логіки.
- Реалізовано програмний додаток для демонстрації розробленої моделі.
- Проведено тестування розробленої системи та аналіз її ефективності.

Об'єктом дослідження є системи тестування знань.

Предметом дослідження є застосування методів нечіткої логіки для оцінювання результатів тестування.

Наукова новизна роботи полягає у вдосконаленні підходів до оцінювання знань шляхом використання нечіткої логіки, що дозволяє враховувати суб'єктивні чинники та знижувати ймовірність упереджених оцінок.

Практичне значення роботи полягає в тому, що розроблена система може бути використана у навчальних закладах для підвищення якості оцінювання знань, а також як основа для подальших досліджень у цій галузі.

Апробація. За темою кваліфікаційної магістерської роботи було опубліковано три тези конференції [1, 2, 3].

РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ ВИКОРИСТАННЯ НЕЧІТКОЇ ЛОГІКИ В СИСТЕМАХ ТЕСТУВАННЯ ЗНАНЬ

1.1 Поняття нечіткої логіки: основи теорії та практичні аспекти

Нечітка логіка (англ. *Fuzzy Logic*) — це розділ математичної логіки, який розширює класичну логіку, вводячи поняття нечітких множин, тобто множин, елементи яких можуть належати до них із певним ступенем. Цей підхід був запропонований у 1965 році Лотфі Заде, американським математиком і вченим азербайджанського походження. Його теорія стала відповіддю на необхідність моделювання складних систем і процесів, для яких традиційні математичні методи були недостатньо ефективними [4].

На відміну від класичної логіки, де кожне твердження може бути або істинним, або хибним (тобто має значення 1 або 0), нечітка логіка дозволяє працювати з проміжними значеннями, що представляють ступінь істинності твердження [4]. Наприклад, твердження "температура тепла" у класичній логіці було б або істинним, або хибним, залежно від заздалегідь визначеного порогу. У нечіткій логіці ступінь тепла може варіюватися від 0 до 1, залежно від фактичної температури.

Нечітка логіка є одним із напрямів сучасної математики, що пропонує ефективні методи роботи з неточними, неоднозначними та розмитими даними. Її основними елементами є нечіткі множини, функції належності, лінгвістичні змінні та нечіткі правила. Кожен із цих компонентів має своє важливе значення у створенні та використанні систем, побудованих на принципах нечіткої логіки.

Поняття нечіткої множини є центральним у цій теорії. На відміну від класичних множин, де елемент або належить множині, або ні, у нечіткій множині кожен елемент має ступінь належності. Цей ступінь виражається числом у межах від нуля до одиниці. Наприклад, у множині "високі люди" людина з ростом 180 см може мати ступінь належності 0.8, що вказує на її високу відповідність цій множині, тоді як людина з ростом 170 см може мати значення 0.5, що свідчить

про помірний ступінь належності. Такий підхід дозволяє працювати з нечіткими, розмитими категоріями, які часто зустрічаються у реальному світі.

Функція належності визначає, яким саме чином розраховується ступінь належності елемента до певної множини. Залежно від контексту й особливостей задачі, ця функція може бути лінійною, сигмоїдною або мати іншу математичну форму. Наприклад, для визначення ступеня належності людини до множини "високі люди" функція належності може бути представлена у вигляді графіка, де значення росту плавно переходять від низького ступеня належності до високого. Це дозволяє гнучко враховувати специфіку реальних даних.

Лінгвістичні змінні є ще одним важливим компонентом нечіткої логіки. Вони використовуються для опису значень у вигляді слів або фраз, що робить моделювання більш зручним і зрозумілим для користувачів. Наприклад, змінна "температура" може мати значення "низька", "середня" або "висока". Такий підхід дозволяє інтуїтивно описувати складні системи, що містять велику кількість нечітких даних.

Нечіткі правила є основою для моделювання й прийняття рішень у системах, побудованих на основі нечіткої логіки. Вони мають вигляд умовних тверджень, наприклад: "Якщо температура висока, то зменшити потужність нагрівача". Такі правила дають змогу формалізувати знання експертів та застосовувати їх у різних галузях. Процес прийняття рішень у таких системах зазвичай включає побудову бази знань, яка містить набір нечітких правил, переведення вхідних даних у нечіткі множини (цей етап називається фізифікація), обробку правил із подальшою агрегацією результатів і переведенням їх назад у точні значення (дефазифікація). Найбільш поширеним підходом для реалізації цього процесу є метод Мамдані, що передбачає структуровану та послідовну обробку інформації.

Теоретичні основи нечіткої логіки також передбачають виконання операцій над нечіткими множинами. Наприклад, об'єднання множин визначає ступінь належності елемента до об'єднаної множини за формулою, яка враховує

максимальне значення ступенів належності цього елемента до кожної з множин і визначається за формулою:

$$\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$$

Перетин множин, навпаки, визначає ступінь належності через мінімальне значення цих ступенів:

$$\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$$

Операція доповнення базується на обчисленні різниці між одиницею та ступенем належності елемента до множини:

$$\mu_{\neg A}(x) = 1 - \mu_A(x)$$

Ці математичні операції створюють основу для складних алгоритмів і дозволяють виконувати гнучкий аналіз розмитих даних.

Нечітка логіка знайшла широке практичне застосування в різних галузях. У технічних системах, таких як побутова техніка, вона використовується для автоматизації процесів. Наприклад, сучасні пральні машини визначають оптимальний режим прання, враховуючи ступінь забруднення одягу та його вагу. Це дозволяє забезпечити більш ефективне й екологічне використання ресурсів. У робототехніці нечітка логіка дозволяє створювати адаптивні алгоритми, які враховують непередбачуваність реального середовища, що є важливим для керування роботами у складних умовах.

В економіці та фінансах нечітка логіка застосовується для аналізу ризиків, оцінки інвестицій та прогнозування ринкових змін, особливо в ситуаціях, коли точні дані недоступні. У медицині її використовують для створення діагностичних систем, здатних працювати з розмитими даними, наприклад, для моделювання процесів постановки діагнозів, де симптоми не завжди мають чіткі межі. Освітня сфера також активно впроваджує нечітку логіку, зокрема для систем тестування знань, де важливо враховувати варіативність відповідей студентів, які можуть бути частково правильними.

Серед переваг нечіткої логіки виділяється її здатність працювати з неточними та неповними даними, а також інтеграція з іншими методами, такими як штучні нейронні мережі. Гнучкість і зрозумілість цього підходу забезпечують легкість інтерпретації результатів. Однак існують і певні недоліки, зокрема складність налаштування функцій належності та правил, що вимагає від користувачів спеціальних знань. Крім того, немає універсального підходу до вибору методу дефазифікації, що ускладнює розробку універсальних рішень для різних задач.

Отже, нечітка логіка є потужним інструментом для моделювання процесів, які характеризуються невизначеністю або нечіткістю даних. Завдяки своїй здатності адаптуватися до складних умов і враховувати неточності, вона знаходить застосування в широкому спектрі завдань — від побутових пристроїв до складних інтелектуальних систем. У контексті тестування знань нечітка логіка дозволяє створювати більш гнучкі та точні методи оцінювання, що значно підвищує якість освітнього процесу.

1.2 Особливості застосування нечіткої логіки в інтелектуальних системах

Застосування нечіткої логіки в інтелектуальних системах стало одним із ключових напрямів сучасних досліджень у галузі штучного інтелекту. Основна причина цього полягає у здатності нечіткої логіки ефективно працювати з неоднозначними, розмитими або неповними даними, що характерні для багатьох реальних ситуацій. Завдяки цій властивості нечітка логіка широко використовується у різноманітних системах, які імітують людське мислення та прийняття рішень.

Однією з головних переваг нечіткої логіки є її здатність адаптуватися до розмитих понять і нечітко визначених умов. Наприклад, в умовах, коли точні математичні моделі складно побудувати через невизначеність вихідних даних, нечітка логіка дозволяє створити модель, яка спирається на наближення та лінгвістичні змінні. У таких випадках поняття, що використовуються, описуються не як чітко визначені величини, а як діапазони значень із

відповідними ступенями належності. Це дозволяє розробляти системи, здатні працювати з поняттями типу "низький ризик", "помірний дохід" або "швидке реагування" [7].

Інтелектуальні системи, побудовані на основі нечіткої логіки, також відрізняються зручністю розробки. Вони дозволяють формалізувати знання експертів у вигляді нечітких правил, що мають форму "Якщо... то...". Це не лише спрощує процес моделювання складних процесів, але й робить ці моделі зрозумілими для людей, які не мають спеціалізованих знань у галузі математичного моделювання [7]. Наприклад, у медичних системах це може бути правило: "Якщо температура пацієнта висока і тиск знижений, то ймовірність критичного стану висока". Такі правила легко сприймаються лікарями і можуть бути швидко адаптовані до специфічних умов.

Ще однією важливою характеристикою нечіткої логіки є її прозорість. Рішення, прийняті на основі нечітких правил, можуть бути пояснені у зрозумілих термінах. Ця властивість особливо цінна в системах, де критично важливо розуміння користувачем того, як і чому система прийняла певне рішення. Наприклад, у фінансових аналітичних системах нечітка логіка дозволяє пояснити, чому певний інвестиційний ризик був класифікований як "високий" або "помірний".

Нечітка логіка також успішно використовується у сфері управління технічними системами. Наприклад, у побутових пристроях, таких як пральні машини або кондиціонери, нечіткі контролери забезпечують оптимізацію роботи залежно від змінних умов. У промислових системах вона використовується для керування складними технологічними процесами, де важко створити точні математичні моделі через значну кількість змінних і невизначеність у даних [8].

У робототехніці нечітка логіка дозволяє створювати адаптивні алгоритми керування, які забезпечують ефективну взаємодію роботів із навколишнім середовищем. Наприклад, автономні транспортні засоби використовують нечітку логіку для прийняття рішень у складних дорожніх ситуаціях. Завдяки здатності аналізувати розмиті параметри, такі як "інтенсивність руху" або

"рівень видимості", ці системи можуть швидко адаптувати свою поведінку до змінних умов.

Освітні системи також виграють від використання нечіткої логіки, зокрема, у процесах оцінювання знань. Врахування таких факторів, як складність завдань, час, витрачений на їх виконання, і ступінь правильності відповідей, дозволяє створювати більш об'єктивні й індивідуально адаптовані методики оцінки. Це допомагає забезпечити ефективніше навчання, враховуючи унікальні особливості кожного студента.

Інтеграція нечіткої логіки з іншими підходами штучного інтелекту також є важливим аспектом її застосування. Наприклад, у поєднанні з нейронними мережами можна створювати системи, які поєднують здатність до навчання та адаптації з інтерпретованістю та гнучкістю нечітких моделей. Генетичні алгоритми, у свою чергу, можуть використовуватися для оптимізації параметрів нечітких систем, таких як функції належності та правила.

Попри численні переваги, застосування нечіткої логіки має й свої виклики. Одним із них є складність створення та налаштування нечітких правил і функцій належності, що вимагає високої кваліфікації фахівців. Крім того, база правил може стати надмірно громіздкою у великих системах, що ускладнює її підтримку та модифікацію. Вибір методів дефазифікації, які переводять нечіткі дані у чіткі результати, також часто є емпіричним і залежить від специфіки завдання.

Нечітка логіка забезпечує потужний інструментарій для вирішення широкого кола задач в інтелектуальних системах, адаптуючи їх до роботи з неточними даними та забезпечуючи прозорість і зручність використання. Її здатність працювати з лінгвістичними змінними та моделювати людські способи мислення робить її однією з найважливіших методологій у сучасних технологіях.

1.3 Аналіз можливостей використання нечіткої логіки в системах тестування знань

З розвитком технічного прогресу одним із головних завдань є покращення якості навчання новітнім технологіям. Одним зі способів вирішення цієї

проблеми є широке використання комп'ютерних технологій на всіх етапах навчання, зокрема під час тестування знань.

На сьогодні найбільш значущою проблемою в системах управління знаннями є примітивний підхід до розрахунку кінцевої оцінки. Найчастіше кількість балів за тестове завдання визначається як відношення кількості правильних відповідей до загальної кількості запитань, а підсумковий бал тесту обчислюється простим додаванням.

Водночас кожне питання має певний рівень складності, який може бути більшим або меншим, а також практичну значущість. Наприклад, неправильна відповідь на фундаментальне чи практично важливе запитання повинна суттєво впливати на загальну оцінку навичок майбутнього спеціаліста.

Перехід від оцінки правильності запропонованих відповідей у рамках бінарної логіки ("правильно–неправильно") до універсальнішої схеми оцінювання, яка враховує частково правильні чи неповні відповіді, можливий за допомогою функцій належності, визначених у категоріях нечіткої логіки. Такий підхід не заперечує традиційного методу, адже за сучасними уявленнями бінарну логіку можна розглядати як окремий випадок нечіткої логіки.

Застосування нечіткої логіки разом із традиційними методами дозволяє спростити створення та розширення можливостей останніх. Для підвищення точності та надійності оцінювання результатів навчання запропоновано використання апарату нечіткої логіки.

Традиційні системи тестування зазвичай оцінюють знання, базуючись на кількості правильних відповідей, що поділяються на загальну кількість запитань для отримання середнього балу. Цей підхід, хоча і є простим для реалізації, має значні недоліки, які обмежують його ефективність та об'єктивність.

Перш за все, такий підхід ігнорує складність запитань. Усі питання, незалежно від рівня їх складності, мають однакову вагу в підсумковій оцінці. Це означає, що правильна відповідь на легке запитання оцінюється так само, як і відповідь на складне, хоча обсяг знань і зусиль, необхідних для вирішення цих завдань, може суттєво відрізнятися.

Другою проблемою є ігнорування практичної значущості запитань. У традиційних системах всі помилки мають однаковий вплив на кінцеву оцінку, незалежно від важливості конкретного питання. Це не враховує того, що помилки в завданнях із високою практичною значущістю можуть бути більш критичними, ніж у завданнях, які мають менший вплив на загальну оцінку знань.

Крім того, традиційний підхід базується на чіткій дихотомії "правильно/неправильно", що не дозволяє врахувати частково правильні або неточні відповіді. Наприклад, якщо учасник тесту демонструє часткове розуміння концепції, але припускається незначних помилок, це не відображається у підсумковій оцінці. Таким чином, система не стимулює детальний аналіз відповідей і не дозволяє отримати більш гнучкий та об'єктивний результат.

Зазначені обмеження вказують на потребу у вдосконаленні підходів до тестування знань, зокрема, через використання методів нечіткої логіки. Ці методи здатні враховувати рівень складності, практичну значущість та часткову правильність відповідей, забезпечуючи більш точну та справедливую оцінку знань.

Нечітка логіка дозволяє враховувати вищезазначені аспекти через використання мовних змінних, нечітких множин і функцій належності.

Структура нечіткого виведення показана на рис. 2. Вона складається з кількох шарів, де:

1-й шар - це вхідний шар, який реалізує тестові питання. Це шар фазифікації вхідних змінних.

2-й шар - проміжний шар, який реалізує логічні правила.

3-й шар - вихідний шар, що реалізує оцінку за Т-бальною шкалою. Це шар дефазифікації.

$$y_j = \sum w_i x_i, \quad i = \overline{1, N}, \quad j = \overline{1, 12}.$$

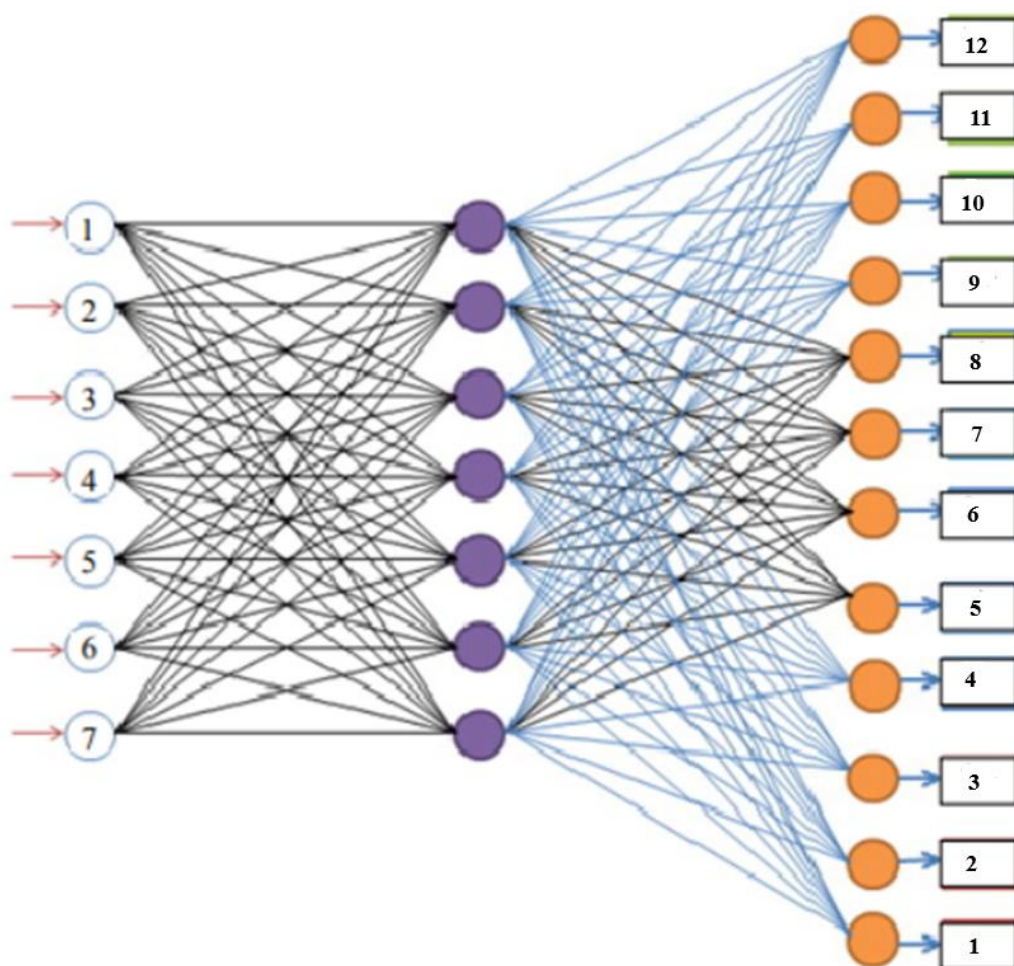


Рис. 1.2. Структура нечіткого виведення (на прикладі 12-бальної шкали)

Нечітка логіка відкриває широкі можливості для вдосконалення систем оцінювання знань, дозволяючи враховувати різні аспекти та рівні невизначеності. Основним інструментом у таких системах є використання лінгвістичних змінних, які можуть описувати ключові параметри оцінювання, такі як складність завдань, їх практична значущість, актуальність, точність, оригінальність тощо.

