МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

другого (магістерського) рівня вищої освіти

на тему: **«Веб-орієнтований ужиток для роботи з контролерами нечіткої логіки»**

Виконав: студент групи Іт-61

Спеціальності 126 «Інформаційні системи та технології»
(шифр і назва)
Іванців Тарас Богданович
(Прізвище та ініціали)

Керівник: к.т.н., в.о. доцента Ковалишин О.С.
(Прізвище та ініціали)

Рецензент: к.т.н., доцент Станько В.Ю.
(Прізвище та ініціали)

ДУБЛЯНИ-2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Другий (магістерський) рівень вищої освіти
Спеціальність 126 «Інформаційні системи та технології»

«ЗАТВЕРДЖУЮ»

Завідувач кафедри_____

д.т.н., проф. А.М. Тригуба
«____» _____2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу студенту

Іванціву Тарасу Богдановичу

1. Тема роботи: «Веб-орієнтований ужиток для роботи з контролерами нечіткої логіки»

Керівник роботи Ковалишин Олег Степанович, к.т.н., в.о. доцента
затверджені наказом по університету від 12.09.2024 року № 616/к-с.

2. Строк подання студентом роботи 06.12.2024 р.

3. Вихідні дані до роботи: *дані про мікроконтролери нечіткої логіки; методи і алгоритмів побудови контролерів нечіткої логіки*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)
Вступ.
1. Характеристика web-ужитку для роботи з контролерами нечіткої логіки
2. Огляд літературних джерел
3. Системний аналіз
4. Концептуальна модель
5. Розробка web-ужитку
6. Економічна характеристика проектного рішення
Висновки
Список використаної літератури

5. Перелік ілюстраційного матеріалу (з точним зазначенням обов'язкових слайдів): *ілюстративний матеріал у вигляді презентації (до 15 слайдів) як додаток до доповіді за темою дипломної роботи*.

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3, 4, 5	*Ковалишин О.С, к.т.н., в.о. доцента кафедри інформаційних технологій*		

7. Дата видачі завдання 12 вересня 2024 р.

Календарний план

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів роботи	Примітка
1	*Написання першого розділу*	*12.09-20.09.24*	
2	*Виконання другого розділу та аркушів ілюстраційного матеріалу до нього*	*21.09-14.10.24*	
3.	*Виконання третього та четвертого розділів та аркушів ілюстраційного матеріалу до них*	*15.10-10.11.24*	
4.	*Написання п'ятого розділу та аркушів ілюстраційного матеріалу до нього*	*11.11-20.11.24*	
5.	*Оцінення ефективності проектних рішень*	*21.11-30.30.24*	
6.	*Завершення оформлення розрахунково-пояснювальної записки та аркушів ілюстраційного матеріалу*	*01-04.12.24*	

Студент _____ Іванців Т.Б.
(підпис)

Керівник роботи _____ Ковалишин О.С.
(підпис)

УДК 004.8:631.1

Веб-орієнтований ужиток для роботи з контролерами нечіткої логіки.

Іванців Т.Б. Кафедра інформаційних технологій – Дубляни, ЛНУП, 2024.

Кваліфікаційна робота: 90 с. текст. част., 13 рис., 7 табл., 14 арк. ілюстраційного матеріалу, 23 джерела.

Проведено огляд літературних джерел, що підтвердив актуальність досліджуваної теми. Розроблено дерево цілей, яке систематизує завдання проекту, та проаналізовано стратегії їх досягнення. Розроблено web-ужиток для роботи з контролерами нечіткої логіки.

Оптимізовано швидкісний алгоритм нечіткого логічного виведення для використання широким колом спеціалістів.

Створено інтуїтивно зрозумілий користувацький інтерфейс, що полегшує налаштування контролерів.

Побудовано концептуальну модель системи, що відображає структуру компонентів, методи їхньої взаємодії всередині системи та з зовнішнім середовищем.

Крім того, виконано економічний аналіз проектного рішення, який підтвердив його ефективність та доцільність впровадження.

Ключові слова: контролери нечіткої логіки, веб-орієнтований ужиток, користувацький інтерфейс.

ВСТУП

Теорія нечітких множин широко застосовується у створенні контролерів нечіткої логіки, які є важливим інструментом у сучасних складних системах управління. Відмінною особливістю цих контролерів є те, що вони працюють не за традиційними алгоритмами чи диференціальними рівняннями, а на основі експертних знань. Це робить їх надзвичайно гнучкими та здатними адаптуватися до широкого спектра задач, включаючи ті, де вхідні дані неможливо точно визначити або вони мають динамічний характер.

Для опису системи нечіткої логіки використовуються лінгвістичні змінні, які моделюють реальні процеси через нечіткі множини. Завдяки цьому контролери нечіткої логіки ефективно працюють у ситуаціях, де традиційні методи виявляються надто громіздкими чи взагалі непридатними.

Основні переваги систем нечіткої логіки

1. **Робота з нечіткими даними**. Системи нечіткої логіки здатні обробляти вхідні дані, які задаються у вигляді оцінок чи значень, що змінюються у часі. Наприклад, вони можуть аналізувати результати соціологічних опитувань, дані маркетингових досліджень або інформацію, яка має імовірнісний характер.
2. **Формалізація нечітких критеріїв**. Контролери можуть працювати з такими критеріями, як "більшість", "можливо", "переважно", що дозволяє адаптувати їх під реальні умови, де точність формулювання є неможливою.
3. **Оцінювання якісних характеристик**. Нечітка логіка дозволяє аналізувати не лише числові значення даних, а й їхню якісну природу, враховуючи ступінь вірогідності чи ймовірності.
4. **Швидке моделювання складних систем**. Завдяки використанню принципів нечіткої логіки, складні динамічні системи можуть бути

змодельовані у короткі терміни, без необхідності детального математичного опису кожного параметра.

На ринку України технології нечіткої логіки почали активно розвиватися з середини 1990-х років. Основними споживачами таких систем стали фінансові установи, аналітичні центри та підприємства, які займаються політичним і економічним аналізом. Ці системи широко застосовувалися для моделювання сценаріїв у різних галузях, від біржової торгівлі до маркетингових стратегій.

У світовій практиці нечітка логіка використовується у багатьох сферах, таких як транспорт, побутова техніка, військові системи управління та ситуаційні центри. Наприклад, системи керування залізничними потягами або пральними машинами часто базуються на алгоритмах нечіткої логіки. Особливо цікавою є її роль у кризовому управлінні, де вона застосовується для аналізу складних політичних і економічних ситуацій.

Важливим прикладом промислового застосування нечіткої логіки стало моделювання системи охорони здоров’я Великобританії (NHS), що дозволило не лише скоротити витрати на соціальні потреби, а й запровадити інноваційні підходи до планування витрат.

Сьогодні на ринку представлено більше 100 програмних рішень, які інтегрують елементи нечіткої логіки. Одним із лідерів є американська компанія Hyper Logic, яка спеціалізується на експертних системах. Їхній продукт CubiCalc пропонує потужні інструменти для розробки систем управління на основі нечіткої логіки. Інші компанії, як-от IntelligenceWare, InfraLogic та Aptronix, також роблять вагомий внесок у розвиток цієї галузі.

На території СНД популярним рішенням є російський програмний продукт "Бізнес-прогноз". Цей пакет допомагає бізнесу оцінювати ризики та прибутковість інвестиційних проєктів, надаючи простий і зрозумілий інтерфейс для роботи навіть з неточними даними.

