МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук, фізики та математики
Кафедра інформатики, програмної інженерії та економічної кібернетики

ПРОГРАМНИЙ МОДУЛЬ ПРЕДИКАТНОГО ПЕРЕТВОРЮВАЧА ЧИСЛОВИХ ІНТЕРВАЛІВ

Кваліфікаційна робота (проєкт) на здобуття ступеня вищої освіти "магістр"

Виконав: студент 2 курсу
Спеціальності 126 Інформаційні системи та технології
Освітньо-професійної програми Інформаційні системи та технології другого (магістерського) рівня вищої освіти Кліманов Георгій Миколайович
Керівники кандидат фізико-математичних наук, доктор педагогічних наук, професор Співаковський Олександр Володимирович доктор фізико-математичних наук, професор Песчаненко Володимир Сергійович
Рецензент кандидат фізико-математичних наук, доцент Бистрянцева Анастасія Миколаївна

Херсон – 2019

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

RSS (Rich Site Summary) – XML-формати, що використовуються для публікації змінної інформації (блоги, новини);

SQL ін’єкції – способ злому веб-проектів, що працюють з базами даних, здатний виконати запит до бази даних з метою отримання або обробки даних бази;

URL (Uniform Resource Locator) - адреса певного ресурсу;

ORM (Object-relational mapping) — технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування;

JS (JavaScript) – мова програмування;

API (Application Programming Interface) — це набір готових класів, процедур, функцій, структур і констант, що надаються додатком (бібліотекою, сервісом) для використання в зовнішніх програмних продуктах;

ВСТУП

Актуальність теми. Отримання базових знань є досить актуальною темою у суспільстві. Сервіс, функціонально спрямований на вдосконалення та перевірку базових математичних знань, матиме успіх серед користувачів, які зацікавлені в отриманні знань з математики.

Об'єкт дослідження : Числові інтервали та логіка предикатів.

Предмет дослідження : Програмний засіб, розроблений для удосконалення та засвоєння знань з математики.

Мета дослідження : Реалізувати модуль предикатного перетворювача числових інтервалів на основі редактору, написаному, застосовуючи web-програмування.

Завдання дослідження :

1. Ознайомитися з числовими інтервалами та логікою предикатів;
2. Ознайомитися та обрати інструменти для реалізації програмного продукту;
3. Реалізувати бек-енд додатку перетворювача числових інтервалів;
4. Реалізувати фронт-енд додатку перетворювача числових інтервалів;

Структура роботи : Робота складається з 2 розділів. У першому розділі описується числові інтервали та логіка предикатів. У другому розділі описується проектування, вибір технологій для реалізації проекту, розробка, тестування та впровадження програмних об’єктів.

Методи дослідження : Аналіз існуючих сервісів математичного напряму, емпіричні методи, проведення системного підходу, порівняння існуючих сервісів, виявлення недоліків та переваг.

Зв’язок роботи з науковими програмами, планами, темами : У рамках створення проекту математичного редактору проектується та розробляється програмне забезпечення для його реалізації.

Наукова новизна одержаних результатів створенні удосконаленої версії вже існуючих математичних сервісів

Інформаційна база : підручники з мультимедійних технологій, вкладка «довідник» у нашому сервісі.

Практичне значення полягає у використанні у подальшому для навчального процесу, а також запровадження нового функціоналу.

РОЗДІЛ 1
ПРЕДИКАТНИЙ ПЕРЕТВОРЮВАЧ ЧИСЛОВИХ ІНТЕРВАЛІВ

1.1. Введення в числові інтервали.

Саме по собі рівняння на числовій прямій має лише одну точку, котра повністю залежить від перетворень, котрі були виконано та від обраного кореня. Рішення рівняння і є числовою безліччю та на числовій прямій це все відображатиметься однією точкою, але такрж є в математиці значно узагальнені типи відносин між двома числами - нерівності. Числова пряма цих нерівностей повністю розділяється деяким визначеним числом, та від цієї прямої віднімається деяка частина - визначення виразу або числовий проміжок.

Числові проміжки дуже щільно перетинаються разом з нерівностями, але це перетинання ніяк не доказує, що ці проміжки пов'язані лише з ними. Числові проміжки, інтервали, відрізки або промені – множина значень змінної, котрі повністю задовольняють якусь визначену нерівності. Тобто, проміжок – це безліч всіх точок на числовій прямій, обмеженою якимись рамками. Це і є причиною таких тісних зв’язків теми числових проміжків з поняттям змінної. У будь-якому місці, де може бути змінна, або довільна точка x на числовій прямій, і її використовують, є також і числові проміжки, інтервали, значення котрих дорівнює x. Часами значення проміжків може бути будь-яким, але це теж числовий проміжок, що охоплює всю числову пряму.

Цінність числових проміжків, котрі можна виділити серед множин, об'єктами яких є числа, в тому, що дуже легко уявити безліч, відповідне вказаною числовому проміжку, і навпаки. Тому з допомогою числових проміжків дуже зручно записувати рішення нерівностей. Тоді як безліччю рішення рівняння буде не числовий проміжок, а просто кілька чисел на числовій прямій, з нерівностями, інакше кажучи, будь-якими обмеженнями значення змінної з'являються числові проміжки.

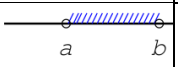
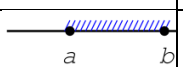
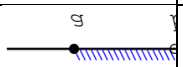
Числовий проміжок – це безліч всіх точок числової прямої, котра обмежена точками на числовій прямій.

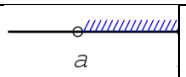
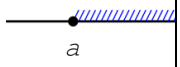
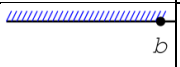
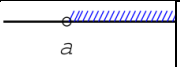
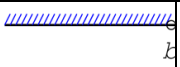
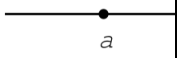
Числовий проміжок виду «безліч значень x, укладених між деякими числами» можна представити трьома видами математичних позначень: спеціальними позначеннями проміжків, однією або подвійною нерівностями (так звані ланцюжки нерівностей) або геометрично на числовій прямій. Всі ці позначення мають один сенс. Вони потрібні для обмеження значень якогось математичного об'єкта, змінної величини будь-якого виразу зі змінною.

Після усього вище написаного можна прийти до висновку, що так як є різні типи нерівностей, то і типи числових проміжків бувають різні.

Види числових нерівностей

Кожен тип числового проміжку має власну назву, так зване особливе позначення. Для позначення числових проміжків використовують круглі та квадратні дужки. Кругла дужка означає, що кінцева, яка визначає межу, точка на числовій прямій у цій дужки не входить в безліч точок даного проміжку. А інша дужка означає, що кінець входить в проміжок. З нескінченністю (з одного з боків проміжок не обмежений) використовують круглу дужку. Також існують ситуації, коли замість круглих дужок можна писати квадратні, повернені у зворотний бік: (a; b) ⇔] a; b [

Вид проміжку	**Геометричне позначення**	**Позначення**	**Запис у вигляді нерівностей**
Інтервал		(a;b)	a < x < b
Сегмент		[a;b]	a ≤ x ≤ b
Напівінтервал		[a;b)	a ≤ x < b

Напівінтервал		(a;b]	a < x ≤ b
Промінь		[a;+∞)	x ≥ a
Промінь		(-∞;b]	x ≤ b
Відкритий промінь		(a;+∞)	x > a
Відкритий промінь		(-∞;b)	x < b
Множина всіх чисел		(-∞;+∞)	x ∈ ℝ
Рівність		[a;a] або x=a	x = a
Пуста множина		∅	x∈∅⇔x∈{ }

З назвами проміжків може виникнути плутанина: є величезна кількість варіантів. Тому краще завжди точно їх вказувати. В англомовній літературі використовується тільки термін інтервал ("interval") - відкритий, замкнений, напіввідкритий (напівзамкнений).

За допомогою проміжків в математиці позначається дуже велика кількість речей: є проміжки ізоляції при вирішенні рівнянь, проміжки інтегрування, проміжки збіжності рядів. Проміжками прийнято завжди позначати при дослідженні функції її область значень і область визначення.

Гранична точка - це точка, яка визначає межу числового проміжку. Гранична точка може як належати числовому проміжку, так і не належати йому. На

кресленнях граничні точки, які не належать оскільки він розглядався числовому проміжку, позначають незакрашенним кругом, а належать - зафарбовані кругом.

Відкритий промінь - це безліч точок прямої, що лежать по одну сторону від граничної точки, яка не входить в дану безліч. Відкритим промінь називається саме через граничної точки, яка йому не належить.

Замкнуте промінь - це безліч точок прямої, що лежать по одну сторону від граничної точки, що належить даній безлічі. На кресленнях граничні точки, що належать розглянутого безлічі, позначаються зафарбовані кругом.

Відрізок - це безліч точок прямої, що лежать між двома граничними точками, які належать даній безлічі. Такі безлічі задаються подвійними нестрогими нерівностями.