Кожна з цих змінних може бути описана через набір термінів, наприклад, "низький", "середній" і "високий", які мають відповідні функції належності. Це дозволяє перетворювати кількісні параметри в якісні оцінки та враховувати нечіткість у даних.

Рішення у таких системах приймаються шляхом нечіткого логічного виведення, яке передбачає:

1. Фазифікацію вхідних змінних – переведення кількісних даних у терміни лінгвістичних змінних.
2. Застосування правил нечіткої логіки, які базуються на експертних знаннях. Наприклад, правила можуть описувати взаємозв'язки між складністю завдань і їхньою важливістю.
3. Агрегацію результатів – об'єднання результатів виведення з усіх правил.
4. Дефазифікацію – переведення нечіткої оцінки в конкретне числове значення, яке може бути використане для прийняття рішень.

Такі підходи дозволяють створювати системи оцінювання, що враховують як загальну якість тестових завдань, так і індивідуальні особливості виконання. Гнучкість і адаптивність нечіткої логіки сприяють побудові ефективних моделей оцінювання в умовах невизначеності та забезпечують точнішу інтерпретацію результатів у порівнянні зі звичайними методами.

1.4 Постановка задачі та вибір технологій для розробки

Виходячи з теоретичних основ, розглянутих у попередніх розділах, метою цієї роботи є аналіз можливостей використання методів нечіткої логіки для підвищення ефективності систем тестування знань. Відповідно, завдання цієї роботи включають розробку веб орієнтованої системи тестування, яка базується на принципах нечіткого виведення, та порівняння її результативності із традиційними підходами до оцінювання знань.

Розробка підходу використання нечіткої логіки в системі тестування знань розпочинається з врахування характеристик, які були детально визначені в теоретичній частині дослідження. Основою цієї системи є кілька ключових компонентів, що забезпечують її функціональність та ефективність. По-перше, визначаються лінгвістичні змінні, які є інструментом для опису нечітких даних у вигляді зрозумілих термінів, наприклад, "низький рівень знань" чи "високий рівень складності завдань". Ці змінні дозволяють створити інтуїтивно зрозумілу модель оцінювання. Наступним важливим елементом є набір правил, побудованих на основі алгоритму Мамдані. Ці правила використовуються для

формалізації процесу прийняття рішень, враховуючи взаємозв'язок між вхідними даними та очікуваними результатами. Крім того, функції належності забезпечують можливість представлення нечітких даних у вигляді графіків, що дозволяє користувачам краще розуміти логіку системи. Завдяки фазифікації, агрегуванню та дефазифікації дані обробляються та перетворюються у кінцевий результат, який є зрозумілим і точним.

Наступним кроком у розробці є створення веб-орієнтованої системи тестування. Для забезпечення зручності використання та інтеграції з сучасними технологіями було обрано підхід, який передбачає реалізацію системи у вигляді веб-додатка. Основу розробки складає стек технологій, який включає кілька потужних інструментів. Використання ASP.NET Core MVC дозволяє створити додаток із чіткою архітектурою Model-View-Controller, що сприяє підвищенню продуктивності та забезпечує можливість масштабування системи. Завдяки інтеграції Razor Pages розробники можуть поєднувати серверну логіку з відображенням, що полегшує створення інтерфейсу користувача. Як база даних використовується SQLite, яка є легкою, продуктивною та чудово підходить для невеликих і середніх веб-застосунків. Інтеграція з Entity Framework Core спрощує роботу з базами даних завдяки підтримці LINQ, що забезпечує зручний доступ до даних і їх обробку.

Для реалізації нечіткої логіки використовується Python разом із бібліотекою Scikit-Fuzzy. Цей інструмент надає широкий набір функцій для роботи з нечіткими множинами, що дозволяє створювати складні моделі нечіткого виведення. Завдяки бібліотеці Matplotlib забезпечується якісна побудова графіків, які наочно демонструють функції належності та результати оцінювання. Обробка даних виконується Python-скриптом, який інтегрується з веб-інтерфейсом. Він отримує дані з бази, проводить аналіз за допомогою нечіткої логіки та повертає результати у форматі, придатному для подальшого відображення.

Створена система тестування знань має забезпечити низку функцій, які сприяють її практичності та зручності. Веб-інтерфейс дозволяє користувачам

завантажувати та керувати тестовими завданнями, налаштовувати параметри тестів і лінгвістичні змінні, а також визначати правила виведення. Система автоматично розраховує оцінку на основі нечіткої логіки, забезпечуючи об'єктивність та враховуючи нюанси відповідей. Результати представлено у вигляді графіків і таблиць, що дозволяє порівнювати їх із традиційними методами оцінювання. Це порівняння є ключовим для аналізу ефективності системи.

Для оцінки ефективності розробленого підходу буде проведено експериментальне тестування. Воно включає кілька етапів, зокрема тестування на вибірці користувачів із застосуванням обох методів оцінювання, традиційного та з використанням нечіткої логіки. Зібрані дані будуть аналізуватися з метою оцінки точності, зручності використання системи та адаптивності результатів. Цей аналіз допоможе виявити переваги та недоліки запропонованого підходу та сформулювати рекомендації щодо його вдосконалення.

Одним із важливих напрямків у розробці є моделювання адаптивних тестів. Завдяки використанню нечіткої логіки система може підлаштовувати складність завдань залежно від рівня знань користувача, що робить процес тестування більш індивідуалізованим і ефективним. Наприклад, якщо система визначає, що користувач має високий рівень знань, вона автоматично пропонує складніші запитання. Оцінювання також стає більш об'єктивним, оскільки враховуються практичні аспекти відповідей, а не лише їх формальна правильність.

Розроблена система має широкий спектр можливих застосувань. У закладах освіти вона може використовуватися для адаптивного тестування знань студентів, надаючи можливість враховувати індивідуальні особливості кожного учасника. У корпоративному середовищі система стане інструментом для оцінювання професійних навичок співробітників, що дозволить більш точно визначити рівень їхньої підготовки та потребу в додатковому навчанні. У наукових дослідженнях розроблена система стане корисною для аналізу ефективності тестових завдань та порівняння різних методів оцінювання знань.

Таким чином, запропонована система є універсальним інструментом, який може бути адаптований до різних сфер діяльності.

Таким чином, основним результатом роботи буде створення інструменту, який використовує переваги нечіткої логіки для покращення процесу тестування знань. Розроблена система не лише спрощує процес розробки тестів і аналізу результатів, але й забезпечує більш об'єктивну та адаптивну оцінку рівня знань.

Висновки до першого розділу.

У першому розділі проаналізовано традиційні методи тестування знань та їх недоліки, зокрема ігнорування складності завдань і практичного значення питань. Запропоновано використання нечіткої логіки для подолання цих обмежень завдяки застосуванню лінгвістичних змінних, нечітких множин і алгоритмів виведення. Розглянуто основи методу Мамдані, який дозволяє створити більш адаптивну та об'єктивну систему оцінювання. Поставлено завдання розробити веб-орієнтовану систему тестування з використанням нечіткої логіки, що враховує складність і практичну значущість питань. Це забезпечить гнучкість і підвищить точність оцінювання знань у порівнянні з традиційними підходами.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА МОДЕЛІ ТЕСТУВАННЯ ЗНАНЬ ІЗ ВИКОРИСТАННЯМ НЕЧІТКОЇ ЛОГІКИ

2.1 Розробка моделі оцінювання результатів тестування за допомогою нечіткої логіки

Модель оцінювання результатів тестування базується на використанні нечіткої логіки для визначення якості кожного питання тесту. Основними факторами, що впливають на якість питання, є його складність (S) і практичне значення (Z). Ці два параметри були визначені як лінгвістичні змінні, кожна з яких має три терміни: "низький", "середній", і "високий".

Вихідна змінна, коефіцієнт якості (k), оцінюється в діапазоні від 1 до 10 і впливає на вагу питання в загальній оцінці студента. Для обчислення коефіцієнта якості використовується алгоритм Мамдані, який дозволяє формалізувати правила оцінки на основі нечіткої логіки.

Максимальна кількість балів для кожної позиції заздалегідь визначена. Для обох лінгвістичних змінних S та Z надається набір термінів ("низький", "проміжний", "високий"), а їхні функції належності показані на рисунках 3 і 4.

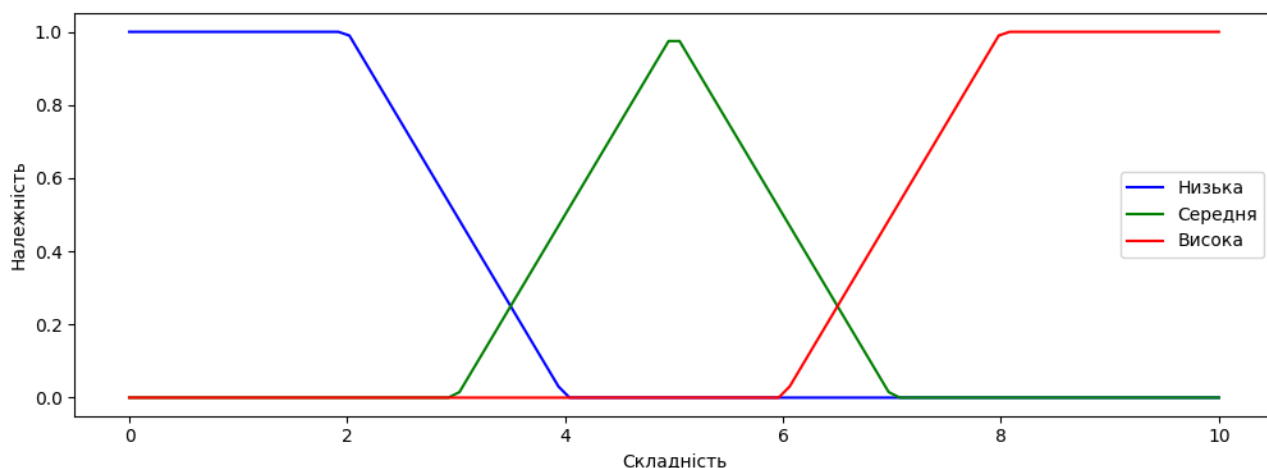


Рис. 2.1. Функція належності для змінної «складність»

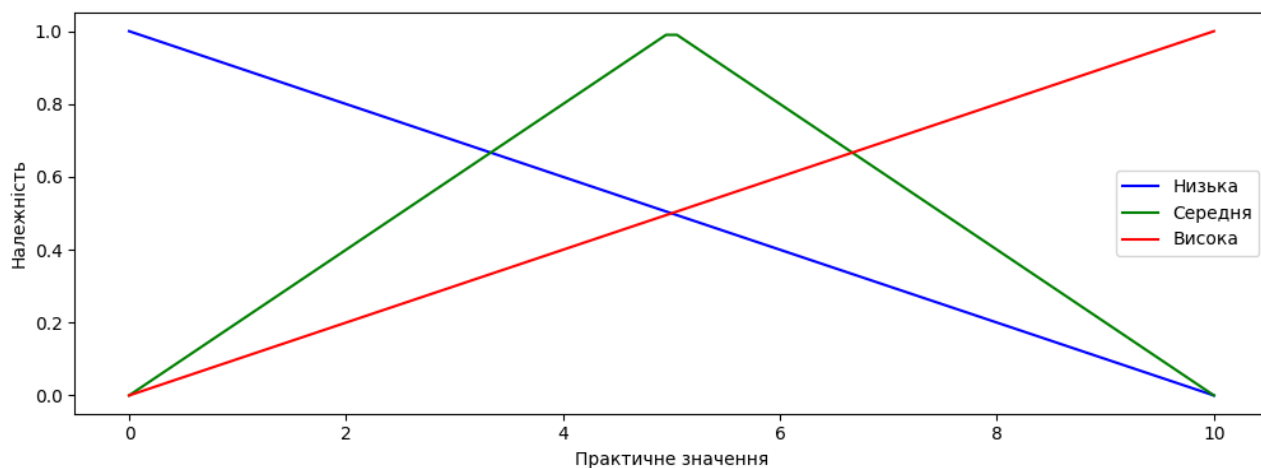


Рис. 2.2. Функція належності для змінної «практичне значення»

Вихідна змінна - k , це коефіцієнт якості питання, який визначається в діапазоні від двох до восьми. Кількість отриманих балів учня за відповідь на питання множиться на числове значення коефіцієнта k . Різні питання можуть мати різну максимально можливу кількість балів.

Далі можна сформулювати правила висновку в формі Мамдані:

Якщо S є "низьким" і Z є "низьким", то k є "низьким".

Якщо S є "проміжним" і Z є "проміжним", то k є "проміжним".

Якщо S є "високим" і Z є "високим", то k є "високим".

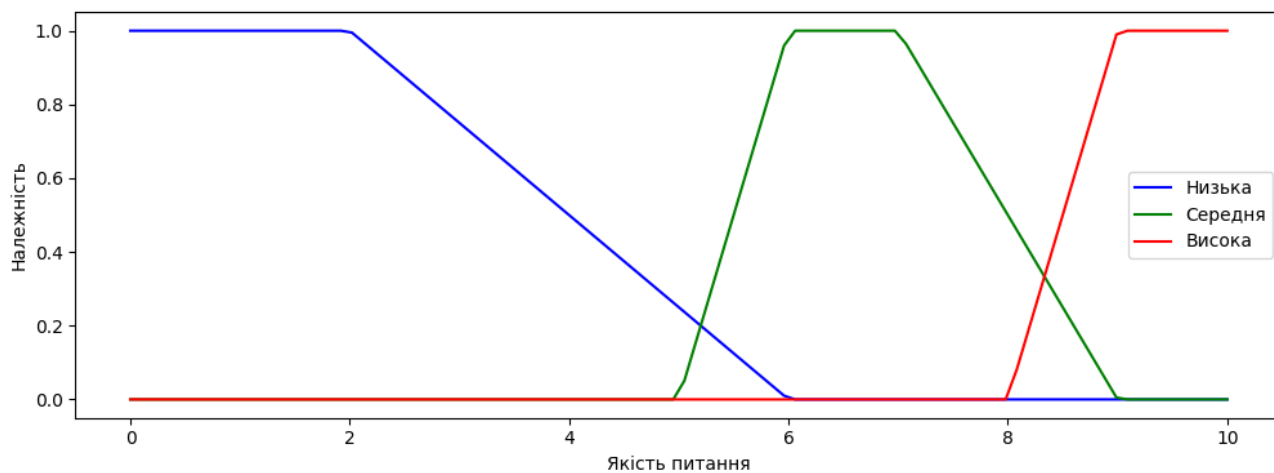


Рис. 2.3. Функція належності для «коефіцієнта якості»

Тому визначити етапи прийняття рішень за умов невизначеності можна з використанням алгоритму Мамдані:

1. вибір факторів, на підставі яких прийматиметься рішення. Для кожного фактора встановити його значення (набір термінів), визначити функції належності для кожного лінгвістичного терміну з основного набору

термінів. Крім того, визначається рівень впевненості для кожного правила;

2. розрахунок виведення кожного правила за допомогою операції мінімуму;
3. об'єднання всіх нечітких множин, отриманих на виведенні правил, за допомогою операції максимуму в одній нечіткій множині;
4. перехід від нечіткої множини до конкретного значення.

Для розробки та реалізації даного алгоритму на практиці було використано мову програмування Python. Для створення моделі було використано бібліотеку Scikit-Fuzzy, яка дозволяє створювати лінгвістичні змінні, визначати функції належності та будувати правила висновку [13].

Створення вхідних і вихідних змінних:

```
difficulty = ctrl.Antecedent(np.arange(0, 11, 1), 'difficulty')
practical_value = ctrl.Antecedent(np.arange(0, 11, 1), 'practical_value')
quality_coefficient = ctrl.Consequent(np.arange(0, 11, 1), 'quality_coefficient')
```

- difficulty: змінна, що відображає складність питання, приймає значення від 0 до 10.
- practical_value: змінна, що оцінює практичну важливість питання, також у діапазоні від 0 до 10.
- quality_coefficient: вихідна змінна, яка визначає якість питання. Її значення варіюються від 0 до 10.

Ці змінні створюються за допомогою об'єктів Antecedent і Consequent з бібліотеки **Scikit-Fuzzy**, що дозволяє автоматично генерувати функції належності.

Для кожної змінної задаються три функції належності: "низький", "середній", і "високий". Вони описуються математичною моделлю: трикутні (trimf) і трапецієподібні (trapmf) функції.

```
difficulty['low'] = fuzz.trapmf(difficulty.universe, [0, 0, 2, 4])
difficulty['medium'] = fuzz.trimf(difficulty.universe, [3, 5, 7])
difficulty['high'] = fuzz.trapmf(difficulty.universe, [6, 8, 10, 10])
```

- "Низька складність": визначається як трапецієподібна функція з піковим значенням у діапазоні від 0 до 4.
- "Середня складність": трикутна функція, що має максимум при значенні 5.
- "Висока складність": трапецієподібна функція з піком від 6 до 10.

Аналогічно задаються функції для змінних `practical_value` та `quality_coefficient`:

```
practical_value['low'] = fuzz.trimf(practical_value.universe, [0, 0, 10])
practical_value['medium'] = fuzz.trimf(practical_value.universe, [0, 5, 10])
practical_value['high'] = fuzz.trimf(practical_value.universe, [0, 10, 10])

quality_coefficient['low'] = fuzz.trapmf(quality_coefficient.universe, [0, 0, 2, 6])
quality_coefficient['medium'] = fuzz.trapmf(quality_coefficient.universe, [5, 6, 7, 9])
quality_coefficient['high'] = fuzz.trapmf(quality_coefficient.universe, [8, 9, 10, 10])
```

Для опису логіки оцінювання використовується 9 правил, які визначають взаємозв'язок між складністю, практичним значенням і коефіцієнтом якості.

```
rule1 = ctrl.Rule(difficulty['low'] & practical_value['low'],
quality_coefficient['low'])
rule2 = ctrl.Rule(difficulty['low'] & practical_value['medium'],
quality_coefficient['low'])
rule3 = ctrl.Rule(difficulty['low'] & practical_value['high'],
quality_coefficient['medium'])
rule4 = ctrl.Rule(difficulty['medium'] & practical_value['low'],
quality_coefficient['low'])
rule5 = ctrl.Rule(difficulty['medium'] & practical_value['medium'],
quality_coefficient['medium'])
rule6 = ctrl.Rule(difficulty['medium'] & practical_value['high'],
quality_coefficient['high'])
rule7 = ctrl.Rule(difficulty['high'] & practical_value['low'],
quality_coefficient['medium'])
rule8 = ctrl.Rule(difficulty['high'] & practical_value['medium'],
quality_coefficient['high'])
rule9 = ctrl.Rule(difficulty['high'] & practical_value['high'],
quality_coefficient['high'])
```

Кожне правило визначає, як конкретна комбінація значень вхідних змінних впливає на вихідну змінну.