Метою дослідження є розробка веб-застосунку для налаштування контролерів нечіткої логіки з можливістю їх використання у вирішенні специфічних задач.Досягнення поставленої мети включало розвязання таких задач:

1. Провести аналіз існуючих джерел інформації з тематики.
2. Виконати системний аналіз предметної області.
3. Розробити концептуальну модель системи.
4. Створити функціональний веб-застосунок.
5. Оцінити економічну ефективність розробленого рішення.

Об’єктом дослідження є процеси імплементації контролерів нечіткої логіки з використанням уніфікованих web-інтерфейсів.

Предметом дослідження є алгоритми та засоби побудови web-ужитків.

В результаті проведених досліджень отримані наступні наукові результати:

1. Розроблено алгоритм, який забезпечує одночасний доступ до сервера для 100 користувачів без втрати продуктивності.
2. Оптимізовано швидкісний алгоритм нечіткого логічного виведення для використання широким колом спеціалістів.
3. Створено інтуїтивно зрозумілий користувацький інтерфейс, що полегшує налаштування контролерів.
4. Реалізовано веб-застосунок, який підтримує швидкий доступ та роботу в реальному часі, забезпечуючи високу ефективність для кінцевих користувачів.

РОЗДІЛ 1. ХАРАКТЕРИСТИКА WEB-УЖИТКУ ДЛЯ РОБОТИ З КОНТРОЛЕРАМИ НЕЧІТКОЇ ЛОГІКИ

Контролери нечіткої логіки широко застосовуються в автоматизації технологічних процесів, а також у робототехніці, де вони використовуються для управління рухом роботів. Завдяки здатності працювати з неповними та нечіткими даними, ці системи є ефективним інструментом у складних динамічних середовищах. Перспективним напрямом розвитку нечітких логічних систем є використання спеціалізованих web-застосунків, які дозволяють налаштовувати контролери на вирішення конкретних задач. Розміщення таких застосунків на віддалених серверах скорочує витрати на розробку, забезпечує зручний доступ до найновіших технологій та спрощує оновлення системи. Це особливо важливо для підприємств, які прагнуть впроваджувати сучасні інновації з мінімальними фінансовими затратами.

Web-застосунки для роботи з контролерами нечіткої логіки розробляються висококваліфікованими фахівцями, що гарантує їхню відповідність сучасним технічним вимогам. Постійне оновлення таких систем дозволяє забезпечити високу продуктивність, точність і надійність контролерів, які адаптуються до конкретних задач. Метою магістерської роботи є створення web-орієнтованої системи, яка дає змогу швидко створювати і налаштовувати контролери нечіткої логіки за допомогою веб-інструментів. Крім того, система забезпечуватиме можливість доступу до контролерів і роботи з ними у режимі реального часу.

На високому рівні абстракції система складається з двох основних компонентів. Перший – це компонент роботи з нечіткою логікою, що відповідає за ініціалізацію та інстанціювання контролерів на основі заданих конфігурацій, а також за управління цими контролерами у реальному часі. Другий компонент – це модуль роботи з веб, який надає графічний web-інтерфейс для налаштування контролерів, забезпечує механізми

довготривалого збереження конфігурацій, а також платформонезалежні web-сервіси для ініціалізації контролерів на основі раніше створених налаштувань. Крім того, цей компонент відповідає за швидкісний обмін даними для роботи з контролерами в реальному часі, що є критично важливим для задач, які потребують оперативного реагування.

Особлива увага приділяється дизайну користувацького інтерфейсу, який має бути мінімалістичним і зручним для користувачів. Структура інтерфейсу передбачає поділ на сторінки, кожна з яких виконує одну конкретну функцію, що значно знижує ризик помилок. Дані, введені користувачами, проходитимуть перевірку на коректність, що забезпечить стабільність роботи системи навіть за інтенсивного навантаження. Система також повинна витримувати до 100 паралельних сесій користувачів, при цьому забезпечуючи час відгуку не більше однієї секунди.

Для розробки web-застосунку обрано мову програмування Java версії 1.7, яка забезпечує надійність і високий рівень продуктивності. Процес розробки здійснюватиметься з використанням інтегрованого середовища IntelliJ IDEA. Бібліотека TController буде використана для реалізації алгоритмів нечіткої логіки, що забезпечить необхідну функціональність для роботи з контролерами. Такий підхід дозволяє створити ефективну, зручну у використанні та масштабовану систему, яка відповідатиме сучасним вимогам до автоматизованих систем управління.

РОЗДІЛ 2. ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ

2.1. Огляд архітектури веб-застосунків

Веб-застосунок, або веб-сайт (від англ. website, що перекладається як "місце" чи "майданчик" в Інтернеті), є сукупністю веб-сторінок, доступних у мережі Інтернет. Ці сторінки об'єднані спільним змістом та навігаційними засобами, що робить їх єдиним цілісним ресурсом. З фізичної точки зору, сайт може розміщуватися як на одному сервері, так і на декількох, залежно від його масштабів і технічних вимог.

Термін "сайт" також використовується для позначення вузла мережі Інтернет, тобто комп'ютера з унікальною IP-адресою, а також будь-якого об'єкта в Інтернеті, що має адресацію для ідентифікації в мережі (наприклад, FTP-сайт, WWW-сайт тощо). Веб-сайт являє собою набір взаємопов'язаних інформаційних ресурсів, які можна переглядати через комп'ютерну мережу за допомогою браузерів. Веб-вузол може бути представлений як набір електронних документів, онлайнових сервісів тощо.

Основним елементом сайту є веб-сторінка — електронний документ, написаний здебільшого на мові розмітки Hypertext Markup Language (HTML або XHTML). Веб-сторінка може містити текст, зображення, відео, а також елементи, запозичені з інших сайтів, за умови підтримки відповідних форматів розмітки [1].

Передача веб-сторінок між сервером і клієнтом здійснюється через протокол передачі гіпертексту (HTTP), який може використовувати шифрування для безпеки (HTTPS). Браузер інтерпретує HTML-розмітку, візуалізуючи її на екрані користувача у зрозумілому вигляді. Доступ до сторінок веб-сайту зазвичай здійснюється через Уніфікований Локатор Ресурсу (URL), який визначає ієрархічну організацію сторінок у межах одного ресурсу.

В основі функціонування веб-сайтів лежить клієнт-серверна архітектура [2]. Ця архітектура є базовим шаблоном для створення розподілених мережевих застосунків і включає три основні компоненти: сервери, клієнти та мережу, яка забезпечує їхню взаємодію. Сервери надають ресурси або послуги, клієнти — отримують ці послуги, а мережа відповідає за обмін даними між ними.

Ключовою особливістю клієнт-серверної архітектури є незалежність клієнтів і серверів. Сервери можуть обробляти численні запити від різних клієнтів одночасно, а клієнти можуть звертатися до кількох серверів за потреби. Взаємодія здійснюється на трьох рівнях: рівні представлення даних (інтерфейс користувача), прикладному рівні (логіка роботи програми) та рівні управління даними (збереження та обробка інформації).

Клієнт-серверна архітектура поділяється на дві основні моделі: модель тонкого клієнта і модель товстого клієнта. У моделі тонкого клієнта вся логіка застосунку і управління даними зосереджені на сервері, а клієнт забезпечує лише представлення даних користувачу. Це підхід, який часто використовується в мобільних пристроях, кишенькових комп'ютерах тощо. У моделі товстого клієнта, навпаки, сервер обмежується управлінням даними, тоді як обробка інформації і взаємодія з користувачем виконується на клієнтській стороні [4].