Інтервал - це безліч точок прямої, що лежать між двома граничними точками, що не належать даній безлічі. Такі безлічі задаються подвійними строгими нерівностями.

Напівінтервал - це безліч точок прямої, що лежать між двома граничними точками, одна з яких належить множині, а інша не належить.

Системи та сукупності нерівностей

Отже, змінну x або значення деякого виразу можна порівняти з якоюсь постійною величиною - це нерівність, але можна порівнювати цей вислів з декількома величинами - подвійна нерівність, ланцюжок нерівностей і т.д. Саме це було показано – як інтервал і відрізок, які обидва є системою нерівностей.

Якщо ставити завдання знайти безліч спільних рішень двох або більше нерівностей, то можна говорити про рішення системи нерівностей (також як з рівняннями - хоча можна сказати, що рівняння - це окремий випадок).

Тоді очевидно, що значення змінної, використаної в нерівностях, при якому кожне з них звертається в вірне, називається рішення системи нерівностей.

Всі нерівності, що входять в систему об'єднують фігурною дужкою - "{". Іноді їх записують у вигляді подвійної нерівності (як показано вище) або навіть ланцюжком нерівностей. Приклад типового запису:

$$\begin{cases} f(x) \leq 30 \\ g(x) \geq 5 \end{cases}$$

Далі системи нерівностей можна класифікувати як рівносильні, якщо вони мають спільне безліч рішень. Звідси слід, що більш складні системи можна спрощувати (наприклад, використовуючи геометричне рішення).

Фігурну дужку можна умовно, грубо кажучи, назвати еквівалентом союзу "І" для нерівностей.

Однак, бувають і інші випадки. Так крім перетину множин рішень буває їх об'єднання: якщо ставиться завдання знайти безліч всіх таких значень змінної, кожне з яких є рішенням хоча б одного з цих нерівностей, то кажуть, що треба вирішити сукупність нерівностей.

Отже, все нерівності в сукупності об'єднують дужкою сукупності "[". Якщо значення змінної задовольняє хоча б одному нерівності з сукупності, то воно належить множині рішень всієї сукупності. Також і з рівняннями (знову ж їх можна назвати окремим випадком).

Якщо фігурна дужка - і, то дужка сукупності - це, умовно, кажучи простою мовою, еквівалент союзу "АБО" для нерівностей (хоча це, звичайно, буде логічне або, що включає випадок, що задовольняє обидві умови).

Отже, рішення сукупності нерівностей - це значення змінної, при якому хоча б одне нерівність, звертається в вірне.

Безліч рішень, як сукупності, так і системи нерівностей, можна визначити через дві основні бінарні операції для роботи з множинами - перетин і об'єднання. Безліч рішень системи нерівностей - це перетин множин рішень нерівностей, її

складових. Безліч рішень сукупності нерівностей - це об'єднання множин рішень нерівностей, її складових.

1.2. Логіка предикатів.

Поняття «предикат» узагальнює поняття «висловлювання». Предикат - це висловлювання, в яке можна підставляти аргументи. Якщо аргумент один - то предикат виражає властивість аргументу, якщо більше - то відношення між аргументами.

Приклад предикатів. Висловлювання: «Сократ – людина», «Платон – людина». Обидва ці висловлювання висловлюють властивість «бути людиною». Таким чином, ми можемо розглядати предикат «бути людиною» і говорити, що він виконується для Сократа і Платона.

Мова логіки висловлювань не цілком підходить для вираження логічних міркувань, проведених людьми, більш зручний для цього мову логіки предикатів.

Приклад міркування, що не виразність в логіці висловлювань. Всі люди смертні. Сократ - людина. Отже, Сократ смертний.

Це міркування на мові логіки висловлювань можна записати трьома окремими висловлюваннями. Однак ніякого зв'язку між ними встановити не вдасться. Мовою логіки предикатів ці пропозиції можна виразити за допомогою двох предикатів: «бути людиною» і «бути смертним». Перше речення встановлює зв'язок між цими предикатами.

Предикатна сигнатура - це безліч символів двох типів - об'єктні константи і предикатні константи - з невід'ємним цілим числом, названим арність, призначеним кожної предикатной константі. Предикатну константу називають пропозіціональній, якщо її арність дорівнює 0. Пропозіціональні константи є аналогом атомів в логіці висловлювань. Предикатна константа унарна, якщо її

арність дорівнює 1, і бінарна, якщо її арність дорівнює 2. Наприклад, ми можемо визначити предикатну сигнатуру

{A, P, Q}

оголошуючи *a* об'єктної константою, P - унарною предикатною константою, і Q - бінарної предикатною константою.

Візьмемо предикатну сигнатуру s, яка включає принаймні одну предикатну константу і не включає жодного з наступних символів:

- об'єктні змінні x, y, z, x1, y1, z1, x2, y2, z2, ...,
- пропозиціональні зв'язки,
- квантор загальності "і квантор існування $,
- дужки і кома.

Алфавіт логіки предикатів складається з елементів з s і чотирьох груп додаткових символів, зазначених вище. Рядок - це кінцева послідовність символів з цього алфавіту.

Терм - це об'єктна константа або об'єктна змінна. Рядок називається атомарної формулою, якщо вона є пропозиціональною константою або має вигляд $R(t1, ..., t_n)$, де R - предикатна константа арності n ($n > 0$) и $t1, ... , t_n$ – терми. Наприклад, якщо ми розглядаємо сигнатуру {A, P, Q}, то P (a) і Q (a, x) - атомарні формули.

Безліч X рядків замкнуто щодо правил побудови (для логіки предикатів), якщо:

- кожна атомарна формула належить X,
- для будь-якого рядка F якщо F належить X, то ¬F теж належить,
- для будь-яких рядків F, G і будь-який бінарної зв'язки Д, якщо F і G належать X, то також належить (F Д G),

- для будь-якого квантора K, будь-якої змінної v і будь-якого рядка F якщо F належить X, то також належить *Kv F*.

Рядок F є формулою, якщо F належить всім множинам, які замкнуті щодо правил побудови.

Наприклад, якщо розглянута сигнатура є {A, P, Q}, тоді $(\neg P(a) \vee \exists x(P(x) \& Q(x, y)))$ – формула.

У разі предикатних формул доказ по структурної індукції має такий вигляд. Для даного властивості формул ми перевіряємо, що:

- кожна атомарна формула володіє цією властивістю,
- для будь-якої формули F, що володіє цією властивістю, ¬F також володіє цією властивістю,
- для будь-яких формул F, G, що володіють цією властивістю, і будь-який бінарної зв'язки Д, (F Д G) також володіє цією властивістю,
- для будь-якого квантора K, будь-якої змінної v і будь-якої формули F, що володіє цією властивістю, Kv F також володіє цією властивістю.

Тоді це властивість виконується для всіх формул.

Безліч вільних змінних * формули F визначається рекурсивно, в такий спосіб:

- Всі змінні, що входять в атомарному формулу, є вільними змінними цієї формули,
- вільні змінні формули F є вільними змінними формули ¬F,
- змінні, які є вільними для хоча б однієї з формул F або G, є вільними змінними формули (F Д G),
- всі вільні змінні формули F крім v є вільними змінними формули Kv F.

Формула без вільних змінних називається замкнутої формулою, або пропозицією. Змінна v пов'язана у формулі F, якщо F містить входження Kv, де K - квантор.

Підстановка

Нехай F - формула і v - змінна. Результат підстановки терма t замість v в F - формула, певна рекурсивно наступним чином:

- результат підстановки t замість v в атомарному формулу F виходить з F одночасною заміною всіх входжень v на t,
- якщо результат підстановки t замість v в F є F 'тоді результат підстановки t замість v в ¬F є ¬F',
- якщо результат підстановки t замість v в F і G є F 'і G' тоді результат підстановки t замість v в (F Д G) є (F'Д G '),
- якщо результат підстановки t замість v в F є F 'і w - змінна, що відрізняється від v, тоді результат підстановки t замість v в Kw F є Kw F',
- результат підстановки t замість v в Kv F є Kv F.

Щоб розрізняти «погані» підстановки від «хороших», ми визначимо, коли терм t є підстановлювальний для змінної v у формулі F.

- Якщо F - атомарна, тоді t є підстановлювальний для змінної v в F,
- t є підстановлювальний для змінної v в ¬F тоді і тільки тоді, коли t є підстановлювальний для v в F,
- t є підстановлювальний для v в (F Д G) тоді і тільки тоді, коли t є підстановлювальний для v і в F і в G,
- t є підстановлювальний для v в Kw F тоді і тільки тоді, коли:

1. t не містить w і є підстановлювальний для v в F, або
2. v не є вільною змінною формули Kw F.

Терм, який не містить жодної пов'язаної змінної формули F, є підстановлювальний в F для будь-якої змінної.

Універсальне замикання формули F - це пропозиція $\forall v1 \cdots v_n F$, *де* $v1, \ldots, v_n$ – всі вільні змінні F.