Після створення правил вони об'єднуються у систему управління.

```
quality_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7,
rule8, rule9])
quality_simulation = ctrl.ControlSystemSimulation(quality_ctrl)
```

Функція для обчислення коефіцієнта якості (k) для заданого рівня складності й практичного значення:

```
def calculate_quality_coefficient(difficulty_level, practical_value_level):
    quality_simulation.input['difficulty'] = difficulty_level
    quality_simulation.input['practical_value'] = practical_value_level
    quality_simulation.compute()
    return quality_simulation.output['quality_coefficient']
```

Ця функція отримує значення двох змінних, виконує обчислення і повертає результат.

Для аналізу та перевірки правильності заданих функцій належності була створена функція візуалізації:

```
def plot_membership_functions():
    x_difficulty = np.linspace(0, 10, 100)

    difficulty_low = fuzz.trapmf(x_difficulty, [0, 0, 2, 4])
    difficulty_medium = fuzz.trimf(x_difficulty, [3, 5, 7])
    difficulty_high = fuzz.trapmf(x_difficulty, [6, 8, 10, 10])

    fig, axs = plt.subplots(3, 1, figsize=(8, 12))

    axs[0].plot(x_difficulty, difficulty_low, 'b', label='Низька')
    axs[0].plot(x_difficulty, difficulty_medium, 'g', label='Середня')
    axs[0].plot(x_difficulty, difficulty_high, 'r', label='Висока')
    axs[0].set_title('Функції належності для складності')
    axs[0].set_xlabel("Складність")
    axs[0].set_ylabel("Належність")
    axs[0].legend()

    plt.tight_layout()
    plt.show()
```

Ця функція будує графіки для кожної змінної, демонструючи, як вони поділяються на терміни.

Запропонований підхід дозволяє враховувати суб'єктивність і неповноту інформації при визначенні підсумкових оцінок, що підвищує гнучкість та об'єктивність системи. Розроблена модель забезпечує адаптацію до різних умов оцінювання та сприяє підвищенню точності аналізу результатів тестування. Це створює основу для подальшої реалізації методів у програмному забезпеченні.

2.2 Проектування веб-орієнтованої системи тестування знань

Проектування системи тестування знань є одним із ключових етапів її розробки, оскільки на цьому етапі визначаються основні компоненти, їхні зв'язки та функції. Розглянемо процес проектування, орієнтуючись на структурну та об'єктно-орієнтовану архітектуру.

На початку було визначено основні функціональні частини системи: модуль управління тестами, модуль проведення тестування та модуль оцінювання результатів. Кожен із цих модулів реалізує певну бізнес-логіку, що забезпечує коректне функціонування системи в цілому [17].

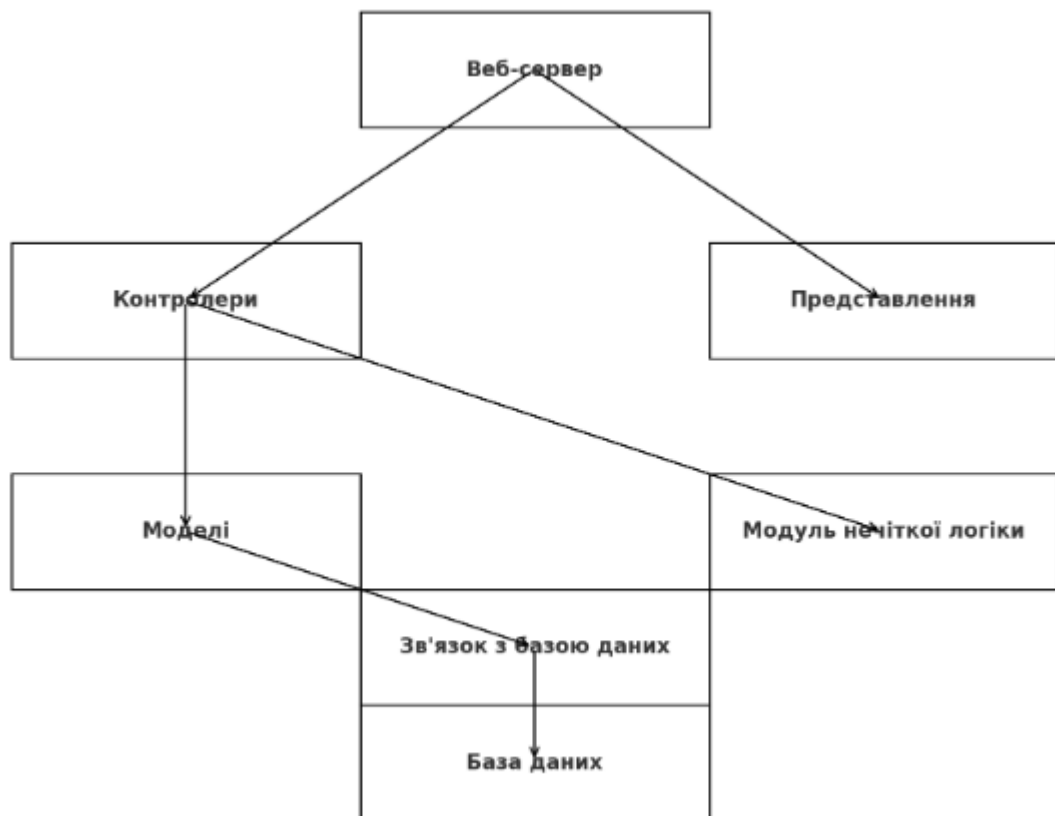


Рис. 2.4. Загальна схема структури додатку

На рівні проектування було виділено кілька ключових сутностей, які реалізуються у вигляді класів. Кожна сутність моделює відповідний аспект системи тестування.

На етапі проектування системи тестування знань було виділено кілька ключових сутностей, які реалізуються через класи. Кожна сутність відповідає

певному аспекту функціональності системи та забезпечує її структурованість і логічну цілісність.

Однією з основних сутностей є клас `Test`, який об'єднує питання, результати тестування та основні параметри тесту. Цей клас містить дані про назву, опис і тривалість тесту, а також включає список питань і результати, отримані студентами. Така структура дозволяє централізовано керувати усіма тестами в системі та забезпечує зручність в управлінні.

Клас `Question` реалізує модель питання як ключового елементу тесту. Він включає текст самого питання, рівень його складності, практичну значущість і перелік можливих відповідей. Це дозволяє враховувати різні аспекти оцінювання знань, включаючи такі характеристики, як складність завдань і їхня релевантність до практичних потреб.

Для представлення можливих відповідей на питання використовується клас `AnswerOption`. Він містить текст варіанту відповіді та маркер, який вказує, чи є цей варіант правильним. Це дозволяє гнучко формувати набір можливих відповідей для кожного питання, що є основою для адаптивного підходу в тестуванні.

Процес оцінювання результатів тестування реалізовано через клас `TestResult`, який моделює підсумки виконання тесту студентом. У цьому класі зберігається ім'я студента, отримана оцінка, а також список відповідей на всі питання тесту. Такий підхід дозволяє створювати деталізовані звіти, які містять інформацію про правильність відповідей і розраховані бали.

Окремо виділяється клас `Answer`, який моделює вибір студента на кожне питання. Він включає інформацію про обраний варіант відповіді, правильність вибору та отриманий бал із урахуванням вагових коефіцієнтів, визначених у системі. Цей клас відіграє ключову роль у формуванні точного результату тестування.

Усі сутності в системі взаємопов'язані, що забезпечує відображення реальної логіки роботи тестування. Наприклад, кожен тест містить список питань, кожне питання має набір варіантів відповідей, а результати тесту складаються зі списку

відповідей студента, кожна з яких прив'язана до певного питання. Такий підхід забезпечує цілісність даних і дозволяє ефективно керувати усіма етапами тестування.

Також було виділено допоміжні класи – view models(моделі представлення). Ці класи являть собою проміжний шар між основними класами і представленнями і можуть містити додаткову інформацію. Загальна діаграма класів систем зображена на Рис. 2.5.

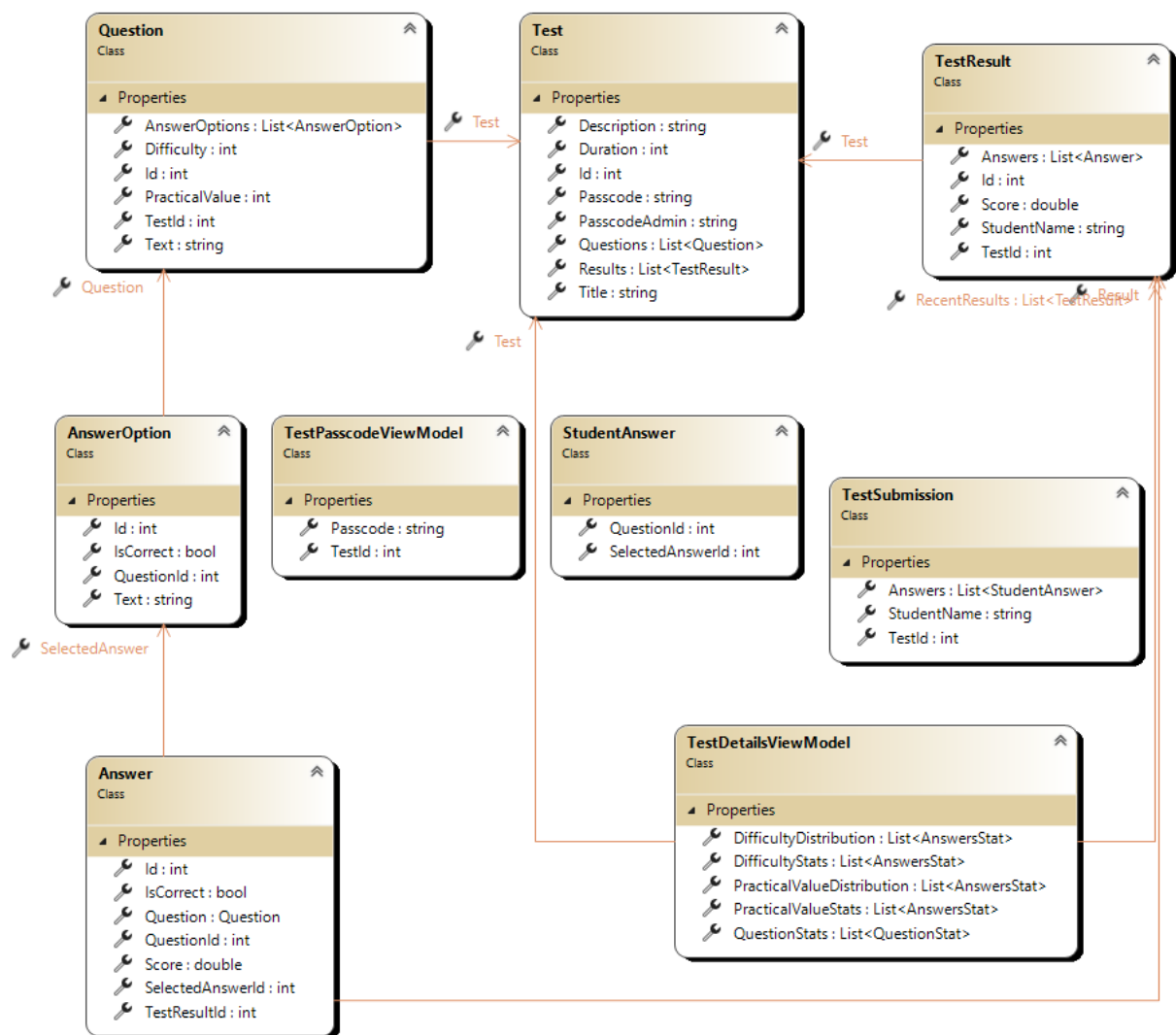


Рис. 2.5. Діаграма класів системи

При проектуванні бази даних для системи тестування знань було обрано реляційну модель, яка забезпечує ефективне управління даними та їх інтеграцію. У процесі розробки моделі бази даних було враховано структуру класів, що використовуються в програмній частині, оскільки вони безпосередньо

відповідають сутностям у базі даних. Таким чином, кожен клас відповідає одній таблиці в базі даних.

Модель бази даних складається з п'яти основних таблиць: Test, Question, AnswerOption, TestResult і Answer. Вони пов'язані між собою через зовнішні ключі, що забезпечує узгодженість та цілісність даних.

Таблиця 2.1

Опис таблиці Test

Назва поля	Тип даних	Опис	Обмеження
Id	INT	Унікальний ідентифікатор тесту	PRIMARY KEY, AUTO_INCREMENT
Title	NVARCHAR(255)	Назва тесту	NOT NULL
Description	NVARCHAR(MAX)	Опис тесту	NULLABLE
Duration	INT	Тривалість тесту в хвилинали	NOT NULL
Passcode	NVARCHAR(50)	Пароль для проходження тесту	NULLABLE
PasscodeAdmin	NVARCHAR(50)	Адміністративний пароль	NULLABLE

Таблиця 2.2

Опис таблиці Question

Назва поля	Тип даних	Опис	Обмеження
Id	INT	Унікальний ідентифікатор питання	PRIMARY KEY, AUTO_INCREMENT
Text	NVARCHAR(MAX)	Текст питання	NOT NULL

Difficulty	INT	Рівень складності (1-10)	NOT NULL, CHECK (Difficulty BETWEEN 1 AND 10)
PracticalValue	INT	Практична значущість (1-10)	NOT NULL, CHECK (PracticalValue BETWEEN 1 AND 10)
TestId	INT	Зовнішній ключ до таблиці Test	FOREIGN KEY (TestId) REFERENCES Test(Id)

Таблиця 2.3

Опис таблиці AnswerOption

Назва поля	Тип даних	Опис	Обмеження
Id	INT	Унікальний ідентифікатор відповіді	PRIMARY KEY, AUTO_INCREMENT
Text	NVARCHAR(MAX)	Текст варіанту відповіді	NOT NULL
IsCorrect	BIT	Позначає, чи є відповідь правильною	NOT NULL
QuestionId	INT	Зовнішній ключ до таблиці Question	FOREIGN KEY (QuestionId) REFERENCES Question(Id)

Таблиця 2.4

Опис таблиці TestResult

Назва поля	Тип даних	Опис	Обмеження
Id	INT	Унікальний ідентифікатор результату	PRIMARY KEY, AUTO_INCREMENT
TestId	INT	Зовнішній ключ до таблиці Test	FOREIGN KEY (TestId) REFERENCES Test(Id)
StudentName	NVARCHAR(255)	Ім'я студента	NOT NULL
Score	FLOAT	Підсумковий бал	NOT NULL

Таблиця 2.5

Опис таблиці Answer

Назва поля	Тип даних	Опис	Обмеження
Id	INT	Унікальний ідентифікатор відповіді	PRIMARY KEY, AUTO_INCREMENT
TestId	INT	Зовнішній ключ до таблиці TestResult	FOREIGN KEY (TestId) REFERENCES TestResult(Id)
QuestionId	INT	Зовнішній ключ до таблиці Question	FOREIGN KEY (QuestionId) REFERENCES Question(Id)
SelectedAnswerId	INT	Зовнішній ключ до таблиці AnswerOption	FOREIGN KEY (SelectedAnswerId) REFERENCES AnswerOption(Id)

IsCorrect	BIT	Позначає, чи є відповідь правильною	NOT NULL
Score	FLOAT	Бал, отриманий за це питання	NOT NULL

У системі тестування знань взаємозв'язки між таблицями бази даних відіграють важливу роль у забезпеченні узгодженості й функціональності даних. Усі сутності мають чітко визначені зв'язки, що відображають логіку їх взаємодії та забезпечують коректну організацію інформації.

Таблиця Test пов'язана із таблицею Question через зв'язок "один-до-багатьох". Це означає, що один тест може містити багато питань. Така структура дає змогу зберігати всі питання, які належать до певного тесту, у єдиній таблиці, тоді як кожне питання посилається на свій тест через зовнішній ключ. Завдяки цьому система дозволяє гнучко керувати тестами, додаючи до них питання або видаляючи їх без шкоди для інших тестів.

Таблиця Question, у свою чергу, пов'язана із таблицею AnswerOption, також через зв'язок "один-до-багатьох". Це дозволяє кожному питанню мати кілька варіантів відповідей, які зберігаються у таблиці варіантів. Для кожного варіанта відповідей фіксується текст відповіді та позначка про її правильність, що забезпечує ефективний процес оцінювання під час тестування.

Таблиця TestResult зберігає результати проходження тестів і має зв'язок "один-до-багатьох" із таблицею Test. Один тест може мати багато результатів, кожен з яких відповідає спробі проходження тесту певним студентом. Такий зв'язок дозволяє зберігати історію тестування, надаючи доступ до всіх попередніх результатів.

Таблиця TestResult також пов'язана із таблицею Answer через зв'язок "один-до-багатьох". Кожен результат тесту може містити багато відповідей, які студенти надали під час проходження тесту. Це забезпечує детальну інформацію

про кожен вибір, зроблений студентом, і дозволяє аналізувати, які питання викликали складнощі.

Таблиця Answer, у свою чергу, має зв'язок "один-до-багатьох" із таблицею Question, оскільки однакове питання може бути використане в різних тестах або спробах. Такий підхід дозволяє зберігати відповіді на питання разом із відповідними атрибутами, як-от правильність, бал за відповідь і час, витрачений на питання.

Усі ці зв'язки забезпечують цілісність і логічну структуру бази даних. Кожен елемент у системі тестування знань чітко взаємодіє з іншими, що дозволяє створювати, змінювати й аналізувати тести, питання, результати та відповіді. Ця архітектура є основою для успішного функціонування системи та її розширення в майбутньому.

Контролери системи тестування знань забезпечують основну логіку взаємодії користувачів із додатком, дозволяючи працювати з тестами, питаннями та результатами. На основі підходу Model-View-Controller (MVC) реалізовано два ключових контролери: TestController та TestTakingController. Їхні функції деталізують, як система обробляє запити користувачів, забезпечуючи зручність роботи як для студентів, так і для адміністраторів.