З середини 1990-х років почала активно розвиватися трирівнева клієнт-серверна архітектура, яка передбачає розділення прикладної логіки і управління даними. З’явився окремий проміжний рівень для обробки запитів. Програми цього рівня часто працюють під управлінням серверів застосунків або веб-серверів, забезпечуючи додаткову функціональність. Управління даними покладається на сервери баз даних, найчастіше організованих за реляційною моделлю.

Для взаємодії користувача із системою використовується стандартне програмне забезпечення — веб-браузери. Це позбавляє потреби

встановлювати додаткові програми, хоча в окремих випадках такі програми можуть бути необхідними. Веб-оглядачі формують запити, які пересилаються на сервер. Сервер обробляє запити, взаємодіє із програмними модулями проміжного рівня та базами даних, а результати повертаються користувачеві. Дані з бази зазвичай передаються в обробленому вигляді, щоб забезпечити їхню готовність до відображення в браузері.

Веб-сервер, що є основою всесвітньої павутини, приймає HTTP-запити від клієнтів і формує відповіді, часто разом із HTML-документами, зображеннями чи іншими ресурсами. Як програмне забезпечення, так і апаратна платформа, веб-сервер забезпечує централізоване управління взаємодією з клієнтами [5]. Клієнти отримують доступ до веб-сервера через URL-адресу потрібної веб-сторінки або ресурсу.

2.2. Огляд сучасних технологій розробки та підтримки веб-сайтів

Модернові технології розробки та підтримки веб-сайтів базуються на потужних платформах, які дозволяють ефективно управляти інформаційним наповненням, а також обробляти дані, що надходять від користувачів. Як правило, ці рішення спираються на серверні технології, такі як ASP, ASP.NET, JSP, PHP, або використовують готові інструменти для створення організаційних та промислових сайтів із впровадженням відповідних технологій. Розглянемо найпоширеніші технології, які є основою для сучасних веб-рішень.

Технологія ASP та її розширення ASP.NET (Active Server Pages) розроблена корпорацією Microsoft для створення динамічних веб-сторінок. Вона дозволяє веб-майстрам генерувати сторінки, які автоматично оновлюються, і відокремлювати процес створення дизайну від програмної логіки. ASP-сторінки включають HTML-текст із вбудованими сценаріями мов JavaScript або VBScript, які обробляються сервером і динамічно перетворюються у HTML-код для браузера. Подальший розвиток цієї

технології привів до створення JSP (Java Server Pages) — серверної технології Java для генерації динамічних сторінок, яка є крос-платформною альтернативою ASP від Microsoft.

Технологія JSP, створена на основі Java Servlet API, дозволяє розробляти серверні сторінки з інтеграцією до інших Java-компонентів. Вона також включає розширення JSF (Java Server Faces), яке спрощує розробку зручних користувацьких інтерфейсів для веб-додатків.

Серед ранніх технологій веб-розробки варто відзначити Common Gateway Interface (CGI). CGI-застосунки є консольними програмами, які генерують HTML-код, що передається браузеру. Вони дозволяють звертатися до серверних програм через URL, де вхідними даними можуть бути HTTP-запити або параметри пошукових систем. Ці програми, зазвичай написані на скриптових мовах, інтегруються із сервером, виконуючи обробку запитів.

Ще однією популярною технологією є PHP (Personal Home Pages) — некомерційна мова програмування для створення динамічних веб-сторінок. PHP відзначається високою практичністю та гнучкістю, дозволяючи інтеграцію з HTML, JavaScript, WML та XML. Завдяки використанню PHP, розробники мають змогу швидко і ефективно вирішувати задачі створення веб-додатків, забезпечуючи функціональність і зручність у роботі.

Для довготривалого зберігання інформації, яку обробляє веб-сайт, використовуються бази даних. Ці бази забезпечують зберігання даних у вигляді взаємопов’язаних таблиць. Найпоширенішими базами даних є SQL, MySQL та Oracle Database, які використовуються залежно від вимог до системи. MySQL, зокрема, є безкоштовною та компактною СКБД, що підтримує багатопоточність і забезпечує високу швидкість виконання команд. Її популярність обумовлена простотою у використанні та підтримкою великої кількості користувачів.

Oracle Database, натомість, має ширший спектр можливостей і застосовується у великих корпоративних системах. Ця система підтримує

операційні системи Windows, Unix, Linux та MacOS і є більш потужною альтернативою MySQL. Проте через високу вартість вона частіше використовується для великих підприємств, ніж для малих і середніх компаній.

Мова SQL (Structured Query Language) використовується для роботи з базами даних, забезпечуючи формування запитів, оновлення та управління реляційними базами даних. SQL також дозволяє визначати структуру бази, контролювати доступ до даних і забезпечувати їхній захист.

Розглянуті технології створення веб-сайтів і управління базами даних забезпечують розробникам інструменти для створення динамічних, функціональних і масштабованих рішень, які відповідають сучасним потребам користувачів. Кожна з технологій має свої особливості та переваги, що дозволяє підібрати оптимальний інструментарій для кожного проєкту.

2.3. Огляд контролерів нечіткої логіки

Одним із найважливіших напрямів використання теорії нечітких множин є контролери нечіткої логіки, які знайшли широке застосування у різноманітних системах керування, включаючи побутові прилади. Ці контролери не спираються на традиційні математичні моделі для опису системи. Натомість вони використовують знання експертів, які представляються у формі, максимально наближеній до вербальної мови, через лінгвістичні змінні та нечіткі множини [7].

Загальна структура нечіткого контролера (fuzzy-контролера) включає такі основні компоненти:

Блок фазифікації: здійснює перетворення точних значень, отриманих на виході об’єкта управління, у нечіткі величини. Ці нечіткі величини описуються через лінгвістичні змінні, які зберігаються у базі знань.

База знань: містить набір правил та лінгвістичних змінних, які використовуються для логічного висновку. Вона є ключовим елементом, який

формує основу функціонування системи.

Блок нечіткого логічного висновку: оперує нечіткими умовними правилами типу "ЯКЩО – ТО" (if – then). Цей блок аналізує вхідні дані, що мають нечіткий характер, та генерує керуючі впливи, які також є нечіткими [8].

Блок дефазифікації: перетворює нечіткі вихідні дані у чітке значення, яке подається на виконавчий пристрій для управління об’єктом.

Графічна структура таких контролерів зображена на рис. 2.1.