У логіці предикатів висновок визначається так само, як і в обчисленні висловлювань та секвенції мають той же синтаксис. Аксіоми теж визначаються так само, як в логіці висловлювань. Всі правила виведення числення висловлень - правила введення і видалення для пропозіціональних зв'язок, правила протиріччя і відомості до протиріччя - включені в безліч правил виведення логіки предикатів, з метапеременнимі для формул розуміються тепер як предикатні формули. На додаток, є чотири нових правил виведення: правила введення і видалення для кванторів.

Коректність і повнота логіки предикатів

Безліч правил виведення для логіки предикатів має властивість коректності та повноти подібно властивостям пропозіціональних висновків.

Теорема коректності. Якщо існує висновок замкнутої формули F з безлічі формул G, тоді G тягне F.

Теорема повноти. Для будь-якої замкнутої формули F і будь-якого безлічі пропозицій G, якщо G тягне F, то існує висновок F з деякого підмножини G.

Повнота логіки предикатів для випадку рахункового G і для іншого безлічі правил виведення була доведена Куртом Геделем в 1930 році.

Функціональні символи і рівність: синтаксис

Логіка предикатів, певна вище трохи більше обмежена, ніж що звичайно називається «логікою першого порядку», і наша наступна мета - видалити ці обмеження. По-перше, ми узагальнимо поняття терма. На додаток до об'єктним констант і об'єктним змінним, ми дозволимо побудова термів з використанням символів для функцій, «функціональних констант». По-друге, ми додамо до мови знак рівності, і рівняння будуть включені як новий тип атомарних формул.

Сигнатура - це безліч символів двох типів - функціональних констант і предикатних констант - з невід'ємним цілим числом, званим арность, пов'язаних з кожним символом. Об'єктна константа - це функціональна константа арності 0. Функціональна константа унарна, якщо її арность дорівнює 1, і бінарних, якщо її арность дорівнює 2. пропозіціональному константи, так само як унарні і бінарні предикатні константи, визначені як вище.

Візьмемо сигнатуру s, що не включає ні додаткових символів, зазначених на початку цієї частини, ні знака рівності. Безліч X рядків замкнуто щодо правил побудови термів, якщо:

- кожна об'єктна константа належить X,
- кожна об'єктна змінна належить X,
- для кожної функціональної константи h арності n (n> 0) і будь-якого рядка t1, ..., tn, якщо t1, ..., tn належить X, тоді також належить h (t1, ..., tn).

Рядок є термом, якщо вона належить все безлічам, які замкнуті щодо правил побудови.

У логіці першого порядку існують три типи атомарних формул:

- пропозіціональні константи,
- такі рядки R (t1, ..., tn) де R - предикатна константа арності n (n> 0) і t1, ..., tn - терми,
- такі рядки (t1 = t2), де t1, t2 - терми.

Взявши в якості безлічі атомарних формул дане безліч, ми отримуємо, що визначення формул (першого порядку) збігається з визначенням предикатних формул на початку цієї частини.

Для будь-яких термів t1 і t2, t1 № t2 позначає формулу ¬ (t1 = t2).

1.3. Теорії першого порядку.

Теорія першого порядку сигнатури s визначається за допомогою аксіом. Інтерпретація, при якій істинні всі аксіоми теорії першого порядку G, називається моделлю G. Якщо теорія першого порядку G здійсненна, ми також говоримо що вона несуперечлива. Логічні наслідки теорії першого порядку називається її теоремами. Доказ пропозиції F в теорії першого порядку G є висновок F з підмножини аксіом з G.

Теореми коректності та повноти виконуються для логік предикатів з функціональними символами і рівністю і можуть бути сформульовані в рамках теорій першого порядку наступним чином. У відповідність з теоремою коректності, якщо існує доказ пропозиції F в теорії першого порядку G, тоді F є теоремою G. У відповідність з теоремою повноти Геделя, зворотне також вірно: для будь-якої теореми F теорії першого порядку G, існує доказ F в G.

Однак, додавання правил виведення для кванторів другого порядку веде до формальної системи яка коректна, але не повна.

Арифметика першого порядку

Ми будемо спрощувати запис формул сигнатури арифметики першого порядку (6) введенням наступного позначення: a буде записуватися як 0, s (t) як t ', f (t1, t2) як t1 + t2, і g (t1, t2) як t1 · t2. Аксіоми арифметики першого порядку є універсальним замиканням наступних формул:

1. $x' \neq 0$.
2. $x' = y' \supset x = y$.
3. $(F(0) \,\&\, \forall\, v\, (F(v) \supset F(v'))) \supset \forall\, v\, F(v)$ для будь-якої формули $F(v)$.
4. $x + 0 = x$.
5. $x + y' = (x + y)'$.
6. $x \cdot 0 = 0$.
7. $x \cdot y' = x \cdot y + x$.

Нестандартні моделі арифметики

Терми 0, 0 ', 0' ', ... називаються цифрами. Модель M арифметики першого порядку стандартна, якщо для кожного c Про | M | існує цифра t така, що tM = c.

Існують моделі арифметики першого порядку, які не володіють цією властивістю. Щоб довести існування такої моделі, корисно розглянути таку теорію першого порядку G. Сигнатура G виходить з сигнатури арифметики першого порядку додаванням літери b в якості нової об'єктної константи. Безліч аксіом G виходить з безлічі аксіом арифметики першого порядку додаванням формул b № 0, b № 0 ', b № 0' ', ... в якості нових аксіом.

Існування нестандартних моделей арифметики випливає з теореми Сколема (1920), який узагальнив ранню роботу Леопольда Лёвенхейма (1915). Можливість таких моделей різко контрастує з результатом завдання 1.41. Різниця пов'язана з тим, що мова арифметики першого порядку є занадто обмеженим для вираження аксіоми індукції. `` Арифметика другого порядку ", в якій схема індукції замінюється аксіомі (8), не має нестандартних моделей.

Класифікація предикатів

Предикат $P(x_1, x_2, \ldots, x_n)$, заданий на множинах$M_1, M_2, \ldots, M_n$, називається:

а) тотожно істинним, якщо при будь-якій підстановці замість змінних $x_1, x_2, \ldots, x_n$ будь-яких конкретних предметів$a_1, a_2, \ldots, a_n$ з множин $M_1, M_2, \ldots, M_n$відповідно він перетворюється в істинне висловлення $P(a_1, a_2, \ldots, a_n)$;

б) тотожне помилковим, якщо при будь-підстановці замість змінних $x_1, x_2, \ldots, x_n$ будь-яких конкретних предметів з множин $M_1, M_2, \ldots, M_n$ відповідно він перетворюється в хибне висловлювання;

в) здійсненним (опровержимие), якщо існує щонайменше один набір конкретних предметів $a_1, a_2, \ldots, a_n$ з множин$M_1, M_2, \ldots, M_n$ відповідно, при підстановці яких замість відповідних предметних змінних в предикат $P(x_1, x_2, \ldots, x_n)$останній перетвориться на справжнє (помилкове) висловлювання $P(a_1, a_2, \ldots, a_n)$.

Відзначимо деякі досить очевидні закономірності взаємозв'язків між предикатами різних типів (рекомендується осмислити їх):

1) кожен тотожне істинний предикат є здійсненним, але зворотне невірно;

2) кожен тотожне помилковий предикат є опровержимие, але зворотне невірно;

3) кожен не тотожне істинний предикат буде опровержимие, але, взагалі кажучи, не буде тотожним хибним;

4) кожен не тотожне помилковий предикат буде здійсненним, але, взагалі кажучи, не буде тотожним істинним.

У термінах безлічі істинності легко висловити поняття, пов'язані з класифікацією предикатів. Справді, n-місцевий предикат $P(x_1, x_2, \ldots, x_n)$, заданий на множинах $M_1, M_2, \ldots, M_n$, буде:

а) тотожно істинним тоді і тільки тоді, коли $P^+ = M_1 \times M_2 \times \ldots \times M_n$

б) тотожне хибним тоді і тільки тоді, коли $P^+ = \varnothing$

в) здійсненним тоді і тільки тоді, коли $P^+ \neq \varnothing$

г) спростовним тоді і тільки тоді, коли $P^+ \neq M_1 \times M_2 \times \ldots \times M_n$

РОЗДІЛ 2

ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО МОДУЛЮ

2.1. Вибір мови програмування.

Однією з найбільших проблем під час підготовки до написання програмного модулю предикатного перетворювача числових інтервалів, безперечно, є вибір мови програмування, фреймворків та середовища розробки програмного забезпечення.

Безумовно, щоб закрити усі поставлені задачі програмного модулю, підходить велика кількість мов програмуванні, тому підібрати зручну у розробці, швидку у виконанні, легку у засвоєнні та таку мову, яка має велике товариство діючих розробників, а також велику кількість теоретичних відомостей та документації – є досить важливою задачею, до виконання якої треба поставитися відповідально. Адже, неправильно обрана мова програмування значно уповільнює процес розробки та може стати головною причиною погано зробленого програмного модулю предикатного перетворювача числових інтервалів.