TestController відіграє важливу роль у роботі системи, зокрема в управлінні тестами та наданні доступу до їхніх характеристик. Коли користувач відкриває сторінку, що містить перелік тестів, цей контролер викликає метод Index, який відповідає за завантаження всіх доступних тестів із бази даних і передачу їх у представлення. Усі доступні тести з відповідними характеристиками, як-от назва, кількість питань, середня складність, тривалість і опис, відображаються у зрозумілому форматі для зручності вибору. Коли користувач хоче переглянути деталі тесту, викликається метод Details. Він завантажує детальну інформацію про обраний тест, включаючи опис, склад питань, їхню складність і практичну значущість, а також останні результати тестування. Крім того, це представлення візуалізує дані за допомогою графіків, таких як діаграма залежності між правильністю відповідей і складністю питань або кругова діаграма, що

демонструє розподіл питань за практичною цінністю. Метод Create у цьому контролері призначений для адміністраторів. Він дозволяє створювати нові тести, додаючи до них питання, які адміністратор може налаштувати, задаючи рівень складності та варіанти відповідей.

Контролер TestTakingController відповідає за організацію процесу проходження тестів. Метод Start запускає тестування, завантажуючи питання з бази даних і передаючи їх у представлення, де вони відображаються у динамічному форматі. Користувач бачить текст питання, варіанти відповідей і таймер, що забезпечує контроль за часом проходження. Коли тест завершено, дані про відповіді надсилаються на сервер через метод Submit. Цей метод аналізує отримані відповіді, оцінює їх на основі алгоритмів нечіткої логіки, а потім зберігає результати в базі даних. Результати відображаються у методі Result, який надає студенту інформацію про правильність і неправильність його відповідей, загальну оцінку, а також візуалізує статистику виконання тесту. Завдяки графікам, студенти можуть оцінити свої сильні та слабкі сторони.

Інтерфейс системи створено з урахуванням потреб користувачів. Сторінка зі списком тестів надає коротку інформацію про кожен тест, а деталі тесту дозволяють глибше зрозуміти його структуру. Процес проходження тесту організовано так, щоб забезпечити інтуїтивно зрозумілу взаємодію із запитаннями. Усі сторінки реалізовані за допомогою Razor-синтаксису, що дозволяє поєднувати серверну логіку з відображенням на HTML-сторінках. Інтерактивні елементи, як-от графіки та діаграми, створюються за допомогою бібліотеки Chart.js, що додає сучасності й візуальної привабливості системі.

Таким чином, взаємозв'язок між контролерами, представленнями та моделями дозволяє досягти високої функціональності системи тестування знань, забезпечуючи ефективне й зручне управління тестами, їхнє проходження та аналіз результатів.

Зважаючи на спроектовані компоненти було розроблено діаграму модулів системи(рис. 2.7) та блок схеми для основних процесів в додатку (рис. 2.6, 2.8).

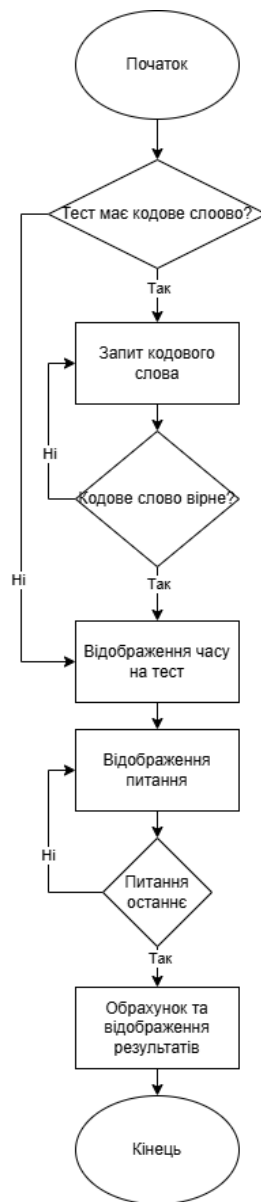


Рис. 2.6. Блок-схема процесу проходження тестування



Рис. 2.7. Діаграма модулів системи



Рис. 2.8. Блок-схема процесу обрахунку оцінки

Отже, було здійснено проектування веб-орієнтованої системи тестування знань, визначено її основні компоненти та структуру. Було розроблено моделі даних, які забезпечують логічну організацію інформації, необхідної для роботи системи.

2.3 Розробка веб-орієнтованої системи тестування знань з використанням методів нечіткої логіки

Спочатку були створені класи та моделі, що описані в розділі 2.2. Після цього можна переходити до безпосередньої розробки функціоналу додатку та реалізації алгоритмів, але перед цим треба налаштувати ASP.NET Core проект для коректної роботи з базою даних та сесіями. Для налаштування бази даних в файлі appsettings.json було заповнено connection string:

```

"ConnectionStrings": {
  "DefaultConnection": "Data Source=app.db"
},

```

Далі було налаштовано Entity Framework – ORM для роботи з БД:

```
SQLitePCL.Batteries.Init();
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));
```

І налаштовано роботу сесій для зберігання певної інформації щодо поточного користувача:

```
builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromMinutes(30);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});
app.UseSession();
```

Після налаштування середовища ASP.NET Core, зокрема підключення бази даних та конфігурації сесій, було розпочато розробку функціоналу веб-додатку. Першим етапом стала реалізація функціоналу для роботи з тестами: створення, перегляд та управління тестами. У контролері HomeController реалізовано методи для перегляду списку тестів, створення нового тесту, а також для відображення деталей тесту. Метод Index надає можливість виконувати пошук тестів за назвою та підтримує пагінацію для зручності користувача.

```
public IActionResult Index(string searchQuery, int? page)
{
    const int pageSize = 6;

    var tests = dbContext.Tests.AsQueryable();

    if (!string.IsNullOrEmpty(searchQuery))
    {
        tests = tests.Where(t => t.Title.Contains(searchQuery));
    }

    var pagedTests = tests.OrderByDescending(x => x.Id).ToPagedList(page ?? 1,
        pageSize);

    ViewBag.SearchQuery = searchQuery;
    return View(pagedTests);
}
```

Цей код дозволяє користувачам легко знаходити необхідні тести за ключовими словами, що значно покращує зручність роботи з системою. Методи Create в HomeController забезпечують як відображення форми створення тесту, так і обробку даних, введених користувачем. Код перевіряє валідність отриманих даних та зберігає тест у базі даних.

[HttpPost]

```

public IActionResult Create(Test test, List<Question> questions)
{
    if (ModelState.IsValid)
    {
        dbContext.Tests.Add(test);
        dbContext.SaveChanges();
        TempData["SuccessMessage"] = "Тест успішно створено!";
        return RedirectToAction("Index");
    }
    return View(test);
}

```

Ця функціональність дозволяє адміністраторам додавати нові тести з можливістю додавання запитань до них.

Метод Details забезпечує відображення детальної інформації про тест, включаючи статистику виконання та успішність відповідей на запитання. Крім того, передбачено механізм захисту доступу до тесту за допомогою кодового слова.

Фрагмент коду для перевірки доступу до тесту:

```

if (test.Passcode != null)
{
    var testAccess = HttpContext.Session.GetInt32($"TestAccessAdmin_{id}");
    if (testAccess != 1)
    {
        return RedirectToAction("EnterPasscode", new { id });
    }
}

```

Цей функціонал забезпечує конфіденційність тестів, доступ до яких обмежено адміністраторами.

Для аналізу результатів тестування реалізовано обчислення статистичних даних, таких як відсоток правильних відповідей залежно від складності запитань чи їх практичної цінності. Ці дані допомагають адміністраторам оцінити ефективність запитань та тесту загалом.

Фрагмент коду для обчислення статистики:

```

var difficultyStats = test.Results.SelectMany(x => x.Answers)
    .GroupBy(q => q.Question.Difficulty)
    .Select(group => new AnswersStat
    {
        Key = group.Key,
        Value = group.Any()
            ? Math.Round(group.Count(a => a.IsCorrect) * 100.0 / group.Count(), 2)
            : 0
    })
    .OrderBy(stat => stat.Key)
    .ToList();

```

Подальша розробка продовжиться реалізацією механізмів оцінювання результатів тестування з використанням методів нечіткої логіки.

Контролер `TestTakingController` обробляє взаємодію користувача з тестовою системою, забезпечуючи доступ до тесту, обробку відповідей та відображення результатів.

Цей фрагмент відповідає за завантаження тесту за вказаним ідентифікатором, включаючи питання та варіанти відповідей. Також перевіряється, чи тест захищений кодовим словом. Якщо захист увімкнено і користувач ще не ввів кодове слово, його перенаправляють на сторінку введення.

```
public IActionResult Index(int id)
{
    var test = dbContext.Tests
        .Include(t => t.Questions)
        .ThenInclude(q => q.AnswerOptions)
        .FirstOrDefault(t => t.Id == id);

    if(test.Passcode != null)
    {
        var testAccess = HttpContext.Session.GetInt32($"TestAccess_{id}");
        if (testAccess != 1)
        {
            return RedirectToAction("EnterPasscode", new { id });
        }
    }

    if (test == null)
    {
        return NotFound("Тест не знайдено.");
    }

    return View(test);
}
```

Цей фрагмент обробляє ситуації, коли тест захищений кодовим словом. Спочатку користувачеві показується форма для введення кодового слова. Якщо код введено правильно, доступ до тесту зберігається в сесії.

```
public IActionResult EnterPasscode(int id)
{
    var test = dbContext.Tests.FirstOrDefault(t => t.Id == id);
    if (test == null)
    {
        return NotFound("Тест не знайдено.");
    }

    return View(new TestPasscodeViewModel { TestId = id });
}
```

Перевірка кодового слова:

```
[HttpPost]
public IActionResult VerifyPasscode(TestPasscodeViewModel testPasscodeViewModel)
{
    var test = dbContext.Tests.FirstOrDefault(t => t.Id ==
testPasscodeViewModel.TestId);
    if (test == null)
    {
```

```

        return NotFound("Тест не знайдено.");
    }

    if (!string.IsNullOrEmpty(test.Passcode) && test.Passcode !=
testPasscodeViewModel.Passcode)
    {
        ModelState.AddModelError("", "Неправильне кодове слово.");
        return View("EnterPasscode", new TestPasscodeViewModel { TestId =
testPasscodeViewModel.TestId });
    }

    HttpContext.Session.SetInt32($"TestAccess_{testPasscodeViewModel.TestId}", 1);
    return RedirectToAction("Index", new { id = testPasscodeViewModel.TestId });
}

```

Цей фрагмент приймає відповіді користувача, перевіряє правильність кожної з них, обчислює бали для кожного питання і загальну оцінку.

1. Завантажуються дані про тест і правильні відповіді на запитання.
2. Оцінка кожного питання базується на складності (Difficulty) та практичній цінності (PracticalValue). Для обчислення використовується функція CalculateQuality, що реалізована через Python-скрипт із нечіткою логікою.
3. Загальний бал обчислюється як відсоткове співвідношення отриманих балів до максимально можливих.
4. Усі відповіді та результат тесту зберігаються в базі даних.

Кожне питання оцінюється за допомогою формули, яка враховує його складність і практичну цінність. Якщо відповідь правильна, до загального балу додається результат розрахунку. Оцінка округлюється до двох знаків після коми.

```

double totalScore = 0;
double maxScores = 0;
var totalQuestions = test.Questions.Count;

var answers = new List<Answer>();

foreach (var answer in submission.Answers)
{
    var question = test.Questions.FirstOrDefault(q => q.Id == answer.QuestionId);

    if (question == null)
        continue;

    bool isCorrect = answer.SelectedAnswerId == question.CorrectAnswerId;
    var quality = CalculateQuality(question.Difficulty, question.PracticalValue);
    maxScores += quality;

    if (isCorrect)
    {
        totalScore += quality;
    }

    answers.Add(new Answer
    {
        IsCorrect = isCorrect,

```

```

        SelectedAnswerId = answer.SelectedAnswerId,
        QuestionId = question.Id,
        Score = isCorrect ? Math.Round(quality, 2) : 0,
    });
}

```

Для виклику Python скрипта і розрахунку якості питання використовується такий код:

```

string scriptPath = Path.Combine(Directory.GetCurrentDirectory(), "Scripts",
"fuzzy_logic.py");

var process = new Process
{
    StartInfo = new ProcessStartInfo
    {
        FileName = "python",
        Arguments = $"\"{scriptPath}\" {difficulty} {practicalValue}",
        RedirectStandardOutput = true,
        RedirectStandardError = true,
        UseShellExecute = false,
        CreateNoWindow = true
    }
};

process.Start();

string output = process.StandardOutput.ReadToEnd();
string error = process.StandardError.ReadToEnd();
process.WaitForExit();

```

Результат тесту, включаючи відповіді та загальний бал, додається до бази даних і доступний для перегляду.

```

var result = new TestResult
{
    TestId = test.Id,
    StudentName = submission.StudentName,
    Answers = answers,
    Score = Math.Round(totalScore / maxScores * 100, 2),
};

dbContext.TestResults.Add(result);
dbContext.SaveChanges();

return RedirectToAction("Result", new { id = result.Id });

```

Цей фрагмент завантажує дані про результати тесту, включаючи деталі кожної відповіді, для відображення користувачеві.

```

public IActionResult Result(int id)
{
    var result = dbContext.TestResults
        .Include(x => x.Test)
        .Include(x => x.Answers).ThenInclude(x => x.Question)
        .Include(x => x.Answers).ThenInclude(x => x.SelectedAnswer)
        .Where(r => r.Id == id)
        .FirstOrDefault();

    if (result == null)
        return NotFound("Тест не знайдено.");

    return View(result);
}

```

```
}
```

Для більш кращого користувацького досвіду значну частину розробки було приділено і розробці клієнтського інтерфейсу, а саме розробці скриптів для більшої інтерактивності сторінок, а також клієнтської варіації.

На формі створення тесту був доданий скрипт для перевірки, що кожне питання має лише одну правильну відповідь:

```
document.getElementById('submit-btn').addEventListener('click', function (event) {
  const questions = document.querySelectorAll('.question');
  for (const question of questions) {
    const correctAnswers = question.querySelectorAll('.is-correct-
checkbox:checked');
    if (correctAnswers.length !== 1) {
      event.preventDefault();
      alert('Кожне питання має мати тільки одну правильну відповідь.');
```

На сторінці деталей тесту засобами JavaScript було реалізовано функціонал для відображення графіків:

```
document.addEventListener('DOMContentLoaded', function () {
  const ctx = document.getElementById('difficultyChart').getContext('2d');
  const difficultyLabels = @Html.Raw(Json.Serialize(Model.DifficultyStats.Select(s
=> s.Key)));
  const correctPercentages = @Html.Raw(Json.Serialize(Model.DifficultyStats.Select(s
=> s.Value)));

  new Chart(ctx, {
    type: 'bar',
    data: {
      labels: difficultyLabels,
      datasets: [{
        label: 'Відсоток правильних відповідей',
        data: correctPercentages,
        backgroundColor: 'rgba(54, 162, 235, 0.6)',
        borderColor: 'rgba(54, 162, 235, 1)',
        borderWidth: 1
      }]
    },
    options: {
      scales: {
        y: {
          beginAtZero: true,
          max: 100
        }
      }
    }
  });
});
```

А на сторінці проходження тесту був доданий скрипт, що відповідає за відображення таймера і автоматично відправляє форму після його завершення:

```
function updateTimer() {
  const minutes = Math.floor(timeLeft / 60);
```

```

const seconds = timeLeft % 60;
timerElement.innerText = `${minutes}:${seconds.toString().padStart(2, '0')}`;

if (timeLeft <= 0) {
  clearInterval(timerInterval);
  document.querySelector('form').submit();
}

timeLeft--;
}

let timerInterval;

startButton.addEventListener('click', function () {
  startSection.style.display = 'none';
  testSection.style.display = 'block';
  questions[0].style.display = 'block';

  if (durationInMinutes > 0) {
    timerInterval = setInterval(updateTimer, 1000);
    updateTimer();
  }
});

```

На цій же сторінці був реалізований скрипт для відображення статусу проходження тесту:

```

function updateQuestionStatus(index) {
  const row = rows[index];
  const badge = row.querySelector('.badge');
  if (isQuestionAnswered(index)) {
    badge.textContent = "Пройдено";
    badge.classList.replace('bg-secondary', 'bg-success');
  } else {
    badge.textContent = "Не пройдено";
    badge.classList.replace('bg-success', 'bg-secondary');
  }
}

function showQuestion(index) {
  questions.forEach((q, i) => q.style.display = i === index ? 'block' : 'none');
  rows.forEach((row, i) => row.classList.toggle('table-active', i === index));

  prevButton.style.display = index > 0 ? 'inline-block' : 'none';
  nextButton.style.display = index < questions.length - 1 ? 'inline-block' : 'none';
  submitButton.style.display = index === questions.length - 1 ? 'inline-block' : 'none';
  nextButton.textContent = index < questions.length - 1 ? "Наступне питання" : "Завершити тест";
  updateProgressBar();
}

function updateProgressBar() {
  const answeredQuestions = Array.from(questions).filter((q, i) => isQuestionAnswered(i)).length;
  const progress = (answeredQuestions / questions.length) * 100;
  progressBar.style.width = `${progress}%`;
  progressBar.setAttribute('aria-valuenow', progress);
  progressBar.innerText = `${Math.round(progress)}%`;
}

```

Висновки до другого розділу

У другому розділі було здійснено проектування та розробку моделі тестування знань із використанням методів нечіткої логіки. Спочатку було сформовано модель оцінювання результатів тестування, яка враховує складність запитань, їх практичну цінність та правильність відповідей. Далі розроблено архітектуру веб-орієнтованої системи, яка забезпечує гнучкість, інтерактивність та відповідність сучасним вимогам до подібних систем. Завершальним етапом стала реалізація цієї системи із застосуванням розробленої моделі нечіткого оцінювання, що дозволяє об'єктивніше оцінювати рівень знань тестованих. Представлений підхід демонструє переваги інтеграції нечіткої логіки в освітні технології.

РОЗДІЛ 3. ПОРЯДОК РОБОТИ З ДОДАТКОМ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

3.1 Порядок роботи з додатком

Після запуску додатка користувач потрапляє на головну сторінку, на якій є список існуючих тестів з можливістю пошуку та функціоналом пагінації.