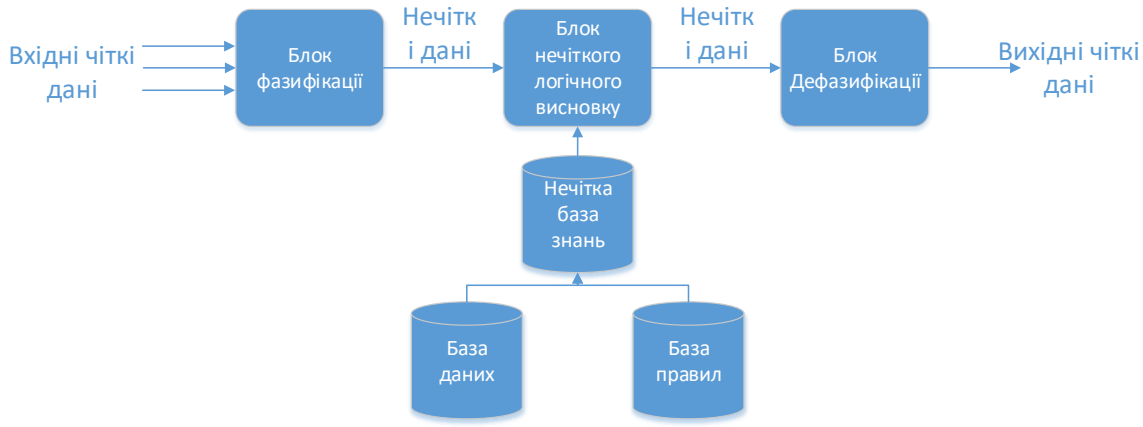

Рисунок 2.1 – Загальна структура fuzzi-контролера

Відповідно, процес роботи контролера нечіткої логіки визначається такими кроками:

- фазифікація (приведення чітких вхідних змінних до нечіткої форми, іншими словами, визначення рівня відповідності входів кожній із нечітких множин);
- логічний вивід (обчислюється значення істинності для передумови кожного правила);
- композиція (об’єднання нечітких виходів правил у загальне вихідне значення);

- дефазифікація (трансформація нечіткого виходу правил на чітке значення).

Зростання популярності систем штучного інтелекту, які інтегрують у свої алгоритми нечітку логіку, є свідченням значних переваг цього підходу. Нечітка логіка дозволяє створювати системи, здатні працювати з неповними або невизначеними даними, що робить її незамінною у складних динамічних середовищах. Ця галузь є актуальною не лише з наукової точки зору, але й з практичної, оскільки вона дозволяє створювати адаптивні, інтуїтивно зрозумілі та ефективні рішення для задач прийняття рішень, автоматизації, управління та аналізу великих даних [9].

Оскільки сучасні технології з кожним роком стають дедалі складнішими, використання нечіткої логіки сприяє розвитку нових підходів до обробки інформації. Її інтеграція з іншими технологіями, такими як штучний інтелект, машинне навчання та обчислювальний інтелект, відкриває нові горизонти для створення гібридних систем, які поєднують гнучкість і потужність аналітичних інструментів.

Перспективи розвитку нечіткої логіки охоплюють численні інноваційні підходи, серед яких:

- **Нечіткі штучні нейронні мережі**: інтеграція принципів нечіткої логіки у традиційні нейронні мережі дозволяє підвищити їхню стійкість до шуму у даних, а також розширити можливості аналізу в умовах невизначеності.
- **Адаптивне поповнення баз нечітких правил**: динамічне оновлення баз знань дозволяє системам самонавчатися, що є критично важливим для додатків, які працюють у змінному середовищі.
- **Підтримка нечітких запитів до баз даних**: цей підхід дає змогу формулювати складні запити, які враховують не лише точні збіги, а й наближені результати, що суттєво покращує якість пошуку інформації.
- **Побудова нечітких когнітивних карт**: використання таких карт

дозволяє візуалізувати складні взаємозв’язки між поняттями у системах аналізу та прийняття рішень.

- **Нечіткі графи, дерева прийняття рішень та мережі Петрі**: ці моделі ефективно використовуються для моделювання складних процесів, де звичайні алгоритми виявляються надто обмеженими.
- **Нечітка кластеризація**: цей метод дозволяє групувати об’єкти на основі нечітких критеріїв, що є корисним у задачах аналізу великих даних та сегментації.

Одним із ключових методів представлення знань у системах нечіткої логіки є **продукційні правила**, які нагадують стиль мислення людини. Ці правила описуються у формі: **"ЯКЩО <логічний вираз>, ТО <оператор>"**, де логічний вираз перевіряє стан системи, а оператор визначає дію, яку необхідно виконати у випадку виконання умови [11]. Такий підхід забезпечує гнучкість у розробці систем, дозволяючи враховувати складні взаємозв’язки між вхідними параметрами.

Продукційні правила є основою для створення зрозумілих і легко модифікованих моделей, що дозволяє розробникам швидко адаптувати системи до нових вимог або умов. Вони також сприяють зручній інтерпретації результатів, що робить системи нечіткої логіки доступними для широкого кола користувачів.

Системи нечіткої логіки мають низку переваг, які роблять їх особливо ефективними у задачах, де традиційні методи стикаються з обмеженнями. До ключових особливостей таких систем належать:

- **Нечітка специфікація параметрів**: здатність працювати із невизначеними або наближеними даними.
- **Нечіткий опис вхідних та вихідних змінних**: можливість оперувати лінгвістичними змінними, які ближчі до природної мови.
- **Нечіткий опис функціонування**: використання правил "ЯКЩО-ТО"

дозволяє легко формалізувати складні залежності.

Головним класом нечітких систем є **нечіткі системи управління** (контролери нечіткої логіки). Вони реалізуються через компоненти фазифікації, дефазифікації, базу знань та блок логічного висновку, що забезпечує їх універсальність та гнучкість у застосуванні [12].

Таким чином, розвиток і вдосконалення систем нечіткої логіки відкриває нові можливості для розв’язання складних завдань у різних сферах, включаючи інженерію, економіку, медицину, екологію та інші галузі, де критично важлива робота з нечіткою інформацією.

Системи нечіткої логіки широко застосовують **лінгвістичні змінні** для опису своїх параметрів, а також вхідних і вихідних величин. Лінгвістичні змінні є ключовим компонентом таких систем, оскільки вони дозволяють формалізувати складні концепції, експертні знання та якісну інформацію, яка часто не піддається точному числовому опису. Вони базуються на термінах природної мови, що робить їх зрозумілими та доступними для користувачів, навіть якщо вони не мають спеціалізованих технічних знань [13].

Наприклад, замість чітких числових значень, таких як "температура = 25°C", лінгвістична змінна може описувати це значення через терми, такі як "тепло" або "прохолодно". Це дозволяє враховувати суб’єктивність сприйняття та підвищує гнучкість системи. Завдяки такому підходу, системи нечіткої логіки можуть ефективно працювати в умовах невизначеності, неточності чи недостатності даних, що є важливим для розв’язання задач у складних динамічних середовищах.

Функції належності $\mu_{A_j}(x)$ однієї лінгвістичної змінної X визначаються в одному вимірювальному просторі X, і ставлять у відповідність кожному x дійсне число з відрізку $[0,1]$. Фактично це суб’єктивна міра того, наскільки елемент x відповідає поняттю, сутність якого формалізується нечіткою множиною[14].

Входи $x = (x_i | i = 1..n)$ та вихід y є чіткими контрольованими величинами. Кожен параметр $x_i, i = 1..n$ має нечіткий відповідник у вигляді лінгвістичної змінної $X = \{A_i | j = 1..m_j\}$.

Правила R_k, $k = 1..N$ перевіряють значення кожної лінгвістичної змінної. Тому максимально можлива кількість правил дорівнює $N_{max} = \prod_{i=1}^{n} m_j$. На практиці така кількість правил не завжди є необхідною . Реальна кількість правил $N \leq N_{max}$.

Вивід правила – лінгвістична змінна $Y = \{B_j | j = 1..m\}$, яка набуває значення одного із термів B_j.

Для узагальнення правил відбувається композиція їх нечітких виходів в одну нечітку множину з її подальшим перетворенням на чітке вихідне значення y [15].