Після довгого аналізу, перечитування офіційних документацій, обговорення цього питання на безлічі сервісів та форумів, було прийнято рішення для реалізації програмного модулю предикатного перетворювача числових інтервалів використати мову програмування Python.

Тож, чому було обрано Python? Одна з найголовніших причин – ця мова програмування є однією з найпопулярніших у світі серед інших мов програмування. Популярність мови програмування дає велику спільноту розробників, які зможуть у будь-який час допомогти у вирішенні проблем та задач програмного модуля предикатного перетворювача числових інтервалів; велику кількість документації, як офіційної, так і авторської на будь-яку тему та

вирішуючи будь-яку проблему (у тому числі і відео-уроки, що є досить важливою частиною під час вивчення мови програмування); безліч готових проектів з откритим доступом та кодом. Також, маємо велику кількість тренажерів для навчання. Це значно спрощує вивчення та розробку, дає змогу ознайомитися з тим, як використовують ті чи інші рішення у своїх проектах діючі програмісти.

Наступним плюсом для вибору Python є зручний та зрозумілий синтаксис написання коду. Він не містить фігурних дужок, виглядає охайно та дуже просто читається програмістами будь-якого рівня. Це дає змогу без проблем вивчати Python на прикладі інших проектів, не маючи незрозумілістей під час читання коду.

Одним з найпопулярніших сфер розробки на Python є написання невеликих скриптів для автоматизації будь-яких процесів та операцій. Це можуть бути аналізатори, виконання послідовності дій та інше. Використання Python для автоматизації зручно тим що:

- Маємо простий синтаксис, який дозволяє дуже бистро писати сценарії поведінки;
- Дуже легку налагодження, пов’язане з тим, що код не компілюється перед запуском.

Стандартна бібліотека Python має дуже велику кількість інструментів для розробки додатку, що дає змогу реалізувати проект будь-якої важкості та спрямованості. До того ж, крім стандартної бібліотеки, написано більше 125000 сторонніх від інших авторів.

Крім того, Python є одним з передових мов програмування для аналізу даних. Популярні бібліотеки такі, як NumPy та matplotlib є потужними інструментами для обробки та візуалізації даних.

Управління пакетами служить сполучною ланкою між різними проектами. За його допомогою можна без проблем писати та обмінюватися пакетами у такому форматі, що інші програмісти мають змогу легко підключатися до інших додатків та проектів.

Python – універсальна мова програмування. Майже нескінченна у типах використання. Розробка додатків та веб-сайтів – не єдине, на що здатен стек технологій. Машинне навчання, нейролінгвістичне програмування, обробка зображень, розробка мобільних та настільних додатків, наука о даних – все це є невеликою частиною можливостей мови програмування Python.

2.2. Вибір бэк-енд фреймворку.

Так як у своєму програмному продукті ми будемо реалізовувати веб-додаток, потрібно обрати Python бек-енд фреймворк. Серед найпоширеніших – Django та Flask. Тому, після довгого дослідження та обговорення був обраний перший – Django.

Головна перевага Django – те, що це най популярніший веб-фреймворк для Python. Це дає нам найбільше суспільство розробників, які мають рішення на будь-яку проблему та можуть допомогти з будь-яким питанням.

Серед очевидних переваг Django – швидкість. Цей фреймворк було розроблено для максимально швидкої розробки додатків. Хоч це і не найголовніша перевага у нашому випадку, але питання швидкості настільки поширене у світі, що про це потрібно було указати.

Django має свої власні додаткові функції, що значно спрощує процес розробки та не змущує програміста звертатися до інших плагінів, проектів та іншого. Серед найпопулярніших: аутентифікація користувача, карта сайту, вміння адміністрації вмісту веб-додатку, RSS та інше.

Однією з головних переваг – реалізація безпеки у Django. Робота з цим фреймворком дає нам захист від помилок, пов’язаних з безпекою сайту, котрі здатні поставити під загрозу весь проект. Серед найпопулярніших помилок безпеки: clickjacking, скриптинг, а також SQL ін’єкції. Тому для ефективного використання аутентифікації користувачів, система логінів та паролів є ключем.

Масштабованість. Веб-фреймворк Django мови Python відмінно відпрацьовує велику кількість трафіку користувачів. Через це велика кількість популярних сайтів та порталів написані за допомогою Django.

Також, особливістю веб-фреймворку Django є те, що він підходить для проектів будь-якої степені важкості. Від навчальних робот до великих корпорацій – з усім здатний працювати цей фреймворк.

Django має величезну купу документації. Окрім величезного суспільства, про Django написано велику кількість як офіційної, так і авторської інформації. Незчисленна кількість прикладів, пояснень та відкритого і добре написаного вихідного коду.

Але серед плюсів використання Django є і негативні сторони:

- Використання маршрутизації з вказівкою на URL;
- Усі компоненти розкриваються спільно;
- Потрібна навичка керування усією системою;
- Все базується на ORM Django;
- Django є монолітним, користувачі не мають змогу змінити звичну для фреймворку поведінку, кардинально повернути її.

Не зважаючі на усі мінуси використання даного веб-фреймворку мови Python Django, він є дуже професіональним, здатним реалізувати будь-яку ідею розробників без проблем, при цьому має безліч плюсів таких, як багата бібліотека функцій, реалізація безпеки сервісу та інше. Також вибір у сторону Django дає змогу адміністратору, який не є програмістом повністю керувати сайтом чи сервісом без ніяких труднощей. З усіма цими задачами повністю справляється стандартна адміністративна панель.

2.3. Вибір фронт-енд фреймворку.

Наступним кроком розробки програмного продукту був вибір фрейморку для фронт-енду проекту. Якщо для вибору мови програмування частини бек-енду був дуже великий та важкий вибір серед безлічі варіантів, фронт-енд налічує всього три фреймворка: AngularJS, React та Vue.JS. Також є ще jQuery, але цей варіант був відкинутий через свій архаїзм, недосконалий код та інше. Тому потрібно було обрати один з трьох інших претендентів.

AngularJS та React – найпопулярніші фреймворки, Vue.JS молодіший та менш популярний. Провівши дослідження, було зрозуміло, що потрібно обирати останній варіант.

AngularJS та React – фреймворки від величезних компаній (Google та Facebook відповідно), у той час як Vue.JS був розроблений однією людиною (колишнім програмістом Google) Еваном Ю. Однією з головних причин створення його – великий поріг входу для початкових програмістів та відсутність можливості написання швидких проектів.

Обраний фреймворк (Vue.JS) використовується у невеликих проектах, якім потрібно додати трохи реактивності, авторизації, користуватись формою для AJAX та інше. Частіше за все його використовують для односторінкових додатків завдяки власним компонентам стану, таким як Router та Vuex. Також на Vue.JS досить зручно використовувати API для створення додатків, але найкраще він підходить для розробки рішень, котрі використовують зовнішні API для обробки даних. Також відмінно підходить для розробки динамічних інтерфейсів, які адаптуються під користувача.

Першою серед переваг Vue.JS відносно інший фронт-енд фреймворків є те, що він єдиний здатний легко інтегруватися в проект, який був написаний без фреймворків. У той час як для інших фреймворків буде потрібно переписувати

велику кількість як JS коду, так і HTML. Vue.JS без проблем інтегрується та тільки доповнює існуючий сервіс.

Великою перевагою є швидкість та розмір файлів Vue.JS. Він має Virtual DOM, який створює копію об'єктного уявлення структурного документа та дозволяє працювати з копією, а не з самим уявленням. Це дуже спрощує написання, підвищує продуктивність та значно прискорює роботу фронт-енд фреймворку.

Для написання на React потрібно мати досвід роботи у написанні JSX та ES2015+. Vue.JS цього не потребує. AngularJS потребує жорстоку структуризацію вашого проекту, Vue.JS проявляє гнучкість у цьому аспекті та є модульним рішенням.

Оптимізувати додаток на Vue.JS значно легше. AngularJS сповільнюється при збільшенні кількості спостерігачів, так як кожен раз при зміні будь-чого у області видимості усі спостерігачі мають бути перезапущеними.

Крива навчання AngularJS та React набагато крутіша. API цих фронт-енд фреймворків величезні, користувачеві потрібно буде розібратися з великою кількістю концепцій щоб стати продуктивним розробником. Звісно, важкість навчання обумовлена їх направленістю тільки на великі, комплексні проекти, але це робить дані фронт-енд фреймворки набагато труднішими для менш досвідчених фронт-енд програмістів.

Серед мінусів Vue.JS можна виділити лише те, що він відносно молодий. Спільнота розробників значно менша, ніж у конкурентів, але зростає досить активно. Тому кількість реалізованих компонентів та написаного коду хоч і невелика, але збільшується. Направленість на односторінкові або невеликі проекти не являється мінусом перед сложнішими конкурентами, а є конструктивною особливістю, так і задуманою. Навпаки, це дає перевагу для початківців або для тих фронт-енд розробників, яким не потрібно робити великий та складний проект.