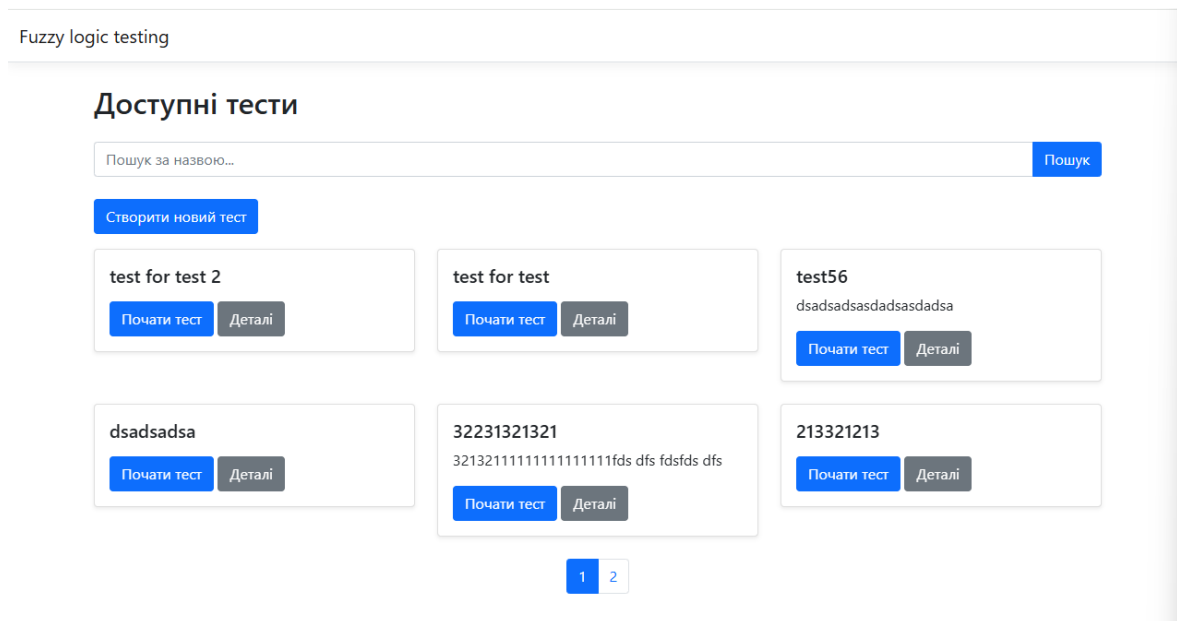


Рис. 3.1. Головна сторінка додатку

При створенні тесту відображається форма, що має всі поля тесту і можливість додавати питання:

Рис. 3.2. Сторінка створення тесту

Після додавання питання відображається відповідний блок, що також має в собі функціонал для давання варіанту відповіді на питання:

Питання

Текст питання:

Складність:

Практичне значення:

Варіанти відповідей

Текст відповіді:

☐ Правильна відповідь

Текст відповіді:

☐ Правильна відповідь

Додати відповідь

Рис. 3.3. Додавання питань та відповідей

Форма має валідацію для обов'язкових полів і також на те, щоб кожне питання мало одну і тільки одну відповідь

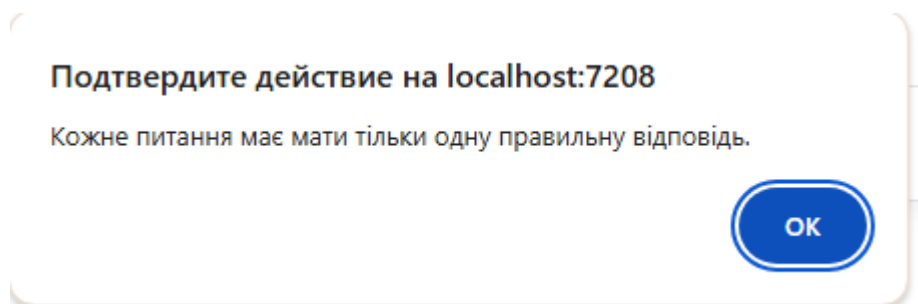


Рис. 3.4. Валідація наявності відповіді

Для початку тестування користувач має ввести кодове слово, що зазначалось при створенні тесту, якщо воно є (для проходження та перегляду деталей тесту є два різних кодових слова).

Введіть кодове слово

- Неправильне кодове слово.

Кодове слово:

•

Підтвердити

Рис. 3.5 Форма введення кодового слова

Після успішного введення кодового слова або його відсутності користувач потрапляє на сторінку з деталями тесту, з якими він може ознайомитись перед проходженням:

Тест: Початкова математика

Виділено часу: 40 хв

Всього питань: 5

Почати тестування

Рис. 3.6. Деталі тесту

Після початку тестування запускається таймер, який оновлюється кожну секунду, а користувач має ввести своє ім'я та дати відповіді на питання. Відображається лише одне питання, для навігації між питаннями використовуються кнопки або таблиця:

Тест: Початкова математика

Виділено часу: 40 хв

Всього питань: 5

Час, що залишився: 39:43

Ваше ім'я:

60%

Скільки сторін в трикутника:

- ☐ 2
☐ 3
☐ 4

Попереднє питання

Завершити тест

Навігація між питаннями

№	Питання	Статус
1	Після того як мама віддала бабусі 7 яєць, залишилося 12. Скільки яєць було спочатку?	Пройдено
2	$10+5=$	Не пройдено
3	менше на 3 означає:	Пройдено
4	якою мірою можна виміряти масу кота:	Пройдено
5	Скільки сторін в трикутника:	Не пройдено

Рис. 3.7. Сторінка проходження тестування

Після завершення тесту користувач потрапляє на сторінку з результатами, де може побачити свою кінцеву оцінку, а також кількість і відсоток правильних відповідей і таблицю з деталями кожної відповіді, в тому числі і бал за неї:

Результат тестування

Ім'я: Максим

Тест: Початкова математика

Правильних відповідей: 3 / 5 (60%)

Оцінка: 65,72

Деталі питань

#	Питання	Відповідь	Бал
1	Після того як мама віддала бабусі 7 яєць, залишилося 12. Скільки яєць було спочатку?	$2+7=20$ (яєць)	0
2	$10+5=$	16	0
3	менше на 3 означає:	від числа відняти 3	6,02
4	якою мірою можна виміряти масу кота:	кг	6,42
5	Скільки сторін в трикутника:	3	3,86

Рис. 3.8. Сторінка результатів тесту

Користувач, що створював тест зі свого боку має доступ до деталей тесту за допомогою окремого кодового слова. Дана сторінка має корисну інформацію, що дозволить тому, хто створив тест краще проаналізувати питання та обізнаність учнів щодо них. Спочатку на цій сторінці відображається список питань з їх даними та відсотком правильних відповідей. Дана інформація може допомогти користувачу оцінити, чи правильні характеристики він надав цьому питанню.

Початкова математика			
Опис: Тест для учнів початкових класів			
Всього питань: 5			
Час на тестування: 40 хв.			
Список питань			
#	Питання	Складність	Відсоток правильних відповідей
1	Після того як мама віддала бабусі 7 яєць, залишилося 12. Скільки яєць було спочатку?	3	66,67%
2	$10+5=$	2	66,67%
3	менше на 3 означає:	4	66,67%
4	якою мірою можна виміряти масу кота:	5	100%
5	Скільки сторін в трикутника:	2	100%

Рис. 3.9. Сторінка деталей тесту – статистика питань

Далі на сторінці знаходяться графіки та діаграми, що допомагають користувачеві краще оцінити загальну картинку, як і складової тесту, так і з тим, як з ним впорались учні:

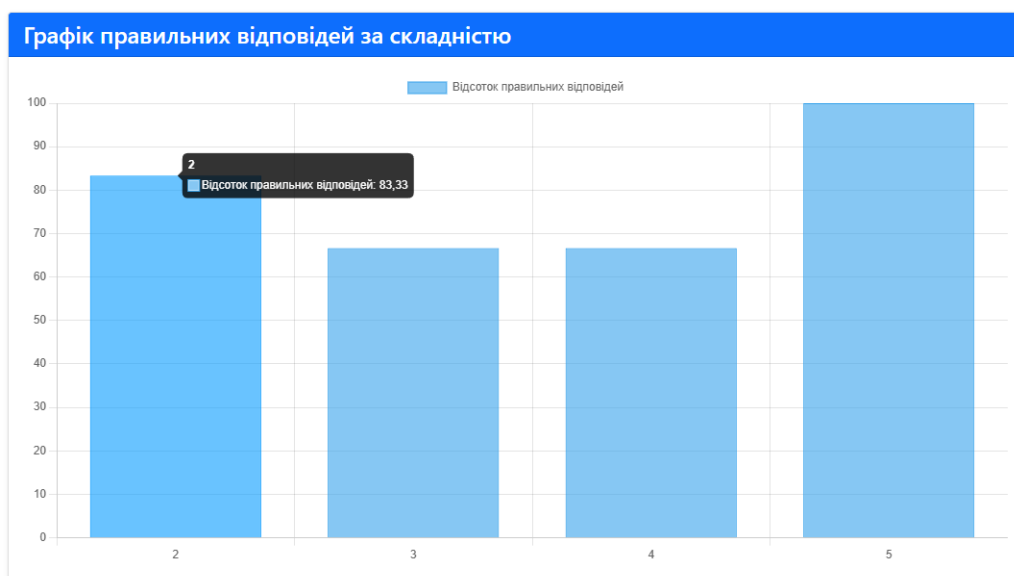


Рис. 3.10. Графік правильних відповідей за складністю

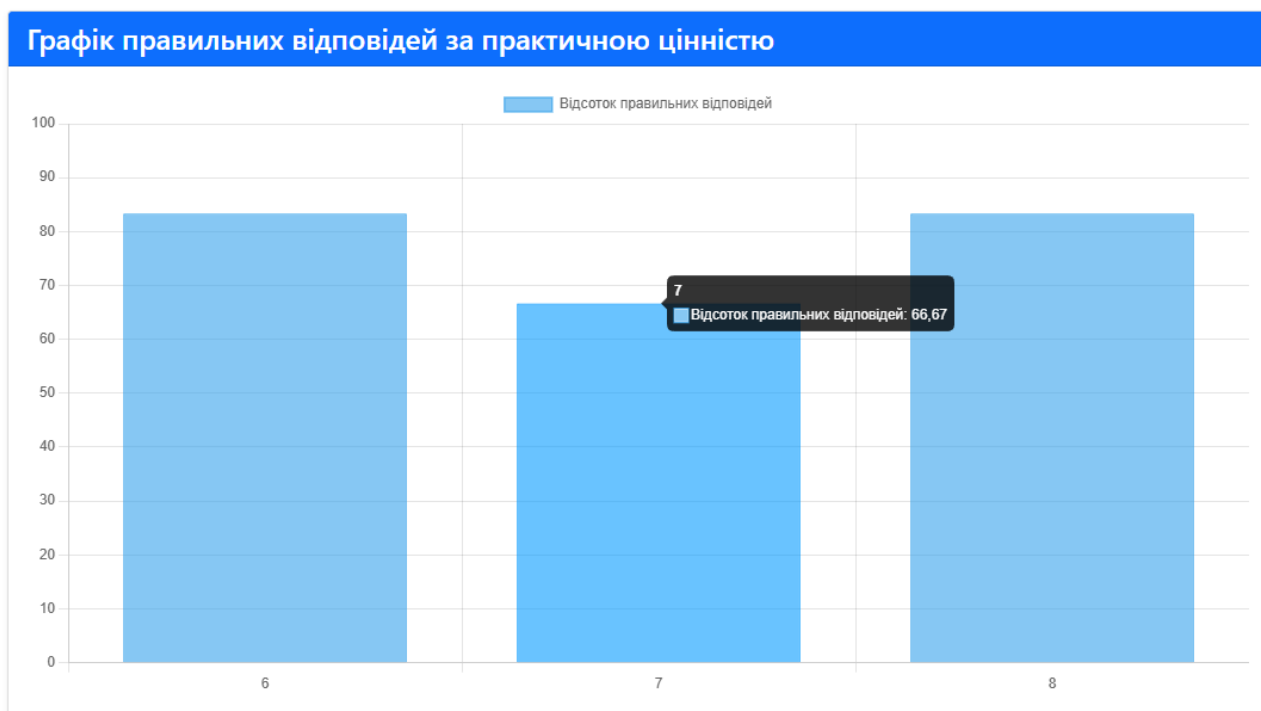


Рис. 3.11. Графік правильних відповідей за практичною цінністю

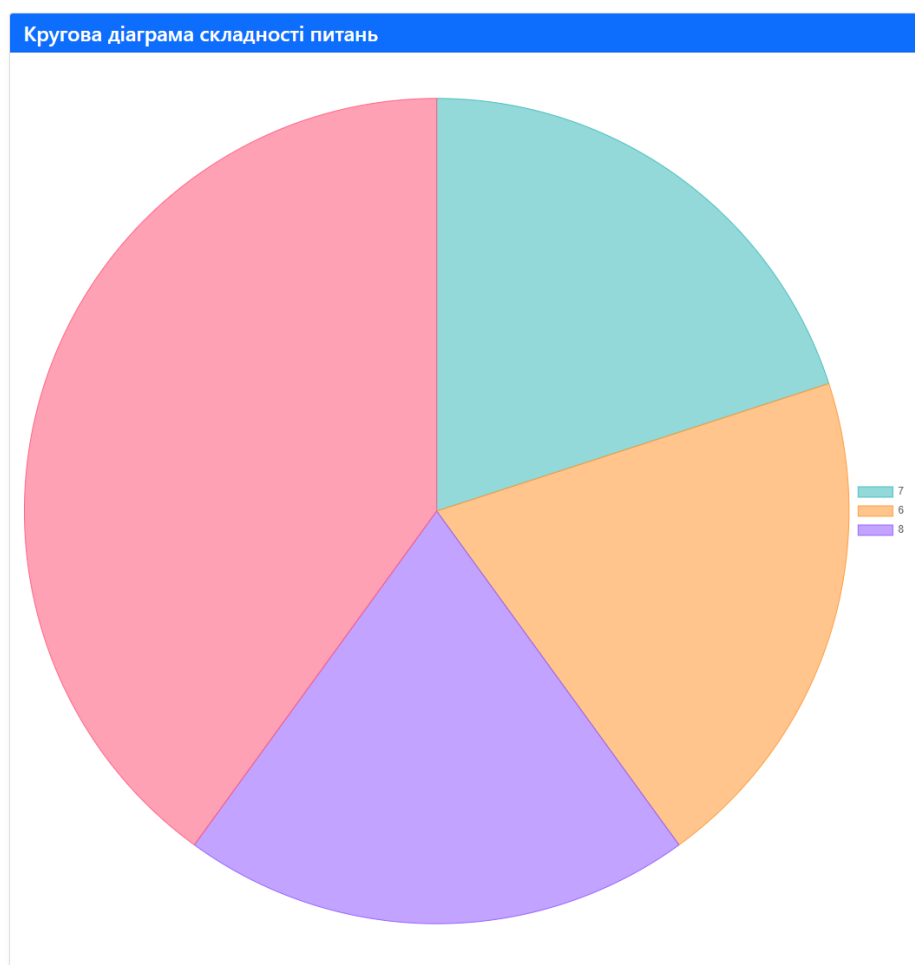


Рис. 3.12. Кругова діаграма складності питань

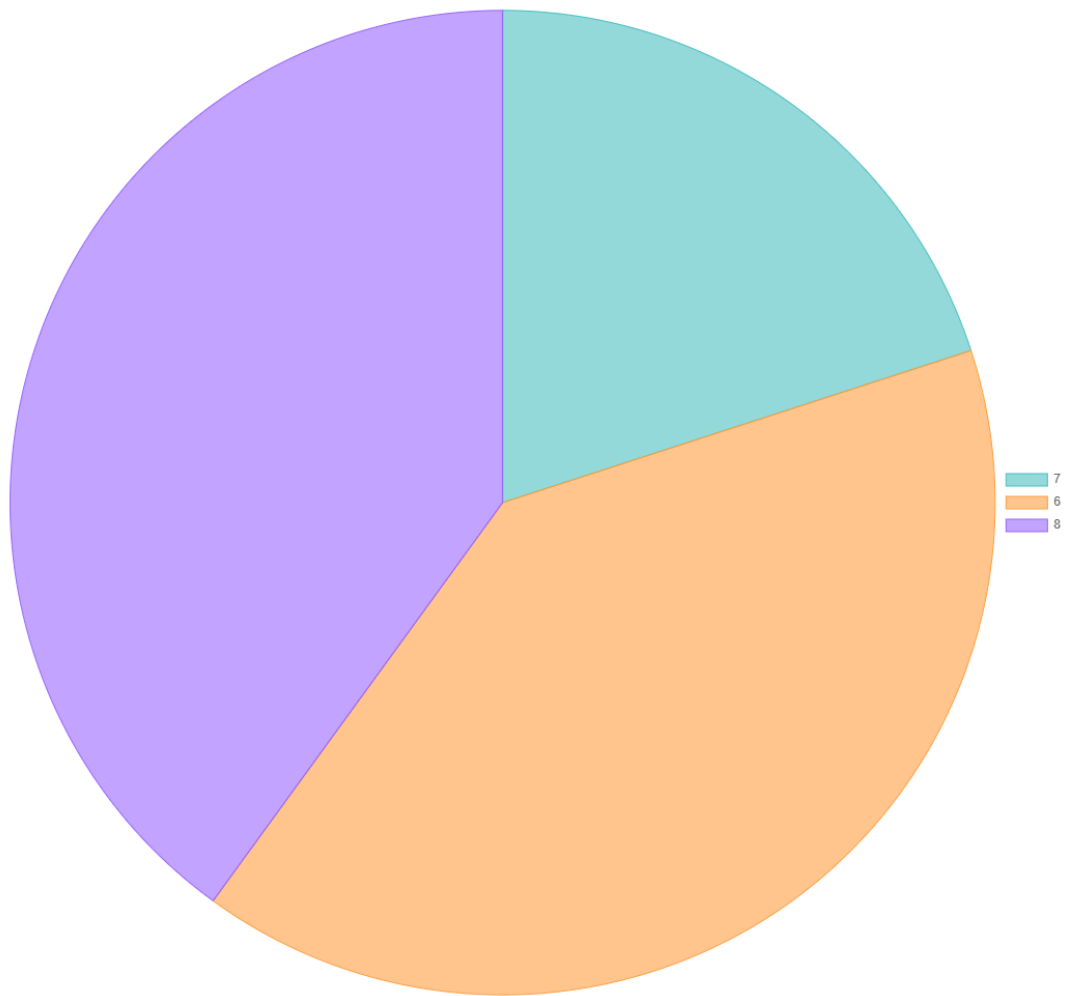


Рис. 3.13. Кругова діаграма практичної цінності питань

В кінці сторінки користувач бачить таблицю з інформацією про останні проходження даного тесту. Тут міститься вся загальна інформація щодо результатів, а при потребі користувач може натиснути кнопку «Деталі» і більш ґрунтовно ознайомитись з результатом конкретного користувача.