Фазифікація полягає у перетворенні чітких вхідних величин $x = (x_1, x_2, \dots, x_n)$ до нечітких множин $A' = (A', A', \dots, A')$.

Під час фазифікації чіткого входу x_i визначають спутені його відповідності кожному лінгвістичному терму $A_{i,j}$ з фунціями належності $\mu_{A_{i,j}}, j = 1..m_i$. Ці ступені є значеннями функцій належності $\mu_{A_{i,j}}$ у точці $x = x_i$.

Нечіткі вхідні значення системи трансформуються на вихідні на основі правил нечіткої логіки[16].

Уявімо що система управління здійснює перетворення значень n вхідних лінгвістичних змінних $x = (x_i | i = 1..n)$ у вихідну лінгвістичну змінну $y = R(x)$ згідно з базою правил $R = \{R_k | k = 1..N\}$.

Правила R відображають закони управління у вигляді нечіткої імплікації $R = A \to B$, яку можна розглядати як нечітку множину на декартовому добутку носіїв вхідних та вихідних розмитих множин. Процес акумулювання нечіткого результату $B` \notin$ з нечітких вхідних множин $A`$ на основі знань $A \to B$ можна зобразити у такому вигляді:

$$B` = A` \cdot R = A` \cdot (A \to B) \quad (2.1)$$

де · – композиційне правило нечіткого виведення.

У практичному застосуванні нечітких систем логічного висновку найчастіше використовується **максимінна композиція**, яка дозволяє поєднати нечіткі множини для отримання вихідних значень. При цьому **нечітка імплікація** реалізується через знаходження мінімуму функцій належності, що відповідають нечітким правилам [17].

Для імітації роботи експертної системи за допомогою схеми імплікації використовується набір **нечітких продукційних правил**. Кожне правило є умовним оператором, який має структуру: **ЯКЩО логічний вираз, ТО оператор**, де логічний вираз є висловлюванням, сформованим на основі базових логічних операцій (наприклад, "І", "АБО", "НЕ") над нечіткими величинами.

Такі правила є основою для прийняття рішень у нечітких системах, оскільки вони дозволяють формалізувати експертні знання у вигляді зрозумілих умовних залежностей. Наприклад, у системі керування кліматом логічний вираз може виглядати так: **ЯКЩО температура "висока" І вологість "низька", ТО збільшити подачу холодного повітря**.

Цей підхід дозволяє враховувати взаємозв'язки між параметрами системи та приймати рішення, адаптовані до конкретних умов. Використання нечітких продукційних правил забезпечує простоту побудови та налаштування системи, що робить її ефективною для широкого спектра задач, де є невизначеність або неточність даних.

Правила можуть визначати відношення відповідності (Є) між вхідними лінгвістичними змінними X та їх нечіткими термами $\{A_{i,j} | i = 1,..,n; j = 1..m_j\}$. Загалом до правила можуть входити усі можливі комбінації лінгвістичних термів для усіх вхідних змінних, об’єднаних логічними

операціями. Для визначення нечіткої кон’юнкції можна використати знаходження мінімуму, а для нечіткої диз'юнкції – знаходження максимуму двох функцій належності [18].

Дефазифікація є важливим етапом у роботі систем нечіткої логіки, який застосовується для перетворення нечітких множин, отриманих на виході системи, у чіткі числові значення. Цей процес особливо корисний, коли результати роботи системи потрібно надати у формі, придатній для використання виконавчими пристроями або для прийняття конкретних рішень.

Існує кілька методів дефазифікації, кожен з яких має свої переваги і застосовується залежно від конкретної задачі[19,20].

Метод середнього центру або центроїдний метод полягає у знаходженні центру ваги (центроїду) який і обирається за вихідний результат:

$$Y = \frac{\sum_{J=1}^{m} y_j B`(y_j)}{\sum_{J=1}^{m} B`(y_j)} \tag{2.2}$$

де $B`(y_j)$ – агреговане значення виходів системи після етапу логічного виведення

Y – результуюче значення (результат дефазифікації)

Для моделі *Такагі–Суджено* вихідні множини правил задаються у вигляді сінглетонів з функціями належності

$$\mu_{o_k}(y_k) = \begin{cases} 1, y_k = o_k \\ 0, y_k <> o_k \end{cases} \tag{2.3}$$

де o_k– вихідне значення k -го правила.

Тоді результуюче чітке вихідне значення системи прийняття рішень обчислюється зважуванням значень активованих правил:

$$y = \frac{\sum_{k=1}^{N} a_k y_k}{\sum_{k=1}^{N} a_k} \quad (2.4)$$

У системах керування отримане чітке вихідне значення використовується у контурі зворотного зв’язку для вироблення керуючих дій.

2.4. Огляд прикладу нечіткого виведення

Для задачі прогнозування ризику аварійності автомобіля використовується база нечітких правил, яка враховує залежності між швидкістю руху автомобіля та погодними умовами. Система базується на введених лінгвістичних змінних:

- **Швидкість руху**: висока, середня, низька;
- **Погодні умови**: сприятливі, несприятливі;
- **Ризик аварійності**: високий, середній, низький.

База правил

Для моделювання взаємозв'язків визначено такі нечіткі продукційні правила:

Для моделювання взаємозв'язків визначено такі нечіткі продукційні правила:

- R1: якщо швидкість висока і погодні умови несприятливі, то ризик високий;
- R2: якщо швидкість середня і погодні умови сприятливі, то ризик середній;
- R3: якщо швидкість низька і погодні умови сприятливі, то ризик низький.

Ці правила є основою для нечіткого логічного висновку.

Нечіткі множини для термів

Терми входів і виходів визначені такими нечіткими множинами:

Змінні та результуючі терми описані такими нечіткими множинами:

- Висока швидкість: {30/0.1; 50/0.5; 70/0.9; 90/1};
- Середня швидкість: {30/0.4; 50/0.8; 70/0.5; 90/0.2};
- Низька швидкість: {30/1; 50/0.5; 70/0.2; 90/0.1};
- Сприятливі погодні умови: {0/0.2; 1/0.5; 2/0.8; 3/1};
- Несприятливі погодні умови: {0/1; 1/0.8; 2/0.4; 3/0.1};
- Високий ризик: {0/0.1; 0.5/0.5; 1/1};
- Середній ризик: {0/0.5; 0.5/1; 1/0.5};
- Низький ризик: {0/1; 0.5/0.8; 1/0.2}.

Завдання: прогнозування ризику

Необхідно оцінити ризик аварійності при умовах середньої швидкості та частково сприятливих погодних умов. Вхідні дані описані нечіткими множинами:

- Швидкість: {30/0.4, 50/0.8, 70/0.5, 90/0.2};
- Погодні умови: {0/0.3, 1/0.7, 2/0.6, 3/0.4}.

Етапи нечіткого логічного виведення

1. **Обчислення рівнів істинності правил**:

   Для правила R1: обчислюється мінімум між максимумами значень для швидкості та погодних умов.

   Результат: min[max(0.4, 0.8, 0.5, 0.2), max(0.3, 0.7, 0.6, 0.4)] = min[0.8, 0.7] = 0.7.

   Для правила R2: обчислюється аналогічно. Результат: min[max(0.4, 0.8, 0.5, 0.2), max(0.2, 0.5, 0.8, 1)] = min[0.8, 0.8] = 0.8.