2.4. Опис програмного продукту.

Для реалізації програмного продукту предикатного перетворювача числових інтервалів було обрано зробити веб-сервіс для перевірки знань, а також надання довідкової математичної інформації будь-якого рівня для користувачів. Реалізовувати почали інструментами, переваги котрих були широко описані у розділах 3.1 – 3.3, а саме бек-енд частина проекту написана на Python з використанням фреймворка Djano. Фронт-енд реалізовано на JavaScript з фреймворком Vue.JS, верстка була написана на HTML + CSS з використанням Bootstrap 4.0, база даних – SQL.

Першим кроком реалізації програмного продукту був аналіз вже існуючих сервісів. Після ознайомлення з усіма відомими нам проектами, було зрозуміло, що не всі мають розширений функціонал: деякі надають лише довідкову математичну інформацію, не маючи можливостей перевірки; інші сервіси надають лише коригувальну функцію, тобто здатні допомогти користувачеві ввести математичну формулу і все; останні сервіси мали змогу перевірити знання, але робили за допомогою тестів, що не повністю задовольняє наші потреби.

Таблиця, що вказана нижче наочно демонструє переваги нашого програмного продукту перед вже існуючими:

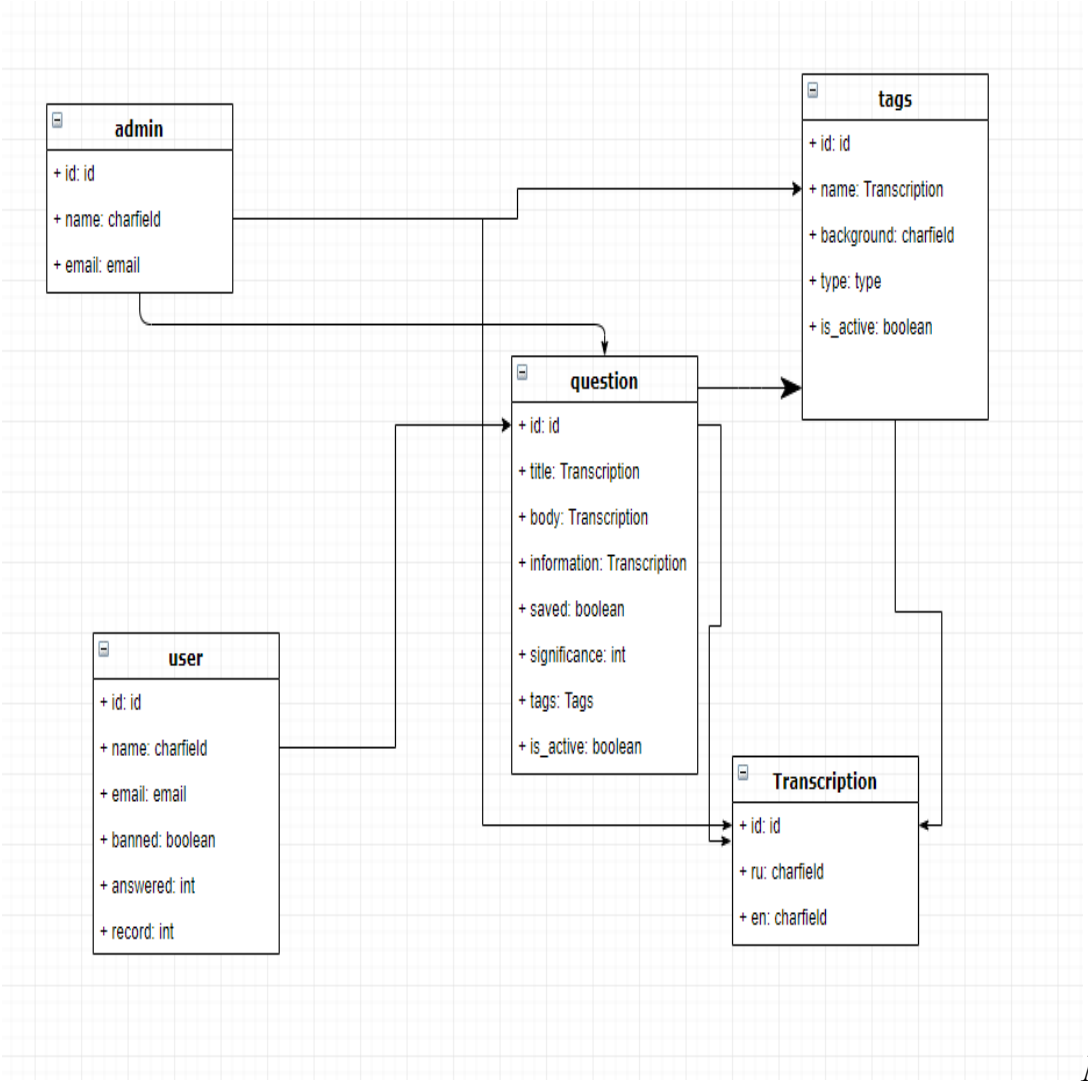
	Назва проекту	**Довідникова інформація**	**Коригування математичних формул**	**Перевірка математичних знань шляхом вводу формул**
	Dpva.ru	+		

	Tehtab.ru	+		
	Dict.scask.ru	+		
	Math Editor		+	
	Wiris.com		+	
	Codecods.com		+	
	Наукова робота	+	+	+

Після аналізу вже існуючих сервісів почався етап проектування. Обдумавши концепцію проекту, було вирішено зробити веб-сервіс, здатний надати довідникову інформацію, мати власний інтерфейс вводу математичних формул та перевірку правильності їх вводу. За перше введення формули має відповідати адміністратор, який поповнює, редагує та знищує всю довідкову інформацію.

Першим кроком розробки програмного продукту предикатного перетворювача математичних інтервалів було створення бази. Перш за все було створено модель Question, що відповідає формулі у нашому проекті. Модель Question має в собі id, title (назва), body (місце для введення формули), information (для введення довідкової інформації), favorite (обране), significance (важність), tags (теги), is_active (для можливості зробити формулу не активною, але не видаляти), а також звичні поля від TimeStampedModel (такі як created та modified). Про поля favorite, significance та tags буде описано далі. Наступною моделлю є вбудована у Django модель Admin. Він нам потрібен для корегування інформації та роботи у

сервісі. Третя модель – User. Це модель користувача, який відповідає на питання. Вона складається з полів id, name (ім’я користувача), email (електронна адреса користувача), banned (можливість заблокувати користувача), answered (кількість правильних відповідей), record (найбільша серія відповідей без помилок). Дві інші моделі виконують допоміжну функцію та мають назви Transcription і Tags. Модель Transcription створена для того, щоб можна було реалізувати у проекті багатомовність. Вона складається з полів id, en та ru (які відповідають за запис інформації англійською та російською мовами). Модель Tags створена для зручної навігації серед формул та має такі поля: id, name (ім’я тегу), background (для навігації за кольором), type та is_active (для можливості зробити тег неактивним). Для наочного дослідження є малюнок №3.1



Мал. 2.1 Діаграма бази даних проекту

Наступним етапом розробки було створення сторінок для нової формули, сторінки усіх формул з адміністративної сторінки, авторизація та клієнтська сторінка, де має бути довідник формул та функціонал перевірки отриманих знань. Для цього у файлі urls.py кореня проекту було зроблено так звані сторінки (path) для кожного функціоналу:

```
from django.urls import include, path
```

```
urlpatterns = [
    path('profile/', views.profile, name='profile'),
    path('singup/', views.singup, name='profile'),
    path('login/', views.login, name='login'),
    path('logout/', views.logout, name='logout'),
    path('edit_question/0/', views.new_question, name='new_question'),
   path('edit_question/<index>/', views.edit_question, name='new_question'),
    path('list_question/', name='list_question'),
    path('list_tags/', name='list_tags'),
 path('library/', name='library'),
path('conversation/', name='conversation'),
]
```

Після створення сторінок, потрібно було написати види для них у файлі views.py кореня проекту предикатного перетворювача числових інтервалів. Написання видів показує Django, який HTML-файл потрібно виконати після переходу на сторінку (тобто, це потрібно для того, щоб на потрібних сторінках проекту предикатного перетворювача числових інтервалів була саме потрібна інформація, яка відповідає адресі сторінки).

Першою сторінкою проекту є авторизація та реєстрація користувача або адміністратора. Для цих цілей в Django реалізована власна модель авторизації django.contrib.auth.models. За її допомоги ми без зайвих труднощів можемо створити форму реєстрації, яка автоматично буде створювати нових користувачів та записувати їх дані до бази даних. Єдине, що остається – змінити стилі сторінки з стандартних Django форм на більш сучасні за допомогою HTML, CSS та Bootstrap.

Після створення користувача та адміністратора, потрібно добавити функціонал створення питання. Для цього потрібно заповнити сторінку верстkою, за допомоги Vue.JS відправити дані до бек-енду та записати їх до бази.