Останні проходження

Ім'я	Оцінка	Правильних відповідей	
Іван	100	5 / 5 (100 %)	Деталі
Петро	75,72	4 / 5 (80 %)	Деталі
Максим	65,72	3 / 5 (60 %)	Деталі

Рис. 3.14. Таблиця проходження тесту

3.2. Оцінка роботи системи та перспективи використання

Розроблена веб-орієнтована система тестування знань із використанням методів нечіткої логіки демонструє ефективність у підході до оцінювання результатів тестування. Одним із ключових аспектів цієї системи є адаптивність до різних рівнів складності питань і врахування практичної цінності відповідей. У класичних тестувальних системах оцінка базується виключно на кількості правильних відповідей, тоді як впровадження нечіткої логіки дозволяє враховувати багатфакторні параметри, такі як складність питання, практична значущість і часткову правильність відповідей.

Дані, отримані зі сторінки аналізу тестування, свідчать про те, що система забезпечує більш гнучкий підхід до оцінювання. Наприклад, навіть за умови однакової кількості правильних відповідей, учасники тесту можуть отримувати різні бали залежно від того, наскільки складні чи значущі були ці питання. Це сприяє більш справедливій оцінці знань і навичок.

Серед сильних сторін системи можна виділити:

- Врахування різних характеристик питань, що робить оцінку результатів більш багатогранною.
- Надання докладної статистики щодо відповідей, що дозволяє аналізувати слабкі місця та успіхи користувачів.
- Зручність і наочність у використанні завдяки інтерфейсу, який містить графіки, таблиці та кругові діаграми.

Використання нечіткої логіки впливає на оцінювання таким чином, що результати тестів стають менш залежними від чіткої відповіді "правильно/неправильно". Наприклад, у випадках, коли відповідь є частково правильною або демонструє розуміння концепції, система нараховує часткові бали. Це особливо корисно для складних питань, де простий двійковий підхід був би несправедливим.

На основі статистики з результатів тестування можна зробити висновок, що учасники, які мають високий рівень практичного застосування знань, отримують вищі бали навіть за меншої кількості правильних відповідей. Це

демонструє можливість адаптації системи до специфічних потреб тестування, наприклад, у професійній чи академічній сфері.

Система тестування знань, яка використовує методи нечіткої логіки, пропонує низку функціональних можливостей, що забезпечують її універсальність, гнучкість і здатність адаптуватися до потреб користувачів у різних сферах.

Однією з основних переваг системи є гнучкість. Вона може бути налаштована для проведення різноманітних видів тестування: від академічних тестів для студентів, які перевіряють рівень знань із певної дисципліни, до сертифікаційних іспитів, спрямованих на оцінку професійних навичок. Це дозволяє використовувати систему в освітніх закладах, корпоративному середовищі та навіть у наукових дослідженнях.

Ще однією важливою можливістю є розширення критеріїв оцінювання. Система дозволяє додавати нові параметри для оцінки результатів тестування. Наприклад, можна враховувати час, витрачений на виконання завдання, що є показником швидкості прийняття рішень, або рівень впевненості користувача в обраній відповіді, який демонструє його суб'єктивне сприйняття власних знань. Це робить оцінювання більш детальним та індивідуалізованим.

Для інтеграції з іншими навчальними платформами передбачено можливість використання API. Це дозволяє зв'язувати систему з популярними системами керування навчанням, як-от Moodle. Завдяки цьому тести, створені в системі, можуть бути безпосередньо доступні в середовищах, знайомих студентам і викладачам, що спрощує процес навчання й оцінювання.

Система також підтримує моделювання складності тестів. Використовуючи методи нечіткої логіки, можна генерувати адаптивні тести, які автоматично підлаштовуються під рівень знань користувача. Наприклад, якщо студент дає правильні відповіді на легкі запитання, система може поступово збільшувати складність наступних завдань. Це дозволяє оцінити реальний рівень знань і сприяє ефективнішому навчанню, адже студенти отримують завдання, відповідні їхньому поточному рівню.

Завдяки цим можливостям система є не лише інструментом для тестування знань, а й ефективним засобом для підвищення рівня навчання, аналізу прогресу та впровадження новітніх технологій у процес навчання й оцінювання.

Розроблена система тестування знань демонструє високу ефективність у процесі оцінювання відповідей користувачів. Вона враховує такі параметри, як складність питань та їхня практична цінність, що дозволяє отримувати більш об'єктивні результати. Представлені графіки та діаграми свідчать про важливість використання нечіткої логіки для аналізу відповідей.

За результатами тестувань видно, що високий відсоток правильних відповідей залежить не тільки від складності завдань, але й від їхньої практичної цінності. Наприклад, питання з низьким рівнем практичної цінності, хоч і є складними, часто отримують менш точну оцінку через суб'єктивність стандартних підходів. З іншого боку, нечітка логіка дозволяє збалансувати ці фактори, застосовуючи вагові коефіцієнти для об'єктивного аналізу.

Графіки, що відображають правильні відповіді за складністю, демонструють тенденцію до більшої кількості правильних відповідей на питання середньої складності. У свою чергу, питання з високою практичною цінністю показують стабільно високі результати, незалежно від їхньої складності. Це свідчить про те, що користувачі більше мотивовані відповідати на такі питання коректно.

Крім того, система дозволяє ефективно виявляти слабкі сторони користувачів, що є корисним для подальшого вдосконалення навчальних програм. Наприклад, аналіз результатів останніх проходжень показав, що навіть користувачі з високими показниками мають проблеми з окремими типами завдань, які можна врахувати при розробці додаткових матеріалів.

Інтеграція методів нечіткої логіки в системи тестування знань відкриває широкі можливості для їх використання в різних сферах. Однією з головних переваг такого підходу є його адаптивність, яка дозволяє враховувати індивідуальні особливості кожного користувача. Це особливо важливо у сфері

освіти, де різний рівень підготовки учнів вимагає персоналізованого підходу до оцінювання.

У шкільній освіті система може використовуватися для швидкої оцінки знань учнів, особливо в умовах дистанційного навчання. Вищі навчальні заклади можуть впроваджувати такі системи для автоматизації іспитів, забезпечуючи об'єктивність оцінювання. Застосування нечіткої логіки дозволить знизити вплив людського фактора, що особливо важливо при великій кількості студентів.

У професійній освіті така система стане корисною для оцінки знань працівників під час сертифікаційних курсів або тренінгів. Наприклад, компанії зможуть проводити регулярну перевірку знань своїх співробітників, адаптуючи питання відповідно до змін у професійних вимогах.

Додатково, система може бути застосована у галузях, де необхідно швидко та якісно перевіряти знання великих груп людей, таких як рекрутинг або корпоративне навчання. Інтеграція таких рішень з сучасними платформами, як-от ASP.NET, дозволяє створювати масштабовані веб-додатки, які будуть доступні для використання з будь-якого пристрою.

У перспективі можливим є вдосконалення системи через впровадження штучного інтелекту для аналізу даних. Це дозволить ще більш точно оцінювати відповіді та прогнозувати можливі результати тестування. Також можливим є розширення функціоналу через інтеграцію з іншими інструментами, такими як системи управління навчанням (LMS), що забезпечить комплексне рішення для освітніх установ.

Таким чином, використання нечіткої логіки у системах тестування знань є не лише актуальним, але й необхідним для підвищення ефективності оцінювання та адаптації до сучасних умов. Розроблена система має значний потенціал для впровадження в освітніх і професійних сферах, забезпечуючи точність, об'єктивність та гнучкість у процесі оцінювання.

Висновки до третього розділу.

У третьому розділі було описано порядок роботи з розробленим додатком та проведено оцінку ефективності системи. Детально розглянуто функціонал програми, зокрема, механізм створення тестів, аналізу результатів та інтерпретації даних. Система довела свою ефективність у забезпеченні об'єктивного оцінювання знань завдяки використанню методів нечіткої логіки. Результати тестування підтверджують потенціал системи для використання у сфері освіти, професійного навчання та інших галузях, де важлива адаптивність і точність оцінювання.

ВИСНОВКИ

У рамках виконання дипломної роботи було розглянуто використання методів нечіткої логіки для оцінювання знань у системах тестування, спроектовано та реалізовано веб-орієнтовану систему тестування знань. Дослідження дозволило розкрити потенціал нечіткої логіки для створення гнучких та адаптивних систем оцінювання, які враховують неоднозначності та суб'єктивність у процесі тестування.

Перший етап роботи полягав у вивченні існуючих підходів до тестування знань і аналізі їх недоліків. Зокрема, традиційні системи оцінювання часто виявляються занадто жорсткими, не враховують варіативність відповідей і контекст. Застосування нечіткої логіки дозволило усунути ці недоліки завдяки використанню лінгвістичних змінних та функцій належності для моделювання процесу оцінювання.

У другому розділі було розроблено математичну модель оцінювання результатів тестування з використанням нечіткої логіки. Запропоновано систему правил на основі бази знань, що враховує рівень складності питань, вагові коефіцієнти відповідей і часові обмеження. Веб-орієнтована система тестування була спроектована з урахуванням сучасних вимог до веб-додатків: адаптивності, продуктивності та зручності використання. Основою реалізації системи став стек технологій ASP.NET Core, Entity Framework для роботи з базою даних, а також JavaScript для інтерактивності.

На практичному етапі була реалізована веб-система, яка дозволяє створювати тестові завдання, проходити тести та отримувати результати з адаптивним оцінюванням. Результати тестування виводяться у вигляді зручного звіту, що враховує рівень підготовки користувача та специфіку питань. Було проведено апробацію системи, яка підтвердила її ефективність та зручність для користувачів.

Таким чином, розроблена система демонструє, що методи нечіткої логіки є перспективними для впровадження у тестові системи, оскільки дозволяють адаптивно оцінювати знання, враховуючи різні фактори та умови. Отримані

результати можуть бути корисними для подальших досліджень і розробок у галузі автоматизованого тестування, а також знайти застосування у навчальних закладах, корпоративному навчанні та інших сферах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zakharchuk M. S., Vakaliuk T. A. ARCHITECTURE OF A WEB-BASED TESTING SYSTEM USING FUZZY LOGIC // Збірник тез доповідей II Міжнар. наук.-практич. конфер. «Інновації та перспективні шляхи розвитку інформаційних технологій» (06 груд. 2023 р., м. Черкаси), Черкас. держ. технол. ун-т. – Черкаси : ЧДТУ, 2023. – 202 с.
2. Zakharchuk M. S., Chyzhmotria O. V. DESIGNING A WEB-BASED FUZZY LOGIC TESTING SYSTEM // Тези доповідей VI Всеукраїнської науково-технічної конференції «Комп'ютерні технології: інновації, проблеми, рішення» (29-30 листопада 2023 р., м. Житомир), Державний університет «Житомирська політехніка», 2023. – 237 с.
3. Zakharchuk M. S., Chyzhmotria O. V. Application of Fuzzy Logic Methods to the Knowledge Testing Systems // XVII міжнародна науково-практична конференція магістрантів на аспірантів (28-30 листопада 2023 р., м. Харків), Національний технічний університет «Харківський політехнічний інститут», 2023. – 3 с.
4. Zadeh L.A. Fuzzy Sets / Information and Control. – 1965.
5. Mamdani E.H., Assilian S. An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller // International Journal of Man-Machine Studies. – 1975.
6. Klir G.J., Yuan B. Fuzzy Sets and Fuzzy Logic: Theory and Applications. – Prentice Hall, 1995. – 574 с.
7. Ross T.J. Fuzzy Logic with Engineering Applications. – John Wiley & Sons, 2016. – 606 с.
8. Mendel J.M. Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions. – Springer, 2001. – 684 с.
9. ASP.NET Core Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/>
10. Freeman A. Pro ASP.NET Core 6 / A. Freeman. – Apress, 2022. – 1080 с.
11. Rahman H. Entity Framework Core in Action. – Manning Publications, 2018. – 530 с.

12. Python Software Foundation / Python 3.10 Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.python.org/3/>
13. McKinney W. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. – O'Reilly Media, 2017. – 550 с.
14. Fowler M. Refactoring: Improving the Design of Existing Code. – Addison-Wesley, 2018. – 448 с.
15. Bishop C.M. Pattern Recognition and Machine Learning. – Springer, 2006. – 738 с.
16. Freeman A. Pro C# 10 and the .NET 6 Platform. – Apress, 2022. – 1240 с.
17. Dey A. Learning ASP.NET Core MVC Programming. – Packt Publishing, 2017. – 350 с.
18. Esposito D. Modern Web Development with ASP.NET Core 3. – Packt Publishing, 2020. – 570 с.
19. Sugeno M. Fuzzy Control. – Academic Press, 1985. – 386 с.
20. Lee C.C. Fuzzy Logic in Control Systems: Fuzzy Logic Controller – Part I // IEEE Transactions on Systems, Man, and Cybernetics. – 1990.
21. Kurniawan B. Introduction to ASP.NET Core Razor Pages. – Brainy Software, 2019. – 234 с.
22. Rumelhart D.E., McClelland J.L. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. – MIT Press, 1986. – 567 с.
23. Web.dev / ASP.NET Performance [Електронний ресурс] – Режим доступу до ресурсу: <https://web.dev/aspnet-performance>
24. Microsoft Docs / ASP.NET Entity Framework [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/ef/>
25. Gabbard R. SQL Server Programming for Developers. – Wiley, 2020. – 421 с.
26. PostgreSQL.org / Офіційна документація [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org/docs/>
27. Web.dev / Google Lighthouse [Електронний ресурс] – Режим доступу до ресурсу: <https://web.dev/lighthouse>

28. Heričko T. Towards Representative Web Performance Measurements with Google Lighthouse / Šumak B., Brdник S. – 7th Student Computer Science Research Conference, 2021.
29. Yager R.R., Filev D.P. Essentials of Fuzzy Modeling and Control. – Wiley, 1994. – 388 с.
30. Talukdar P., Ghosh M. Student Performance Evaluation Using Fuzzy Sets // Journal of Artificial Intelligence Research. – 2007.
31. Kaushik A. Web Analytics: An Hour a Day. – Wiley, 2007. – 443 с.
32. Montemurro M. Django for Professionals: Production Websites with Python & Django. – 2021. – 370 с.
33. Vihang N. Mastering Entity Framework Core 6. – Packt Publishing, 2022. – 384 с.
34. O'Reilly Media / Modern Python CookBook [Електронний ресурс] – Режим доступу до ресурсу: <https://www.oreilly.com/library/view/modern-python-cookbook/>
35. Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press, 2016. – 775 с.
36. Django Project / Офіційна документація [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.djangoproject.com/>
37. Duckett J. HTML and CSS: Design and Build Websites. – Wiley, 2014. – 512 с.
38. Kamali A. Modern Data Science Tools in Python. – Springer, 2020. – 282 с.
39. Cohen D.S. Database Design for Mere Mortals. – Addison-Wesley, 2013. – 640 с.
40. Web.dev / Testing Web Performance [Електронний ресурс] – Режим доступу до ресурсу: <https://web.dev/articles/testing-performance>

ДОДАТКИ

Лістинг fuzzy_logic.py (математична модель обрахунку оцінки):

```

import sys
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt

difficulty = ctrl.Antecedent(np.arange(0, 11, 1), 'difficulty')
practical_value = ctrl.Antecedent(np.arange(0, 11, 1), 'practical_value')
quality_coefficient = ctrl.Consequent(np.arange(0, 11, 1),
'quality_coefficient')

difficulty['low'] = fuzz.trapmf(difficulty.universe, [0, 0, 2, 4])
difficulty['medium'] = fuzz.trimf(difficulty.universe, [3, 5, 7])
difficulty['high'] = fuzz.trapmf(difficulty.universe, [6, 8, 10, 10])

practical_value['low'] = fuzz.trimf(practical_value.universe, [0, 0, 10])
practical_value['medium'] = fuzz.trimf(practical_value.universe, [0, 5, 10])
practical_value['high'] = fuzz.trimf(practical_value.universe, [0, 10, 10])

quality_coefficient['low'] = fuzz.trapmf(quality_coefficient.universe, [0, 0, 2,
6])
quality_coefficient['medium'] = fuzz.trapmf(quality_coefficient.universe, [5, 6,
7, 9])
quality_coefficient['high'] = fuzz.trapmf(quality_coefficient.universe, [8, 9,
10, 10])

rule1 = ctrl.Rule(difficulty['low'] & practical_value['low'],
quality_coefficient['low'])
rule2 = ctrl.Rule(difficulty['low'] & practical_value['medium'],
quality_coefficient['low'])
rule3 = ctrl.Rule(difficulty['low'] & practical_value['high'],
quality_coefficient['medium'])
rule4 = ctrl.Rule(difficulty['medium'] & practical_value['low'],
quality_coefficient['low'])
rule5 = ctrl.Rule(difficulty['medium'] & practical_value['medium'],
quality_coefficient['medium'])
rule6 = ctrl.Rule(difficulty['medium'] & practical_value['high'],
quality_coefficient['high'])
rule7 = ctrl.Rule(difficulty['high'] & practical_value['low'],
quality_coefficient['medium'])
rule8 = ctrl.Rule(difficulty['high'] & practical_value['medium'],
quality_coefficient['high'])
rule9 = ctrl.Rule(difficulty['high'] & practical_value['high'],
quality_coefficient['high'])

quality_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6,
rule7, rule8, rule9])
quality_simulation = ctrl.ControlSystemSimulation(quality_ctrl)

def calculate_quality_coefficient(difficulty_level, practical_value_level):
    quality_simulation.input['difficulty'] = difficulty_level
    quality_simulation.input['practical_value'] = practical_value_level
    quality_simulation.compute()
    return quality_simulation.output['quality_coefficient']

def plot_membership_functions():
    x_difficulty = np.linspace(0, 10, 100)
    x_significance = np.linspace(0, 10, 100)
    x_quality = np.linspace(0, 10, 100)