   Для правила R3: аналогічно. Результат: min[max(1, 0.5, 0.2, 0.1), max(0.3, 0.7, 0.6, 0.4)] = min[1, 0.7] = 0.7.

2. **Обчислення виходів правил**:

   Для кожного правила вихідна множина коригується відповідно до обчислених рівнів істинності.

   Для R1: {0/0.1, 0.5/0.7, 1/0.7}.

   Для R2: {0/0.8, 0.5/0.8, 1/0.8}.

Для R3: {0/1, 0.5/0.7, 1/0.7}.

3. **Агрегування виходів**: Вихідна множина об'єднується, беручи максимум значень для кожного рівня.Результат: {0/1, 0.5/0.8, 1/0.8}.
4. **Дефазифікація виходу**: Обчислюється чітке значення ризику як зважене середнє:y = (0 * 1 + 0.5 * 0.8 + 1 * 0.8) / (1 + 0.8 + 0.8) = 1.2 / 2.6 ≈ 0.46.

Результат

Отримане чітке значення ризику аварійності становить приблизно 46%, що відповідає середньому рівню ризику.

Цей підхід демонструє можливості систем нечіткої логіки для моделювання залежностей між змінними в умовах невизначеності, що робить їх корисними для задач адаптивного прийняття рішень.

2.5. Огляд можливостей використання контролерів нечіткої логіки для задач управління

Прийняття рішень у проблемно-орієнтованих інформаційних системах та системах керування часто відбувається в умовах **апріорної невизначеності**. Ця невизначеність може бути спричинена низкою факторів, серед яких: неточність або неповнота вхідних даних, стохастичний характер зовнішніх впливів, відсутність адекватних математичних моделей, нечіткість визначення мети або людський фактор. У таких умовах прийняття рішень стає складним і пов’язане зі значними ризиками. Неефективні рішення можуть призводити до негативних економічних, технічних чи соціальних наслідків.

Для компенсації цих невизначеностей дедалі частіше застосовуються методи штучного інтелекту, зокрема на основі **нечіткої логіки**. Методи, побудовані на теорії нечітких множин, дозволяють ефективно працювати в умовах невизначеності, використовуючи лінгвістичні величини та висловлювання для опису стратегій прийняття рішень [21].

Методи нечітких множин мають низку унікальних особливостей, які роблять їх незамінними у ситуаціях, де точні математичні моделі неможливо побудувати. Зокрема, теорія нечітких множин дозволяє використовувати неточні, суб'єктивні експертні знання про предметну область без необхідності їхньої формалізації у вигляді традиційних математичних моделей [22]. Це забезпечує:

1. Узгодження суперечливих критеріїв прийняття рішень.
2. Створення логічних регуляторів для складних систем.
3. Можливість лінгвістичного опису складних процесів.
4. Прогнозування поведінки системи та формування альтернативних дій.
5. Формальний опис нечітких правил прийняття рішень.

Завдяки цим властивостям нечіткі множини широко застосовуються у проектуванні інтерфейсів для людино-машинних систем. На основі нечітких множин будуються системи керування, підтримки прийняття рішень, апроксимації, ідентифікації, розпізнавання образів, оптимізації тощо. Нечітка логіка знайшла застосування у таких сферах, як побутова електроніка, діагностика, експертні системи та інші галузі [23].

Експертні системи, які базуються на нечіткій логіці, активно використовуються для підтримки прийняття рішень у військовій справі, медицині, економіці, а також у задачах бізнес-прогнозування, оцінювання ризиків і прибутковості інвестиційних проєктів. З їхньою допомогою моделюються глобальні політичні рішення, аналізуються кризові ситуації та створюються сценарії їх подолання [7, 8]. Одним із ключових застосувань теорії нечітких множин є контролери нечіткої логіки, які використовуються у системах керування побутовими приладами та іншими автоматизованими системами.

Основні переваги застосування нечіткої логіки для автоматизації порівняно з традиційними методами автоматичного управління включають:

- **Підвищення швидкодії**: Нечіткі контролери дозволяють значно скоротити час реагування системи на зовнішні зміни.
- **Гнучкість у розробці систем**: Можливість створення керуючих алгоритмів для об'єктів, які важко формалізувати за допомогою класичних математичних підходів.
- **Адаптивність**: Легкість синтезу адаптивних регуляторів, що базуються на класичних ПІД-регуляторах, але враховують змінні умови.
- **Зниження похибок**: Підвищення точності фільтрації випадкових збурень під час обробки даних від сенсорів.
- **Зменшення ризику помилкових рішень**: Зниження ймовірності помилок у роботі керуючих алгоритмів, що сприяє збільшенню терміну служби обладнання.

Традиційні системи автоматизованого управління базуються на лінійних моделях, які будуються за певними критеріями оптимальності. Хоча ці підходи є ефективними для простих систем, вони часто стикаються з обмеженнями у випадках складних, нелінійних або динамічних об'єктів. Методи спрощення та лінеаризації, які використовуються для таких об'єктів, часто не забезпечують належної якості управління.

Зі збільшенням складності об'єктів управління і виконуваних ними функцій стає дедалі важче застосовувати класичні методи. У таких умовах нечітка логіка є перспективним і ефективним інструментом, що дозволяє враховувати невизначеність і забезпечувати високу якість автоматизованого управління реальними технологічними процесами.

РОЗДІЛ 3. СИСТЕМНИЙ АНАЛІЗ

3.1. Побудова дерева цілей

З метою проведення грунтовного аналізу системи було розроблено дерево цілей (рис 3.1).