Написавши верстку до сторінки створення потрібно додати фронт-енд фреймворк Vue.JS на сторінку. Спочатку додаємо id зі значенням “app” до головного тегу верстки для того, щоб Vue.JS зміг до нього звертатися за допомогою поля “el”, а вже потім створюємо змінні у полі data для того, щоб потім туди записувати дані, які будуть потім збережені у базі.

```
var new = new Vue({
  el: '#app',
  data: {
    transcription: false,
    favorite: false,
        information_ru: ‘’,
    information_en: ‘’,
    body_ru: ‘’,
        body_en: ‘’,
    title_ru: ‘’,
    title_en: ‘’,
significance: ‘’,
              tags: ‘’,
             }
         })
```

Поля зі змінної data записуються за допомогою директиви v-model, яка дозволяє пов’язувати між собою елементи форми та стани проекту.

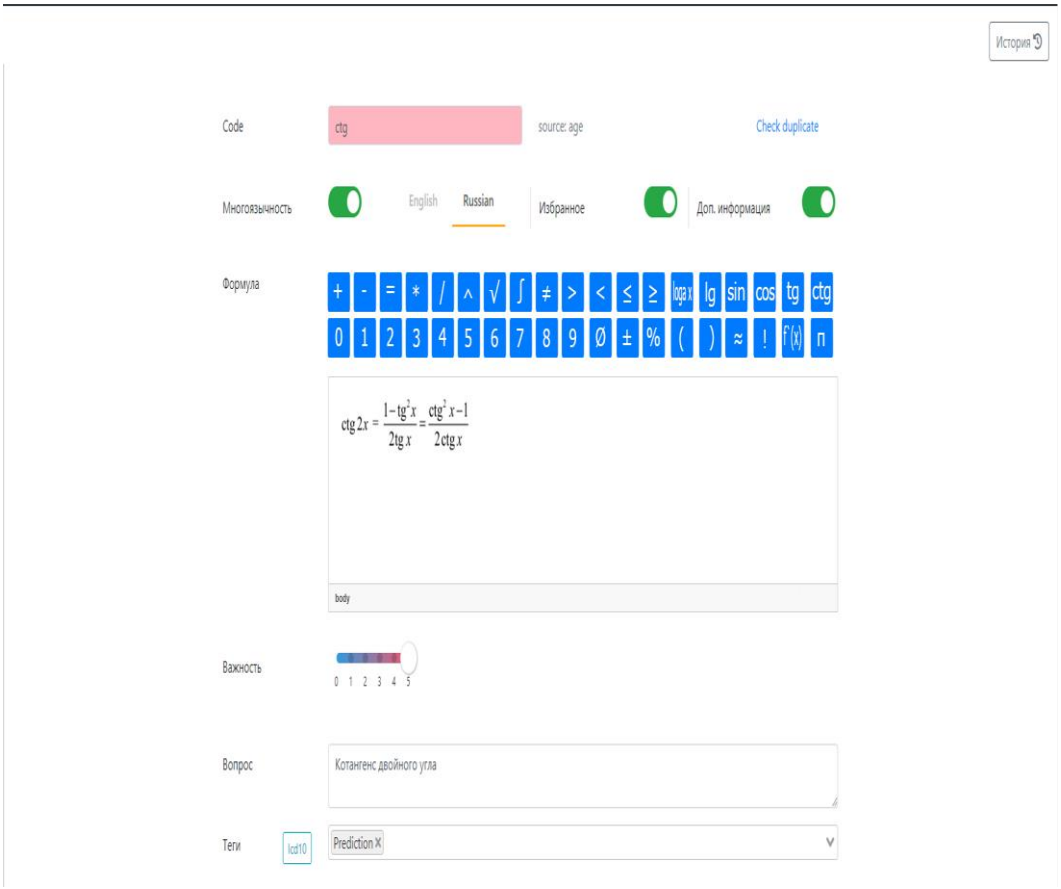
Окремо варто розповісти про функціонал редактора математичних формул. Було реалізовано секцію, котра здатна вводити будь-які математичні формули до поля, а потім зберігати їх у зручному для системи форматі. Усі літерні значення ніяк не залежать від конкретних літер та вміють взаємо заміняти ці значення. Це

дає велике спрощення, як для адміністратора, так і для користувачів тому, що ні першим, ні останнім не потрібно звертати увагу на те, які конкретні літери треба вводити до форми. Усі ці умовності відходять на другий план. Також, при вводі складних функцій (наприклад інтеграл), модуль навчений відкривати модальне вікно для введення допоміжних даних, параметрів, тощо. Це також дуже спрощує роботу з системою.

Після того, як всі дані заповнено, потрібно передати їх на бекенд, де все буде перевірено та записано до бази. З передачею даних з фронт-енду на API бек-енду Vue.JS допомагає axios. Це JavaScript бібліотека, зроблена саме для цих цілей. Все, що потрібно – це визвати axios та передати дані до вже існуючого виду у файлі views.py:

```
axios
  .post("views/edit_question/0", {
    transcription: this.transcription,
    favorite: this.favorite,
          information_ru: this.information_ru,
    information_en: this.information_en,
    body_ru: this.body_ru,
          body_en: this.body_en,
    title_ru: this.title_ru,
    title_en: this.title_en,
  significance: this.significance,
                  tags: this.tags,
  })
  .then(response => console.log(response.data))
  .catch(error => console.log(error));
```

Після відправки даних до API бек-енду, фронт-енд частина закінчується. Але мається ще деякий фунціонал. При першій загрузці сторінки спрацьовує стан mounted, який починає отримувати список тегів, які були створені заздалегідь. Усі вони записуються з бек-енду у форматі JSON та передаються списком до клієнтської частини. Якщо хоча б один тег був створений до того часу – всі вони записуються списком до змінної tags. Після цього вони записуються до поля тегів у формі створення формули за допомогою директиви v-for та виглядають як список. Директива v-select реалізує можливість додавання великої кількості тегів до однієї формули, та записує їх всіх до поля tags. Докладніше на малюнку 3.2:



Мал. 2.2 Сторінка створення формули

Одразу після того, як всі дані були передані, починається їх обробка на бек-енді. За допомогою функціоналу try/except отримана інформація перевіряється на помилки, та при не виявленні їх пропускає дані до іншого етапу.

Після етапу перевірки валідності даних починається етап створення об’єкту моделі Question та запису даних до його полів:

```
new_q = Question.objects.create()
new_q.transcription = transcription
new_q.favorite = favorite
new_q.information_ru = information_ru
new_q.information_en = information_en
new_q.body_ru = body_ru
new_q.body_en = body_en
new_q.title_ru = title_ru
new_q.title_en = title_en
new_q.significance = significance
new_q.tags = tags
new_q.save()
```

При додаванні до формули тегів, вони записуються по типу ManyToMany та створюють окрему таблицю у базі даних під назвою question_tags. У цій таблиці можна побачити усі залежності та зв’язки формул та прив’язаних до цих формул тегів.

Наступним етапом було створення сторінки списку формул з можливістю переходу на сторінку кожної формули. Реалізовано це було також за допомогою Vue.JS та axios. При переході на сторінку списку усіх формул, директива mounted звертається до окремого виду з файлу views.py, котрий передає за допомогою GET запиту до неї усі формули, відсортовані за id. Так це виглядає.

```
All_q = Question.objects.all(order_by=’id’)
```

Потім отримані дані форматуються у JSON формат та відправляється до фронт-енду за допомогою axios:

```
data: {
        questions: ‘’,
}

axios
    .get('views/all_questions/')
    .then(if (!response.error) => (this.questions = data.response));
```

Отримавши дані, Vue.JS починає заповнювати ними таблицю за допомогою все тієї ж директиви v-for, ставлячи у кожну клітину поле даних об’єкту формули. Клітинка з тегами має свій власний v-for для запису всіх тегів по типу запису полів формули. Виглядає це так:

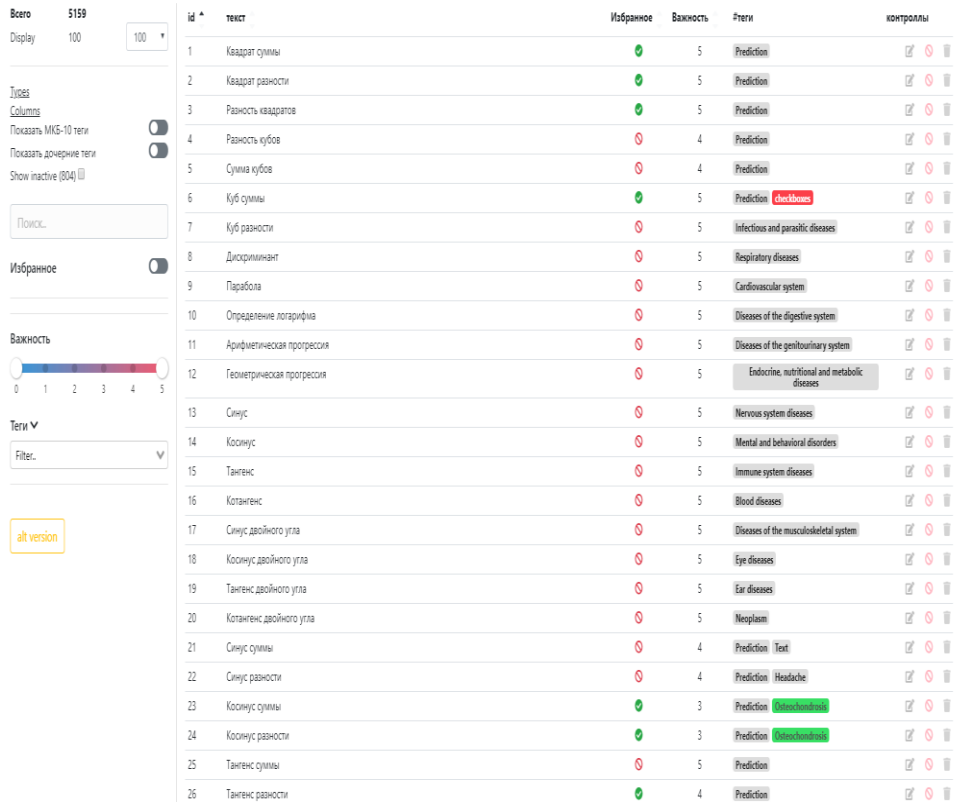
```
<tr v-for=”q in questions”>
        <td>{[q.id]}</td>
   <td>{[q,title]}</td>
   <td>{[q,favorite]}</td>
   <td v-for=”tag in q.tags”>{[tag]}</td>
   <td>{[q.significance]}</td>
   <td>{[q.is_active]}</td>
</tr>
```

На цій сторінці ми маємо функціонал пошуку за назвою, важливістю та прив’язаними тегами. А також можна перейти на сторінку питання, зробити його неактивним або взагалі видалити. При переході на сторінку питання, відкривається точно така сама сторінка, як і при створенні формули, але з вже

заповненими полями даних. Ці поля стають заповненими через те, що директива mounted за допомогою axios відправляє запит до бек-енду та тягне потрібну інформацію з потрібної формули. Визначається ця формула завдяки посиланню у якому і зберігається потрібний нам id.

```
axios
    .get('views/questions/' + this.id + '')
    .then(if (!response.error) => (
            this.transcription: data.response('this.transcription');
      this.favorite: data.response('this.favorite');
            this.information_ru: data.response('information_ru');
      this.information_en: data.response('information_en');
      this.body_ru: data.response('body_ru');
            this.body_en: data.response('body_en');
      this.title_ru: data.response('title_ru');
      this.title_en: data.response('title_en');
      this.significance: data.response('significance');
                  this.tags: data.response('tags');
  ));
```

Наочніше на малюнку 3.3:



Мал. 2.3 Сторінка списку всіх формул

Останньою за списком, але не за значенням є сторінка користувача. За ідеологією це мало бути довідково-навчальним центром, де всі бажаючі змогли б вивчити інформацію за темою, а потім перевірити свої знання шляхом проходження завдання математичної тематики.

Тож, для клієнтської сторінки програмного модуля предикатного перетворювача числових інтервалів було придумано зробити розділення на 2 частини: довідникова інформація, яка містить в собі весь навчальний матеріал у вигляді списку формул з додатковою інформацією. Цей навчальний матеріал можна дуже зручно вивчати та засвоювати інформацію. А вже потім переключатися у режим редагування та перевірки себе.

Режим перевірки представляє собою сторінку з можливістю вибору типу завдання та типу рішення цього завдання для користувача. Наприклад, він має змогу як за формулою визначити назву, так і маючи назву правила, ввести коректну формулу. При виборі другого типу завдання користувачеві також надається вибір важкості виконання: він може посимвольно вводити потрібну

формулу або обрати більш важкий варіант і отримати результат свого введення у редактор проекту предикатного перетворювача чисельних інтервалів тільки після підтвердження закінчення введення. Це дозволяє урізноманітнити процес проходження завдань.

Процес переключання між довідником та редактором реалізовано за допомогою директиви v-if/v-else у тегах верстки сторінки користувача проекту та значення isDirectory:

```
<div v-if=”isDirectory”>
        /*контент з довідником*/
<div>
<div v-else>
        /*контент з перевіркою завдань*/
</div>
```

Перевірку введення формули на правильність було реалізовано так: так як для посимвольного та звичайного вводу потрібні буди різні валідатори – було прийнято рішення розділити їх так само як довідник та редактор за допомогою директиви v-if/v-else та змінної isBySymbols. Звичайна перевірка мала в собі звичний інтерфейс: після введення свого варіанту користувач підтверджував введення кнопкою «Підтвердити». Після чого формула у вигляді строки разом з id формули відправлялася на бек-енд, де у виді перевірки визначався об’єкт question по id та перевірялося body з тим варіантом, який прийшов від користувача. Головною особливістю є те, що користувач не обмежений у використанні літерних символів і може використовувати будь-який, система вивчена обходити це. При успішній перевірці користувач отримував сповіщення про те, що його відповідь вірна. Якщо він не відповідає правильно – отримує зворотне сповіщення, має

змогу ще раз відправити вже нову відповідь або дізнатися правильну відповідь, натиснувши на кнопку «Правильна відповідь», а підтвердивши спливаюче вікно з підтвердженням дії. Посимвольна перевірка має більш «хитру» реалізацію як на фронт-енді, так і на бек-енді. При виборі такого типу перевірки запит відправляється при будь-якій зміні поля введення, тобто як тільки користувач введе перший символ, він відразу матиме відповідь, чи правильно він почав. На бек-енді для чіткої перевірки розраховується отримана відповідь у символах, отримане число строки відіймається від записаної у бази правильної формули та перевіряється. При введенні правильного варіанту система ніяк не реагує, але сигналізує про неправильне введення при першій помилці та пропонує переглянути правильний варіант. Можливість введення будь-яких літерних символів також збережена, як і в звичайній версії, тому з цим у користувачів проекту предикатного перетворювача числових інтервалів не буде ніяких проблем.

Не дивлячись на те, що другий спосіб перевірки введення посимвольно може здатися важко виконуючим для системи, ніяких зависань та довгих виконань після великої кількості тестів на формулах будь-якої важкості виявлено не було. Для більшої наочності є малюнок 3.4:

Мал. 2.4 Сторінка користувача

Після реалізації проекту при його відносної популярності маємо ідеї такого разширення функціоналу: перш за все, реалізація історії змін питань, формул та тегів. Це значно полегшить відстеження змін, дасть змогу отримувати інформацію про те, хто з адміністраторів ці зміни ввів. Наступним поліпшенням функціоналу є строге зв’язання формул та користувача. Це нам дасть можливість відстежувати активність та поведінку нашого користувача, а у майбутньому ще й розробити систему оцінок. Ну і останнім нововведенням буде редизайн, який має поліпшити сприйняття проекту.

На наш погляд, ці зміни мають надати вже діючому проекту більшу гнучкість, що приведе за собою більшу кількість користувачів.

ВИСНОВКИ

У дисертації було здійснено огляд і вивчення числових інтервалів, предикатів та їх логики. Був створений програмний модуль для реалізації редактора та валідатора математичних формул, що дає значну допомогу для користувачів. Також сервіс має ще й довідникову функцію, це дасть змогу ще ефективніше використовувати та удосконалювати знання, отримані під час самостійного вивчення.

Програмний модуль було спроектовано мови програмування Python з використанням їх власного бек-енд фреймворку Django. Клієнтська частина написана за допомогою верстки на HTML, CSS та Bootstrap. Фронт-енд написан на фреймворці Vue.JS, база даних – SQL.

Модуль є сучасним інструментом, що значно прискорює отримання та вдосконалення знань з математики.

Завдяки проекту будь-який бажаючий здатен перевірити себе, витратити час на вдосконалення знань. Також, на базі проекту можна постійно випускати оновлення з новим функціоналом, що не дасть йому стати черговим сервісом на один день.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шевченко, В.Н., Золотых, Н.Ю.: Линейное и целочисленное линейное программирование. Нижний Новгород Издательство Нижегородского гос. Университета (2005)

2. Емеличев, В.А., Ковалев, М.М., Кравцов, М.К.: Многогранники Графы Оптимизация. Москва Наука (1981)

3. Циглер, Г.М.: Теория многогранников. Пер. с англ. под ред. Долбилина Н.П. Москва МЦНМО (2014)

4. Схрейвер, А. Теория линейного и целочисленного программирования. Пер с англ. под ред. Хачияна Л.П. Москва Мир (1991)

5. Schrijver, A.: Theory of Linear and Integer Programming. WileyInterscience series in discrete mathematics. John Wiley & Sons (1998)

6. Ziegler, G.: Lectures on polytopes. Springer-Verlag, GTM 152 (1996) Gr¨unbaum, B.: Convex Polytopes. Springer-Verlag, New York (2003)

7. Груздев Д. В. Модификация алгоритма Фурье — Моцкина для построения триангуляции // Информ. бюл. Ассоциации математического программирования. Вып. 10. XII Всерос. конф. «Математическое программирование и приложения» (тез. докл.). Екатеринбург: УрО РАН, 2003. С. 89-90.