```

```

difficulty_low = fuzz.trapmf(x_difficulty, [0, 0, 2, 4])
difficulty_medium = fuzz.trimf(x_difficulty, [3, 5, 7])
difficulty_high = fuzz.trapmf(x_difficulty, [6, 8, 10, 10])

significance_low = fuzz.trimf(x_significance, [0, 0, 10])
significance_medium = fuzz.trimf(x_significance, [0, 5, 10])
significance_high = fuzz.trimf(x_significance, [0, 10, 10])

quality_low = fuzz.trapmf(x_quality, [0, 0, 2, 6])
quality_medium = fuzz.trapmf(x_quality, [5, 6, 7, 9])
quality_high = fuzz.trapmf(x_quality, [8, 9, 10, 10])

fig, axs = plt.subplots(3, 1, figsize=(8, 12))

axs[0].plot(x_difficulty, difficulty_low, 'b', label='Низька')
axs[0].plot(x_difficulty, difficulty_medium, 'g', label='Середня')
axs[0].plot(x_difficulty, difficulty_high, 'r', label='Висока')
axs[0].set_title('Функції належності для складності')
axs[0].set_xlabel("Складність")
axs[0].set_ylabel("Належність")
axs[0].legend()

axs[1].plot(x_significance, significance_low, 'b', label='Низька')
axs[1].plot(x_significance, significance_medium, 'g', label='Середня')
axs[1].plot(x_significance, significance_high, 'r', label='Висока')
axs[1].set_title('Функції належності для практичного значення')
axs[1].set_xlabel("Практичне значення")
axs[1].set_ylabel("Належність")
axs[1].legend()

axs[2].plot(x_quality, quality_low, 'b', label='Низька')
axs[2].plot(x_quality, quality_medium, 'g', label='Середня')
axs[2].plot(x_quality, quality_high, 'r', label='Висока')
axs[2].set_title('Функції належності для якості питання')
axs[2].set_xlabel("Якість питання")
axs[2].set_ylabel("Належність")
axs[2].legend()

plt.tight_layout()
plt.show()

if __name__ == "__main__":
    if len(sys.argv) > 1:
        difficulty = int(sys.argv[1])
        practical_value = int(sys.argv[2])
        result = calculate_quality_coefficient(difficulty, practical_value)
        print(result)
    else:
        plot_membership_functions()

```

Лістинг HomeController.cs (контролер менеджменту тестів):

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using PagedList.Core;
using System.Diagnostics;
using WepApp.Models;
using WepApp.Models.Database;

namespace WepApp.Controllers
{
    public class HomeController(ApplicationDbContext dbContext) : Controller
    {
        public IActionResult Index(string searchQuery, int? page)
        {
            const int pageSize = 6;

            var tests = dbContext.Tests.AsQueryable();

            if (!string.IsNullOrEmpty(searchQuery))
            {
                tests = tests.Where(t => t.Title.Contains(searchQuery));
            }

            var pagedTests = tests.OrderByDescending(x => x.Id).ToPagedList(page
?? 1, pageSize);

            ViewBag.SearchQuery = searchQuery;
            return View(pagedTests);
        }

        public IActionResult Create()
        {
            return View();
        }

        [HttpPost]
        public IActionResult Create(Test test, List<Question> questions)
        {
            if (ModelState.IsValid)
            {
                dbContext.Tests.Add(test);
                dbContext.SaveChanges();
                TempData["SuccessMessage"] = "Тест успішно створено!";
                return RedirectToAction("Index");
            }
            return View(test);
        }

        public IActionResult Details(int id)
        {
            var test = dbContext.Tests
                .Include(t => t.Results).ThenInclude(x =>
x.Answers).ThenInclude(x => x.Question)
                .Include(x => x.Questions)
                .FirstOrDefault(t => t.Id == id);

            if (test == null)
            {
                return NotFound();
            }
        }
    }
}

```

```

        if (test.Passcode != null)
        {
            var testAccess =
HttpContext.Session.GetInt32($"TestAccessAdmin_{id}");
            if (testAccess != 1)
            {
                return RedirectToAction("EnterPasscode", new { id });
            }
        }

var difficultyStats = test.Results.SelectMany(x => x.Answers)
    .GroupBy(q => q.Question.Difficulty)
    .Select(group => new AnswersStat
    {
        Key = group.Key,
        Value = group.Any()
            ? Math.Round(group.Count(a => a.IsCorrect) * 100.0 /
group.Count(), 2)
            : 0
    })
    .OrderBy(stat => stat.Key)
    .ToList();

var practicalValueStats = test.Results.SelectMany(x => x.Answers)
    .GroupBy(q => q.Question.PracticalValue)
    .Select(group => new AnswersStat
    {
        Key = group.Key,
        Value = group.Any()
            ? Math.Round(group.Count(a => a.IsCorrect) * 100.0 /
group.Count(), 2)
            : 0
    })
    .OrderBy(stat => stat.Key)
    .ToList();

var difficultyDistribution = test.Questions.GroupBy(q =>
q.Difficulty)
    .Select(group => new AnswersStat
    {
        Key = group.Key,
        Value = group.Count()
    })
    .OrderBy(d => d.Value)
    .ToList();

var practicalValueDistribution = test.Questions.GroupBy(q =>
q.PracticalValue)
    .Select(group => new AnswersStat
    {
        Key = group.Key,
        Value = group.Count()
    })
    .OrderBy(d => d.Value)
    .ToList();

var questionStats = test.Results.SelectMany(x => x.Answers)
    .GroupBy(q => q.Question.Id)
    .Select(group => new QuestionStat
    {
        Question = group.First().Question,
        CorrectPercentage = group.Any()

```

```

        ? Math.Round(group.Count(a => a.IsCorrect) * 100.0 /
group.Count(), 2)
        : 0
    })
    .OrderBy(stat => stat.Question.Id)
    .ToList();

    var model = new TestDetailsViewModel
    {
        Test = test,
        RecentResults = test.Results.OrderByDescending(r =>
r.Id).Take(10).ToList(),
        DifficultyStats = difficultyStats,
        QuestionStats = questionStats,
        PracticalValueStats = practicalValueStats,
        DifficultyDistribution = difficultyDistribution,
        PracticalValueDistribution = practicalValueDistribution,
    };

    return View(model);
}

public IActionResult EnterPasscode(int id)
{
    var test = dbContext.Tests.FirstOrDefault(t => t.Id == id);
    if (test == null)
    {
        return NotFound("Тест не знайдено.");
    }

    return View(new TestPasscodeViewModel { TestId = id });
}

[HttpPost]
public IActionResult VerifyPasscode(TestPasscodeViewModel
testPasscodeViewModel)
{
    var test = dbContext.Tests.FirstOrDefault(t => t.Id ==
testPasscodeViewModel.TestId);
    if (test == null)
    {
        return NotFound("Тест не знайдено.");
    }

    if (!string.IsNullOrEmpty(test.Passcode) && test.PasscodeAdmin !=
testPasscodeViewModel.Passcode)
    {
        ModelState.AddModelError("", "Неправильне кодове слово.");
        return View("EnterPasscode", new TestPasscodeViewModel { TestId
= testPasscodeViewModel.TestId });
    }

    HttpContext.Session.SetInt32($"TestAccessAdmin_{testPasscodeViewModel.TestId}",
1);

    return RedirectToAction("Details", new { id =
testPasscodeViewModel.TestId });
}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None,
NoStore = true)]
public IActionResult Error()
{

```

```
        return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??  
HttpContext.TraceIdentifier });  
    }  
}
```

Лістинг TestTakingController.cs (контролер проходження тестів):

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Diagnostics;
using WepApp.Models;
using WepApp.Models.Database;

namespace WepApp.Controllers
{
    public class TestTakingController(ApplicationDbContext dbContext) :
    Controller
    {
        public IActionResult Index(int id)
        {
            var test = dbContext.Tests
                .Include(t => t.Questions)
                .ThenInclude(q => q.AnswerOptions)
                .FirstOrDefault(t => t.Id == id);

            if(test.Passcode != null)
            {
                var testAccess =
HttpContext.Session.GetInt32($"TestAccess_{id}");
                if (testAccess != 1)
                {
                    return RedirectToAction("EnterPasscode", new { id });
                }
            }

            if (test == null)
            {
                return NotFound("Тест не знайдено.");
            }

            return View(test);
        }

        public IActionResult EnterPasscode(int id)
        {
            var test = dbContext.Tests.FirstOrDefault(t => t.Id == id);
            if (test == null)
            {
                return NotFound("Тест не знайдено.");
            }

            return View(new TestPasscodeViewModel { TestId = id });
        }

        [HttpPost]
        public IActionResult VerifyPasscode(TestPasscodeViewModel
testPasscodeViewModel)
        {
            var test = dbContext.Tests.FirstOrDefault(t => t.Id ==
testPasscodeViewModel.TestId);
            if (test == null)
            {
                return NotFound("Тест не знайдено.");
            }
        }
    }
}

```

```

        if (!string.IsNullOrEmpty(test.Passcode) && test.Passcode !=
testPasscodeViewModel.Passcode)
        {
            ModelState.AddModelError("", "Неправильне кодове слово.");
            return View("EnterPasscode", new TestPasscodeViewModel { TestId
= testPasscodeViewModel.TestId });
        }

HttpContext.Session.SetInt32($"TestAccess_{testPasscodeViewModel.TestId}", 1);
return RedirectToAction("Index", new { id =
testPasscodeViewModel.TestId });
    }

[HttpPost]
public IActionResult SubmitTest(TestSubmission submission)
{
    var test = dbContext.Tests
        .Where(t => t.Id == submission.TestId)
        .Select(t => new
        {
            t.Id,
            Questions = t.Questions.Select(q => new
            {
                q.Id,
                q.Difficulty,
                q.PracticalValue,
                CorrectAnswerId = q.AnswerOptions.Where(a =>
a.IsCorrect).Select(a => a.Id).FirstOrDefault()
            }).ToList()
        })
        .FirstOrDefault();

    if (test == null)
        return NotFound("Тест не знайдено");

    double totalScore = 0;
    double maxScores = 0;
    var totalQuestions = test.Questions.Count;

    var answers = new List<Answer>();

    foreach (var answer in submission.Answers)
    {
        var question = test.Questions.FirstOrDefault(q => q.Id ==
answer.QuestionId);

        if (question == null)
            continue;

        bool isCorrect = answer.SelectedAnswerId ==
question.CorrectAnswerId;
        var quality = CalculateQuality(question.Difficulty,
question.PracticalValue);
        maxScores += quality;

        if (isCorrect)
        {
            totalScore += quality;
        }

        answers.Add(new Answer
        {

```

```

        IsCorrect = isCorrect,
        SelectedAnswerId = answer.SelectedAnswerId,
        QuestionId = question.Id,
        Score = isCorrect ? Math.Round(quality, 2) : 0,
    });
}

var result = new TestResult
{
    TestId = test.Id,
    StudentName = submission.StudentName,
    Answers = answers,
    Score = Math.Round(totalScore / maxScores * 100, 2),
};

dbContext.TestResults.Add(result);
dbContext.SaveChanges();

return RedirectToAction("Result", new { id = result.Id });
}

public IActionResult Result(int id)
{
    var result = dbContext.TestResults
        .Include(x => x.Test)
        .Include(x => x.Answers).ThenInclude(x => x.Question)
        .Include(x => x.Answers).ThenInclude(x => x.SelectedAnswer)
        .Where(r => r.Id == id)
        .FirstOrDefault();

    if (result == null)
        return NotFound("Тест не найдено.");

    return View(result);
}

private static double CalculateQuality(int difficulty, int
practicalValue)
{
    try
    {
        string scriptPath =
Path.Combine(Directory.GetCurrentDirectory(), "Scripts", "fuzzy_logic.py");

        var process = new Process
        {
            StartInfo = new ProcessStartInfo
            {
                FileName = "python",
                Arguments = $"\"{scriptPath}\" {difficulty}
{practicalValue}",
                RedirectStandardOutput = true,
                RedirectStandardError = true,
                UseShellExecute = false,
                CreateNoWindow = true
            }
        };

        process.Start();

        string output = process.StandardOutput.ReadToEnd();
        string error = process.StandardError.ReadToEnd();
        process.WaitForExit();
    }
}

```

```

        if (!string.IsNullOrEmpty(error))
        {
            throw new Exception($"Помилка виконання Python-скрипта:
{error}");
        }

        return double.TryParse(output.Replace('.', ',').Trim(), out
double result) ? result : 1.0;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Помилка у методі CalculateQuality:
{ex.Message}");
        return 1.0;
    }
}
}
}

```

Лістинг Create.cshtml (представлення створення тесту):

```

@model Test
@{
    ViewData["Title"] = "Створення тесту";
}

<div class="container mt-5">
    <h2 class="mb-4">Створення тесту</h2>

    @using (Html.BeginForm("Create", "Home", FormMethod.Post, new { @class =
"needs-validation", novalidate = "novalidate" }))
    {
        <div class="mb-3">
            <label for="testTitle" class="form-label">Назва тесту:</label>
            @Html.TextBoxFor(m => m.Title, new { @class = "form-control", id =
"testTitle", required = "required" })
            @Html.ValidationMessageFor(m => m.Title, "", new { @class = "text-
danger" })
        </div>

        <div class="mb-3">
            <label for="testDescr" class="form-label">Опис:</label>
            @Html.TextAreaFor(m => m.Description, new { @class = "form-control",
id = "testDescr" })
            @Html.ValidationMessageFor(m => m.Description, "", new { @class =
"text-danger" })
        </div>

        <div class="mb-3">
            <label for="duration" class="form-label">Тривалість тесту
(хвилини):</label>
            @Html.TextBoxFor(m => m.Duration, new { @class = "form-control",
type = "number", min = "1", required = "required" })
            @Html.ValidationMessageFor(m => m.Duration, "", new { @class =
"text-danger" })
        </div>

        <div class="mb-3">
            <label for="passcode" class="form-label">Кодове слово
(необов'язкове):</label>
            @Html.TextBoxFor(m => m.Passcode, new { @class = "form-control",
type = "password" })
            @Html.ValidationMessageFor(m => m.Passcode, "", new { @class =
"text-danger" })
        </div>

        <div class="mb-3">
            <label for="passcodeAdmin" class="form-label">Кодове слово
адміністратора (необов'язкове):</label>
            @Html.TextBoxFor(m => m.PasscodeAdmin, new { @class = "form-
control", type = "password" })
            @Html.ValidationMessageFor(m => m.PasscodeAdmin, "", new { @class =
"text-danger" })
        </div>

        <h3 class="mt-4">Питання</h3>
        <div id="questions-container" class="mt-3">
            @for (var i = 0; i < (Model?.Questions?.Count ?? 0); i++)
            {
                <div class="question mb-4 p-3 border rounded bg-light">

```

```

        <label class="form-label">Текст питання:</label>
        @Html.TextBox($"Questions[{i}].Text",
Model.Questions[i].Text, new { @class = "form-control", required = "required" })
        @Html.ValidationMessage($"Questions[{i}].Text", "", new {
@class = "text-danger" })

        <label class="form-label mt-3">Складність:</label>
        @Html.TextBox($"Questions[{i}].Difficulty",
Model.Questions[i].Difficulty, new { type = "number", min = "1", max = "10",
@class = "form-control", required = "required" })
        @Html.ValidationMessage($"Questions[{i}].Difficulty", "",
new { @class = "text-danger" })

        <label class="form-label mt-3">Практичне значення:</label>
        @Html.TextBox($"Questions[{i}].PracticalValue",
Model.Questions[i].PracticalValue, new { type = "number", min = "1", max = "10",
@class = "form-control", required = "required" })
        @Html.ValidationMessage($"Questions[{i}].PracticalValue",
"", new { @class = "text-danger" })

        <h4 class="mt-4">Варіанти відповідей</h4>
        <div class="answer-options mt-2">
            @for (var j = 0; j <
Model.Questions[i].AnswerOptions.Count; j++)
            {
                <div class="answer-option mb-2">
                    <label class="form-label">Текст
відповіді:</label>

                    @Html.TextBox($"Questions[{i}].AnswerOptions[{j}].Text",
Model.Questions[i].AnswerOptions[j].Text, new { @class = "form-control",
required = "required" })

                    @Html.ValidationMessage($"Questions[{i}].AnswerOptions[{j}].Text", "", new {
@class = "text-danger" })

                    <label class="form-check-label mt-2">

                        @Html.CheckBox($"Questions[{i}].AnswerOptions[{j}].IsCorrect",
Model.Questions[i].AnswerOptions[j].IsCorrect, new { @class = "form-check-input"
}) Правильна відповідь

                    </label>

                    @Html.ValidationMessage($"Questions[{i}].AnswerOptions[{j}].IsCorrect", "", new
{ @class = "text-danger" })

                </div>
            }
        </div>
        <button type="button" class="btn btn-outline-secondary mt-3"
onclick="addAnswerOption(this)">Додати відповідь</button>
        </div>
    }
</div>

    <button type="button" class="btn btn-outline-primary mt-4"
onclick="addQuestion()">Додати питання</button>
    <button type="submit" class="btn btn-success mt-4" id="submit-btn">Зберегти
тест</button>
    }
</div>

<template id="question-template">
    <div class="question mb-4 p-3 border rounded bg-light">
        <label class="form-label">Текст питання:</label>

```

```

        <input type="text" class="form-control" required data-
name="questions[{{index}}].Text" />

        <label class="form-label mt-3">Складність:</label>
        <input type="number" class="form-control" min="1" max="10" required
data-name="questions[{{index}}].Difficulty" />

        <label class="form-label mt-3">Практичне значення:</label>
        <input type="number" class="form-control" min="1" max="10" required
data-name="questions[{{index}}].PracticalValue" />

        <h4 class="mt-4">Варіанти відповідей</h4>
        <div class="answer-options mt-2"></div>

        <button type="button" class="btn btn-outline-secondary mt-3"
onclick="addAnswerOption(this)">Додати відповідь</button>
    </div>
</template>

<template id="answer-option-template">
    <div class="answer-option mb-2">
        <label class="form-label">Текст відповіді:</label>
        <input type="text" class="form-control" required data-
name="questions[{{questionIndex}}].AnswerOptions[{{answerIndex}}].Text" />
        <label class="form-check-label mt-2">
            <input type="checkbox" class="form-check-input is-correct-checkbox"
value="true" data-
name="questions[{{questionIndex}}].AnswerOptions[{{answerIndex}}].IsCorrect" />
Правильна відповідь
        </label>
    </div>
</template>

<script>
    let questionIndex = @(Model?.Questions?.Count ?? 0);

    function addQuestion() {
        const template = document.getElementById('question-template');
        const container = document.getElementById('questions-container');

        const questionClone = template.content.cloneNode(true);
        const questionElement = questionClone.querySelector('.question');

        questionElement.querySelectorAll('[data-name]').forEach(el => {
            const name = el.getAttribute('data-name').replace('{{index}}',
questionIndex);
            el.setAttribute('name', name);
        });

        container.appendChild(questionClone);

        questionIndex++;
    }

    function addAnswerOption(button) {
        const template = document.getElementById('answer-option-template');
        const answerContainer = button.previousElementSibling;
        const answerIndex = answerContainer.querySelectorAll('.answer-
option').length;

        const answerClone = template.content.cloneNode(true);
        const answerElement = answerClone.querySelector('.answer-option');