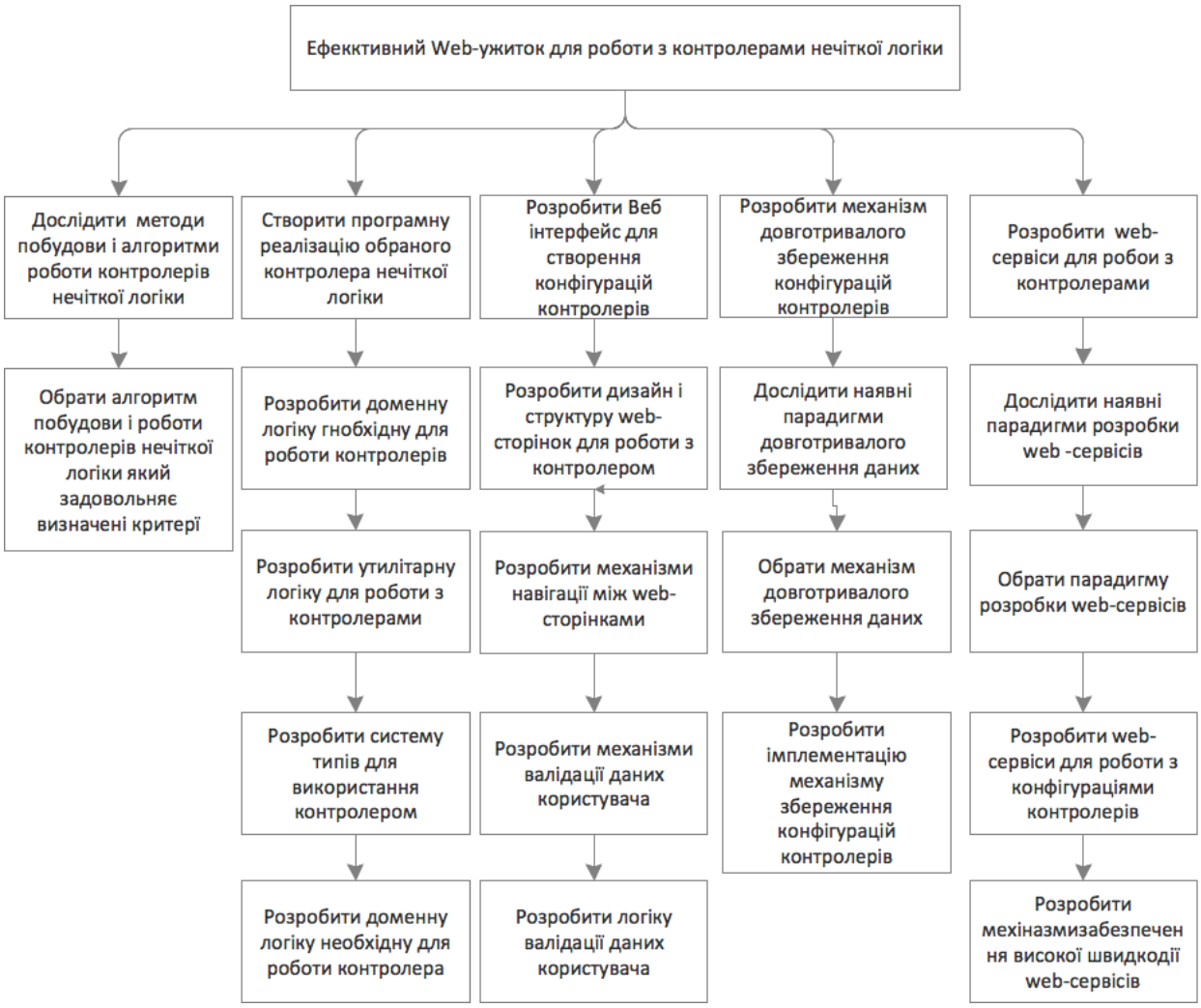

Рисунок 3.1 – Дерево цілей для доменної області “Web-ужиток для роботи з контролерами нечіткої логіки”

3.2. Аналіз методів і алгоритмів побудови контролерів нечіткої логіки

Алгоритм Мамдані. Алгоритм Мамдані складається з декількох послідовних етапів, при цьому кожен наступний етап отримує на вхід результати, отримані на попередньому кроці.

У системах типу Мамдані база знань будується з нечітких висловлювань виду «β належить α» за допомогою логічних зв'язок, таких як «І» та «ЯКЩО-ТО». Наприклад:

ЯКЩО x є високим **І** y є середнім, **ТО** z є високим.

Етапи нечіткого виводу реалізуються наступним чином:

1. Фаззифікація: знаходження ступеня істинності для передумов кожного з правил: $A_1(x0)$, $A_2(x0)$, $B_1(y0)$, $B_2(y0)$
2. Нечіткий логічний висновок: знаходження рівнів відсікання для передумов кожного з правил використовуючи операції мінімуму:

$$\begin{aligned} \alpha_1 &= A_1(x_0) \wedge B_1(y_0), \\ \alpha_2 &= A_2(x_0) \wedge B_2(y_0), \end{aligned} \qquad (3.2.1)$$

де ∧ - операція логічного мінімуму

Далі знаходяться усічені функції належності

$$\begin{aligned} C_1'(z) &= (\alpha_1 \wedge C_1(z)), \\ C_2'(z) &= (\alpha_2 \wedge C_2(z)) \end{aligned} \qquad (3.2.2)$$

3. Композиція з використанням операції максимуму. Виконання обєднання усічених функцій, з метою отримання результуючої нечіткої множини для змінної виходу з функцією приналежності:

$$\mu_{\sum}(z) = C(z) = C_1'(z) \vee C_2'(z) = \\ = (\alpha_1 \wedge C_1(z)) \vee (\alpha_2 \wedge C_2(z)) \quad (3.2.3)$$

4. Приведення до чіткості (дефазифікація) виконується з використанням конкретних методів в залежності від мети розробки контролера нечіткої логіки.

Алгоритм Tsukamoto. Для алгоритму Tsukamoro вихідні дані і база знань представляються в тому ж вигляді як для алгоритму Mamdani, але функції $C_1(z)$, $C_2(z)$ є монотонними. Етапи нечіткого логічного виведення:

1. Фазифікація: знаходження ступеня істинності для передумов кожного з правил: $A_1(x0)$, $A_2(x0)$, $B_1(y0)$, $B_2(y0)$
2. Нечіткий логічний висновок: знаходження рівнів відсікання передумов кожного правила з використанням операцій мінімуму:

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0), \\ \alpha_2 = A_2(x_0) \wedge B_2(y_0), \quad (3.2.4)$$

де ∧ - операція логічного мінімуму

В ході наступного кроку виконується пошук чітких значень z_1 і z_2 з рівнянь

$$\alpha_1 = C_1(z), \\ \alpha_2 = C_2(z) \quad (3.2.5)$$

3. Далі виконується пошук чіткого значення z_0 змінної виходу, як середньозважене z_1 і z_2:

$$z_0 = \frac{\alpha_1 z_1 + \alpha_2 z_2}{\alpha_1 + \alpha_2} \quad (3.2.6)$$

Алгоритм Sugeno. В алгоритмі Sugeno побуова бази правил виконується наступним чином:

Якщо x є A1 і y є B1, то z1=a1x+b1y

Якщо x є A2 і y є B2, то z2=a2x+b2y

Алгоритм передбачає наступні етапи нечіткого логічного висновку:

1. Фазифікація : знаходження ступеня істинності для передумов кожного з правил: $A_1(x0)$, $A_2(x0)$, $B_1(y0)$, $B_2(y0)$
2. Нечіткий логічний висновок: пошук рівнів відсікання для передумов кожного з правилз вокристовуючи операції мінімуму:

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0), \quad (3.2.7)$$
$$\alpha_2 = A_2(x_0) \wedge B_2(y_0),$$

де ∧ - операція логічного мінімуму

Знаходження індивідуальніих виходів правил:

$$z^*_1 = a_1 x + b_1 y \quad (3.2.8)$$
$$z^*_2 = a_2 x + b_2 y$$

3. Визначення чіткеого значення змінної виходу:

$$z_0 = \frac{\alpha_1 z^*_1 + \alpha_2 z^*_2}{\alpha_1 + \alpha_2} \quad (3.2.9)$$

Алгоритм Larsen. Представлення бази знань співпадає з алгоритмом Mamdani.

Етапи логічного виведення включають в себе:

1. Фазифікація: знаходження ступенів істинності для передумов кожного з продукційних правил: $A_1(x0)$, $A_2(x0)$, $B_1(y0)$, $B_2(y0)$
2. Нечіткий логічний висновок: знаходження рівнів відсікання для передумов кожного з правил використовуючи операції мінімуму:

$$\begin{aligned} \alpha_1 &= A_1(x_0) \wedge B_1(y_0), \\ \alpha_2 &= A_2(x_0) \wedge B_2(y_0), \end{aligned} \tag{3.2.10}$$

де ∧ - операція логічного мінімуму

В ході виконання алгоритму Larsen нечітка підмножина для змінної виходу кожного правила знаходиться вокористовуючи оператор мультиплікації.

$$\begin{aligned} C_1'(z) &= (\alpha_1 C_1(z)), \\ C_2'(z) &= (\alpha_2 C_2(z)) \end{aligned} \tag{3.2.11}$$

3. Композиція з використання операції максимуму. Здійснюється обєднання знайдених часткових нечітких підмножин. Результуюча нечтка підмножина для змінної виходу з функцією належності:

$$\begin{aligned} \mu_{\sum}(z) &= C(z) = C_1'(z) \vee C_2'(z) = \\ &= (\alpha_1 C_1(z)) \vee (\alpha_2 C_2(z)) \end{aligned} \tag{3.2.12}$$

Для загального випадку випадку:

$$\mu_{\sum}(z) = \bigvee_{i=1}^{n} (\alpha_i C_i(z)) \tag{3.2.13}$$

5. Дефазифікація виконується з використанням різних методів в залежності від мети розробки контролера нечіткої логіки.