8. Моцкин Т. С , Райфа Х., Томпсон Дж. Л., Тролл Р. М. Метод двойного описания: Матричные игры. М.: Физматгиз, 1961.

9. Понтрягин Л. С. Основы комбинаторной топологии. М.: Наука, 1976.

10. Препарата Ф., Шеймос М. Вычислительная геометрия: Введение. М.: Мир, 1989.

11. Черников С. Н. Линейные неравенства. М.: Наука, 1968. 6. Шевченко В. Н. Качественные вопросы целочисленного программирования. М.: Физматлит, 1995.

12. Шевченко В. Н. О разбиении выпуклого политопа на симплексы без новых вершин // Изв. вузов. Математика. 1997. № 12. С. 89-99.

13. Шевченко В. Н. О максимальных триангуляциях выпуклых политопов // Междунар. конф. «Дискретный анализ и исследование операций». Материалы конф. (Новосибирск, 26 июня - 1 июля 2000). Новосибирск: Изд-во Ин-та математики, 2000. С. 163.

14. Шевченко В. Н., Груздев Д. В. О минимальном разбиении выпуклого многогранника на тетраэдры // Математическое моделирование и оптимальное управление. Нижн. Новгород, 1998. С. 184-193. (Вести. Нижегородского ун-та; Вып. 1(18)).

15. Черников С.Н. Линейные неравенства. М.: Наука, 1968.

16. Ziegler G. Lectures on Polytopes. New York: Springer Verlag, 1998.

17. Черников С.Н. Свертывание систем линейных неравенств // Докл. АН СССР. 1960. Т. 131. No 3. С. 518–521.

18. Черников С.Н. Свертывание конечных систем линейных неравенств // Ж. вычисл. матем. и матем. физ.

19. 1965. Т. 5. No 1. С. 3–20.

20. Моцкин Т.С., Райфа Х., Томпсон Д.Л., Тролл Р.М. Метод двойного описания // Матричные игры. М.:

21. Физматгиз, 1961.

22. Fukuda K., Prodon A. Double description method revisited // Combinatorics and Computer Science. Springer–

23. Verlag, 1996. P. 91–111.

24. Золотых Н.Ю. Новая модификация метода двойного описания для построения остова многогранного

25. конуса // Ж. вычисл. матем. и матем. физ. 2012. Т. 51. No 1. С. 153–163.

26. Лукацкий А.М., Шапот Д.В. Конструктивный алгоритм свертывания систем линейных неравенств вы#

27. сокой размерности // Ж. вычисл. матем. и матем. физ. 2008. Т. 48. No 7. С. 1167–1180.

28. Деза М.М., Лоран М. Геометрия разрезов и метрик. М.: МЦНМО, 2001.

29. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. М.: Мир. 1979.

30. Бренстед А. Введение в теорию выпуклых многогранников. М.: Мир. 1988.

31. Гельфанд И.М., Зелевинский А.В., Капранов М.М. Дискриминанты много-членовот многих переменных и триангуляции многогранников Ньютона // Алгебра и анализ. 1990. Т.2, Вып. 3. С.1-63.

32. Груздев Д.В. Описание множества /-векторов триангуляций 4-мерного куба. // Материалы четвертой молодежной научной школы по дискретной математике и ее приложениям. Москва: Изд-во мех.-мат. ф-та МГУ. 2000. С.28-32.

33. Груздев Д.В. Модификации алгоритма Фурье-Моцкина для построения триангуляций, их разверток и звездных разверток для точечных конфигураций / Нижегор. гос. ун-т. Н. Новгород, 2006. - 54 с. - Деп. в ВИНИТИ 07.02.06 N 131-В2006.

34. Делоне В.Н. О пустом шаре // Известия АН СССР. 1934. VII серия, 6. С.793-800.

35. Емеличев В.А., Ковалев М.М., Кравцов М.К. Многогранники, графы, оптимизация. М: Наука. 1981.

36. Зыков А.А. Основы теории графов. М: Наука. 1987. С.118.

37. Математический энциклопедический словарь. М: Сов. энциклопедия. 1988.

38. Моцкин Т.С., Райфа Х., Томпсон Дж.Л., Тролл Р.М. Метод двойного описания. // Матричные игры. Физматгиз. Москва. 19G1

39. Понтрягин Л.С. Основы комбинаторной топологии. ОГИЗ, 1947.

40. Препарата Ф., Шеймос М. Вычислительная геометрия: Введение. М.: Мир. 1989.

41. Черников С.Н. Линейные неравенства. М.: Наука, 1968.

42. Шевченко В.II. Качественные вопросы целочисленного программирования. М.: Физматлит. 1995.

43. Шевченко В.И. О разбиении выпуклого политопа на симплексы без новых вершин. Известия ВУЗ. Математика. N12. 1997. С.89-99.

44. Шевченко В.Н. О максимальных триангулнциях выпуклых политопов. // Международная конференция "Дискретный анализ и исследование операций": Материалы конференции. Новосибирск: Изд-во Ин-та математики. 2000.

45. Шевченко В.II., Груздев Д.В. Модификация алгоритма Фурье-Моцкина для построения триангуляции и ее звездной развертки для точечной конфигурации в общем положении / Нижегор. гос. ун-т. II. Новгород, 2006. - 22 с. - Деп. в ВИНИТИ 07.02.06 N 132-В2006.

46. 18. Шевченко В.II., Чирков АЛО. О покрытии конечно порожденного конуса элементарными конусами. // Инф. бюллетень N4 ассоциации мат. программирования. Екатеринбург, 1993.

47. 19. Энциклопедия элементарной математики. Кн.5. Геометрия. М.: Наука. 1966.

48. Below A., De Loera J.A., Richter-Gebert J. Finding minimal triangulations of convex 3-polytopes is NP-hard. In proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms. January 9-11, 2000.

49. Bohm J. Complete enumeration of all optimal vertex preserving triangulationsof the 5-cube. Forschungsergebnisse, Fredrich-Shiller Universitat, Jena. N/88/36. 1988. P.l-16.

50. Bruggesser H., Mani P. Shellable decompositions of cells and spheres // Math. Scand. V.29. 1971. P.197-205.

51. Burger E. Uber homogene lineare Ungleichungssysteme // Zeitschrift Angewandte Math, und Mech., 1956. V.36. P.135-139.

52. Cottle R.W. Minimal triangulations of the 4-cube // Discrete Math. 1982. V.40. P.25-29.

53. Gruber P.M., Ryskov S.S. Facet-to-facet implies face-to-face // European J. Combin. 1989. V.10. P.83-84.

54. Grunbaurn B. Convex polytopes. N.Y.: Wiley, 1967.

55. Hughes R.B. Lower bounds on cube simplexity // Discrete Math. 1994. V.133. P. 123-138.

56. Hughes R.B. Minimum-cardinality triangulations of the d-cube for d=5 and d=6 // Discrete Math. 1993. V.118. P.75-118.

57. Hughes R.B., Anderson M.R. A triangulation of the 6-cube with 308 simlices // Discrete Math. 1993. V.133. P.253-256.

58. Klee V. A combinatorial analogue of Poincare's duality theorem // Canad. J. Math. 1964. V. 16. P.517-531.

59. Lee C.W. Regular triangulations of convex polytopes, in Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. 1991. V.4. P.443-456.

60. De Loera J.A. Nonregular Triangulations of Products Of Simplices // Discrete Comput. Geom. 1996. V.15. P.253-264.

61. Mara P.S. Triangulations of the cube // J. Coinbin. Theory Ser. A. 1976. V.20. P. 170-177.

62. Rudin M.E. An unshellable triangulation of a tetrahedron // Bulletin of the American Mathematical Society. 1958. V.64. P.90-91.

63. Sallee J.F. A triangulation of the n-cube // Discrete Math. 1982. V.40. P.81-86.

64. Sallee J.F. A note on minimal triangulations of an n-cube // Discrete Appl. Math. 1982. V.4. P.211-215.

65. Scarf II. The approximation of fixed points of a continuous mapping. // SIAM J. Appl. Math. 1967. V.15, N 5.

66. Stanley R.P. Combinatorics and Commutative Algebra. Series: Progress in mathematics. V.41. Birkhaiiser Boston, 1996.

67. Todd M.J., Tuncel L. A new triangulation for sirnplicial algorithms // SIAM J. Discrete Math. 1993. V.6. P.167-180.

68. Ziegler G.I. Lectures on convex poly topes. Springer-Verlag. 1994.