```

```

    const questionIndex = Array.from(
      document.getElementById('questions-container').children
    ).indexOf(button.closest('.question'));
    answerElement.querySelectorAll('[data-name]').forEach(el => {
      const name = el.getAttribute('data-name')
      .replace('{questionIndex}', questionIndex)
      .replace('{answerIndex}', answerIndex);
      el.setAttribute('name', name);
    });

    answerContainer.appendChild(answerClone);
  }

  document.getElementById('submit-btn').addEventListener('click', function
(event) {
    const questions = document.querySelectorAll('.question');
    for (const question of questions) {
      const correctAnswers = question.querySelectorAll('.form-check-
input:checked');
      if (correctAnswers.length !== 1) {
        event.preventDefault();
        alert('Кожне питання має мати тільки одну правильну
Відповідь.');
```

Лістинг Index.cshtml (представлення проходження тесту):

```

@model Test
@{
    ViewData["Title"] = Model.Title;
}

<div class="container mt-5">
    <h2 class="mb-4">Тест: @Model.Title</h2>

    @if (Model.Duration != 0)
    {
        <div class="d-flex justify-content-between align-items-center mb-4">
            <h3>Виділено часу: @Model.Duration хв</h3>
        </div>
    }

    <div class="d-flex justify-content-between align-items-center mb-4">
        <h3>Всього питань: @Model.Questions.Count</h3>
    </div>

    <div id="start-section" class="text-center my-5">
        <button id="start-test-btn" class="btn btn-primary btn-lg">Почати
тестування</button>
    </div>

    <div id="test-section" style="display: none;">
        @if (Model.Duration != 0)
        {
            <div class="d-flex justify-content-between align-items-center mb-4">
                <h3>Час, що залишився: <span id="timer" class="badge bg-
primary"></span></h3>
            </div>
        }

        @using (Html.BeginForm("SubmitTest", "TestTaking", FormMethod.Post))
        {
            @Html.Hidden("testId", Model.Id)

            <div class="mb-4">
                <label for="studentName" class="form-label">Баче ім'я:</label>
                @Html.TextBox("studentName", null, new { @class = "form-
control", id = "studentName", required = "required" })
            </div>

            <div class="progress mb-4">
                <div id="progress-bar" class="progress-bar" role="progressbar"
style="width: 0%;" aria-valuenow="0" aria-valuemin="0" aria-
valuemax="100">0%</div>
            </div>

            @for (var i = 0; i < Model.Questions.Count; i++)
            {
                var question = Model.Questions[i];
                <div class="question mb-4 p-3 border rounded bg-light" data-
index="@i" style="display: none;">
                    <h4>@question.Text</h4>

                    @foreach (var option in question.AnswerOptions)

```

```

        {
            @Html.Hidden($"Answers[{i}].QuestionId", question.Id)
            <div class="form-check">
                <label class="form-check-label">

@Html.RadioButton($"answers[{i}].SelectedAnswerId", option.Id, new { @class =
"form-check-input" })

                @option.Text
            </label>
            </div>
        }
    </div>
}

<div class="d-flex justify-content-between mt-4">
    <button id="prev-btn" type="button" class="btn btn-secondary"
style="display: none;">Попереднє питання</button>
    <button id="next-btn" type="button" class="btn btn-
primary">Наступне питання</button>
    <button id="submit-btn" type="submit" class="btn btn-success"
style="display: none;">Завершити тест</button>
</div>

<div class="mb-4">
    <h4>Навигация між питаннями</h4>
    <table class="table table-bordered">
        <thead>
            <tr>
                <th>№</th>
                <th>Питання</th>
                <th>Статус</th>
            </tr>
        </thead>
        <tbody>
            @for (var i = 0; i < Model.Questions.Count; i++)
            {
                <tr class="question-row" data-index="@i">
                    <td>@(i + 1)</td>
                    <td>@Model.Questions[i].Text</td>
                    <td><span class="badge bg-secondary">Не
пройдено</span></td>
                </tr>
            }
        </tbody>
    </table>
</div>
}
</div>
</div>

<script>
    document.addEventListener('DOMContentLoaded', function () {
        const questions = document.querySelectorAll('.question');
        const progressBar = document.getElementById('progress-bar');
        const rows = document.querySelectorAll('.question-row');
        const nextButton = document.getElementById('next-btn');
        const prevButton = document.getElementById('prev-btn');
        const submitButton = document.getElementById('submit-btn');
        let currentQuestion = 0;

        function isQuestionAnswered(index) {
            const question = questions[index];
            const selectedOption =
question.querySelector('input[type="radio"]:checked');

```

```

        return selectedOption !== null;
    }

    function updateQuestionStatus(index) {
        const row = rows[index];
        const badge = row.querySelector('.badge');
        if (isQuestionAnswered(index)) {
            badge.textContent = "Пройдено";
            badge.classList.replace('bg-secondary', 'bg-success');
        } else {
            badge.textContent = "Не пройдено";
            badge.classList.replace('bg-success', 'bg-secondary');
        }
    }

    function showQuestion(index) {
        questions.forEach((q, i) => q.style.display = i === index ? 'block'
: 'none');
        rows.forEach((row, i) => row.classList.toggle('table-active', i ===
index));

        prevButton.style.display = index > 0 ? 'inline-block' : 'none';
        nextButton.style.display = index < questions.length - 1 ? 'inline-
block' : 'none';
        submitButton.style.display = index === questions.length - 1 ?
'inline-block' : 'none';
        nextButton.textContent = index < questions.length - 1 ? "Наступне
питання" : "Завершити тест";

        updateProgressBar();
    }

    function updateProgressBar() {
        const answeredQuestions = Array.from(questions).filter((q, i) =>
isQuestionAnswered(i)).length;
        const progress = (answeredQuestions / questions.length) * 100;
        progressBar.style.width = `${progress}%`;
        progressBar.setAttribute('aria-valuenow', progress);
        progressBar.innerText = `${Math.round(progress)}%`;
    }

    rows.forEach((row, index) => {
        row.addEventListener('click', () => {
            currentQuestion = index;
            showQuestion(index);
        });
    });

    nextButton.addEventListener('click', () => {
        updateQuestionStatus(currentQuestion);
        if (currentQuestion < questions.length - 1) {
            currentQuestion++;
            showQuestion(currentQuestion);
        }
    });

    prevButton.addEventListener('click', () => {
        if (currentQuestion > 0) {
            currentQuestion--;
            showQuestion(currentQuestion);
        }
    });

    showQuestion(currentQuestion);

```

```

        updateProgressBar();

        questions.forEach((q, index) => {
            q.addEventListener('change', () => {
                updateQuestionStatus(index);
                updateProgressBar();
            });
        });
    });
</script>

<script>
    document.addEventListener('DOMContentLoaded', function () {
        const startButton = document.getElementById('start-test-btn');
        const testSection = document.getElementById('test-section');
        const startSection = document.getElementById('start-section');
        const timerElement = document.getElementById('timer');
        const questions = document.querySelectorAll('.question');
        const durationInMinutes = @Model.Duration;
        let timeLeft = durationInMinutes * 60;

        function updateTimer() {
            const minutes = Math.floor(timeLeft / 60);
            const seconds = timeLeft % 60;
            timerElement.innerText =
` ${minutes}:${seconds.toString().padStart(2, '0')} `;

            if (timeLeft <= 0) {
                clearInterval(timerInterval);
                document.querySelector('form').submit();
            }

            timeLeft--;
        }

        let timerInterval;

        startButton.addEventListener('click', function () {
            startSection.style.display = 'none';
            testSection.style.display = 'block';
            questions[0].style.display = 'block';

            if (durationInMinutes > 0) {
                timerInterval = setInterval(updateTimer, 1000);
                updateTimer();
            }
        });
    });
</script>

```

Лістинг Details.cshtml (представлення деталей тесту):

```

@model TestDetailsViewModel
@{
    ViewData["Title"] = "Деталі тесту";
}

<div class="container mt-5">
    <div class="card shadow-sm">
        <div class="card-header bg-primary text-white">
            <h2 class="mb-0">@Model.Test.Title</h2>
        </div>
        <div class="card-body">
            @if (!string.IsNullOrEmpty(Model.Test.Description))
            {
                <p class="fs-5"><strong>Опис:</strong>
@Model.Test.Description</p>
            }
            <p class="fs-5"><strong>Всього питань:</strong>
@Model.Test.Questions.Count</p>
            @if (Model.Test.Duration > 0)
            {
                <p class="fs-5"><strong>Час на тестування:</strong>
@Model.Test.Duration хв.</p>
            }

            <h3 class="mt-4">Список питань</h3>
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th>#</th>
                        <th>Питання</th>
                        <th>Складність</th>
                        <th>Практична цінність</th>
                        <th>Відсоток правильних відповідей</th>
                    </tr>
                </thead>
                <tbody>
                    @for (var i = 0; i < Model.QuestionStats.Count; i++)
                    {
                        var stat = Model.QuestionStats[i];
                        <tr>
                            <td>@(i + 1)</td>
                            <td>@stat.Question.Text</td>
                            <td>@stat.Question.Difficulty</td>
                            <td>@stat.Question.PracticalValue</td>
                            <td>@stat.CorrectPercentage%</td>
                        </tr>
                    }
                </tbody>
            </table>

            <div class="card shadow-sm mb-4">
                <div class="card-header bg-primary text-white">
                    <h4 class="mb-0">Графік правильних відповідей за
складністю</h4>
                </div>
                <div class="card-body">
                    <canvas id="difficultyChart" width="400"
height="200"></canvas>
                </div>
            </div>
        </div>
    </div>

```

```

</div>

<div class="card shadow-sm mb-4">
  <div class="card-header bg-primary text-white">
    <h4 class="mb-0">Графік правильних відповідей за практичною
цінністю</h4>
  </div>
  <div class="card-body">
    <canvas id="practicalValueChart" width="400"
height="200"></canvas>
  </div>
</div>

<div class="card shadow-sm mb-4">
  <div class="card-header bg-primary text-white">
    <h4 class="mb-0">Кругова діаграма складності питань</h4>
  </div>
  <div class="card-body">
    <canvas id="difficultyPieChart" width="400"
height="200"></canvas>
  </div>
</div>

<div class="card shadow-sm mb-4">
  <div class="card-header bg-primary text-white">
    <h4 class="mb-0">Кругова діаграма практичної цінності
питань</h4>
  </div>
  <div class="card-body">
    <canvas id="practicalValuePieChart" width="400"
height="200"></canvas>
  </div>
</div>

<h3 class="mt-4">Останні проходження</h3>
@if (!Model.RecentResults.Any())
{
  <p class="text-muted">Цей тест ще ніхто не проходив.</p>
}
else
{
  <table class="table table-striped">
    <thead>
      <tr>
        <th>Ім'я</th>
        <th>Оцінка</th>
        <th>Правильних відповідей</th>
        <th></th>
      </tr>
    </thead>
    <tbody>
      @foreach (var result in Model.RecentResults)
      {
        var answersCount = result.Answers.Count;
        var correctAnswersCount = result.Answers.Count(x =>
x.IsCorrect);
        var correctAnswersPercentage =
Math.Round(correctAnswersCount * 100.0 / answersCount, 2);
        <tr>
          <td>@result.StudentName</td>
          <td>@result.Score</td>
          <td>@correctAnswersCount / @answersCount
(@correctAnswersPercentage %)</td>

```

```

                                <td><a href="@Url.Action("Result", "TestTaking",
new { id = result.Id })" class="btn btn-secondary">Детали</a></td>
                                </tr>
                                }
                                </tbody>
                                </table>
                                }
                                </div>
                                </div>
                                <a href="@Url.Action("Index", "Home")" class="btn btn-secondary mt-3">Назад
до списку</a>
                                </div>

<script>
    document.addEventListener('DOMContentLoaded', function () {
        const ctx = document.getElementById('difficultyChart').getContext('2d');
        const difficultyLabels =
@Html.Raw(Json.Serialize(Model.DifficultyStats.Select(s => s.Key)));
        const correctPercentages =
@Html.Raw(Json.Serialize(Model.DifficultyStats.Select(s => s.Value)));

        new Chart(ctx, {
            type: 'bar',
            data: {
                labels: difficultyLabels,
                datasets: [{
                    label: 'Відсоток правильних відповідей',
                    data: correctPercentages,
                    backgroundColor: 'rgba(54, 162, 235, 0.6)',
                    borderColor: 'rgba(54, 162, 235, 1)',
                    borderWidth: 1
                }]
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true,
                        max: 100
                    }
                }
            }
        });
    });
</script>

<script>
    document.addEventListener('DOMContentLoaded', function () {
        const ctx =
document.getElementById('practicalValueChart').getContext('2d');
        const difficultyLabels =
@Html.Raw(Json.Serialize(Model.PracticalValueStats.Select(s => s.Key)));
        const correctPercentages =
@Html.Raw(Json.Serialize(Model.PracticalValueStats.Select(s => s.Value)));

        new Chart(ctx, {
            type: 'bar',
            data: {
                labels: difficultyLabels,
                datasets: [{
                    label: 'Відсоток правильних відповідей',
                    data: correctPercentages,
                    backgroundColor: 'rgba(54, 162, 235, 0.6)',
                    borderColor: 'rgba(54, 162, 235, 1)',
                    borderWidth: 1
                }]
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true,
                        max: 100
                    }
                }
            }
        });
    });
</script>

```

```

        ]]
    },
    options: {
        scales: {
            y: {
                beginAtZero: true,
                max: 100
            }
        }
    }
});
});
</script>

<script>
    document.addEventListener('DOMContentLoaded', function () {
        const ctx =
document.getElementById('difficultyPieChart').getContext('2d');
        const labels =
@Html.Raw(Json.Serialize(Model.PracticalValueDistribution.Select(d =>
d.Key.ToString())));
        const data =
@Html.Raw(Json.Serialize(Model.DifficultyDistribution.Select(d => d.Value)));

        new Chart(ctx, {
            type: 'pie',
            data: {
                labels: labels,
                datasets: [{
                    data: data,
                    backgroundColor: [
                        'rgba(75, 192, 192, 0.6)',
                        'rgba(255, 159, 64, 0.6)',
                        'rgba(153, 102, 255, 0.6)',
                        'rgba(255, 99, 132, 0.6)',
                        'rgba(54, 162, 235, 0.6)'
                    ],
                    borderColor: [
                        'rgba(75, 192, 192, 1)',
                        'rgba(255, 159, 64, 1)',
                        'rgba(153, 102, 255, 1)',
                        'rgba(255, 99, 132, 1)',
                        'rgba(54, 162, 235, 1)'
                    ],
                    borderWidth: 1
                }
            ]
        },
        options: {
            plugins: {
                legend: {
                    position: 'right'
                }
            }
        }
    });
});
</script>

<script>
    document.addEventListener('DOMContentLoaded', function () {
        const ctx =
document.getElementById('practicalValuePieChart').getContext('2d');

```

```

        const labels =
@Html.Raw(Json.Serialize(Model.PracticalValueDistribution.Select(d =>
d.Key.ToString())));
        const data =
@Html.Raw(Json.Serialize(Model.PracticalValueDistribution.Select(d =>
d.Value)));

        new Chart(ctx, {
            type: 'pie',
            data: {
                labels: labels,
                datasets: [{
                    data: data,
                    backgroundColor: [
                        'rgba(75, 192, 192, 0.6)',
                        'rgba(255, 159, 64, 0.6)',
                        'rgba(153, 102, 255, 0.6)',
                        'rgba(255, 99, 132, 0.6)',
                        'rgba(54, 162, 235, 0.6)'
                    ],
                    borderColor: [
                        'rgba(75, 192, 192, 1)',
                        'rgba(255, 159, 64, 1)',
                        'rgba(153, 102, 255, 1)',
                        'rgba(255, 99, 132, 1)',
                        'rgba(54, 162, 235, 1)'
                    ],
                    borderWidth: 1
                }]
            },
            options: {
                plugins: {
                    legend: {
                        position: 'right'
                    }
                }
            }
        });
    });
</script>

```

Лістинг Result.cshtml (представлення результату тестування):

```

@model TestResult
@{
    ViewData["Title"] = "Результат";
    var answersCount = Model.Answers.Count;
    var correctAnswersCount = Model.Answers.Count(x => x.IsCorrect);
    var correctAnswersPercentage = Math.Round(correctAnswersCount * 100.0 /
answersCount, 2);
}

<div class="container mt-5">
    <div class="card shadow mb-4">
        <div class="card-header bg-primary text-white">
            <h2 class="mb-0 text-center">Результат тестування</h2>
        </div>
        <div class="card-body">
            <p class="fs-5"><strong>Ім'я:</strong> <span class="text-
secondary">@Model.StudentName</span></p>
            <p class="fs-5"><strong>Тест:</strong> <span class="text-
secondary">@Model.Test.Title</span></p>
            <p class="fs-5"><strong>Правильних відповідей:</strong> <span
class="text-secondary">@correctAnswersCount / @answersCount
(@correctAnswersPercentage%)</span></p>
            <p class="fs-5"><strong>Оцінка:</strong> <span id="score"
class="badge fs-6">@Model.Score</span></p>
        </div>
    </div>

    <div class="card shadow">
        <div class="card-header bg-secondary text-white">
            <h4 class="mb-0">Деталі питань</h4>
        </div>
        <div class="card-body">
            @if (!Model.Answers.Any())
            {
                <p class="text-muted">Відповіді на питання не знайдено.</p>
            }
            else
            {
                <table class="table table-bordered">
                    <thead>
                        <tr>
                            <th>#</th>
                            <th>Питання</th>
                            <th>Відповідь</th>
                            <th>Бал</th>
                        </tr>
                    </thead>
                    <tbody>
                        @for (var i = 0; i < Model.Answers.Count; i++)
                        {
                            var answer = Model.Answers[i];
                            var question = answer.Question;
                            var isCorrectClass = answer.IsCorrect ? "table-
success" : "table-danger";
                            <tr class="@isCorrectClass">
                                <td>@(i + 1)</td>
                                <td>@question.Text</td>
                                <td>@answer.SelectedAnswer.Text</td>
                                <td>@answer.Score</td>
                            </tr>
                        }
                    </tbody>
                </table>
            }
        </div>
    </div>
</div>

```

```

        </tr>
      }
    </tbody>
  </table>
}
</div>
</div>
</div>
<script>
  document.addEventListener('DOMContentLoaded', function () {
    const scoreElement = document.getElementById('score');
    const score = parseFloat(scoreElement.innerText);

    if (score >= 80) {
      scoreElement.classList.add('bg-success');
    } else if (score >= 60) {
      scoreElement.classList.add('bg-warning');
    } else {
      scoreElement.classList.add('bg-danger');
    }
  });
</script>

```