3.3. Аналіз сучасних інструментів розробки контролерів нечіткої логіки

На сучасному етапі розвитку технологій можна виділити три основні типи інструментів, які використовуються для розробки контролерів нечіткої логіки:

1. Бібліотеки для інтеграції контролерів нечіткої логіки у програмні застосунки

   Прикладом є Fuzzy Logic Tools та подібні рішення, які дозволяють розробникам інтегрувати нечіткі контролери безпосередньо у свої програми.

2. Інтегровані середовища розробки

   Ці середовища, такі як Xfuzzy 3, fuzzyTECH, Mathlab Fuzzy Logic ToolBox, забезпечують комплексні інструменти для розробки, тестування та реалізації нечітких систем.

3. Демонстраційні веб-сайти

   До таких належать Fuzzy Calculator, Fuzzy Pendulum Demo тощо, які орієнтовані на візуалізацію і демонстрацію роботи нечітких алгоритмів.

Fuzzy Logic Tools (FLT)

Fuzzy Logic Tools (FLT) — це фреймворк на мові C++, розроблений для створення, аналізу та інтеграції багатовходових і багатовихідних (MIMO) нечітких контролерів. Цей інструмент підтримує алгоритми Такагі-Сугено і не має обмежень щодо розмірів вхідних та вихідних векторів.

Основні особливості FLT:

- Гнучкість у виборі функцій належності: підтримуються різні типи функцій належності, які можна змішувати в одній системі.
- Можливість інтеграції: забезпечується взаємодія з іншими системами через Mathlab API.
- Функції обробки текстових файлів: читання і запис нечітких систем із текстових форматів.

- Модульна архітектура: підтримка створення субсистем нечіткої логіки, що спрощує побудову великих систем.

FLT ідеально підходить для розробки систем на базі моделі Такагі-Сугено, оскільки дозволяє адаптувати контролери до конкретних потреб користувача.

Xfuzzy 3

Xfuzzy 3 є інтегрованим середовищем розробки для систем нечіткої логіки. Воно включає в себе набір інструментів, які охоплюють весь цикл розробки нечітких систем — від початкового опису моделі до її кінцевої реалізації.

Основні особливості Xfuzzy 3:

- Гнучкість: середовище дозволяє користувачеві розширювати набір доступних функцій, що робить його підходящим для складних і нестандартних задач.
- Мультиплатформність: Xfuzzy 3 написане мовою Java і може працювати на будь-якій платформі з встановленим Java Runtime Environment (JRE).
- Інструменти автоматизації: підтримка автоматизованого проектування і налаштування нечітких систем.
- Модульність: середовище надає можливість інтеграції окремих компонентів у більші системи.

На рис. 3.2 зображено функціональну структуру Xfuzzy 3, яка демонструє його основні компоненти, включаючи інтерфейси для проектування, симуляції та налаштування систем.

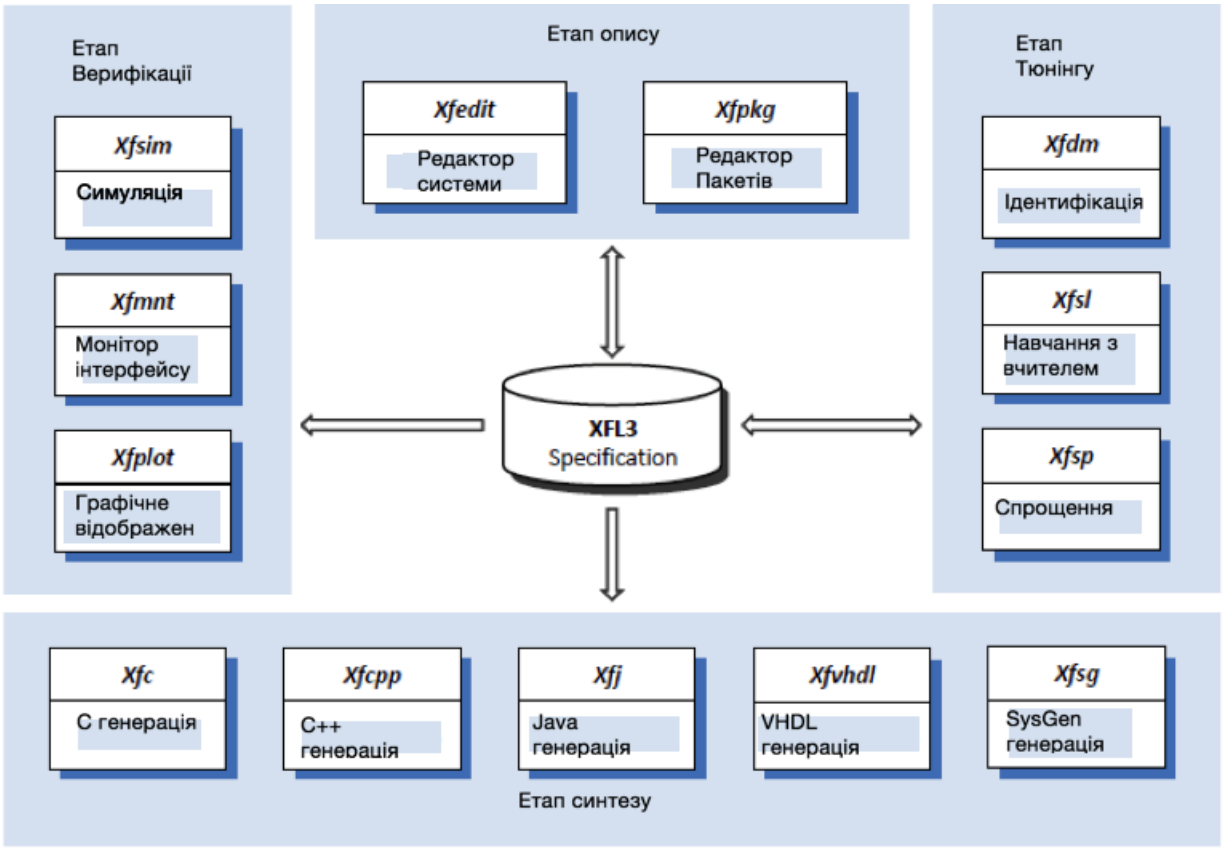

Рисунок 3.2 – Функціональний дизайн середовища Xfuzzy3

Процес розробки нечітких систем поділяється на кілька основних етапів, кожен з яких включає специфічні інструменти та завдання:

1. **Описовий етап**

   На цьому етапі використовуються графічні інструменти для визначення структури нечіткої системи, включаючи її вхідні та вихідні змінні, функції належності та правила логічного висновку. Це базовий етап, який формує основу для подальшої роботи.

2. **Етап перевірки**

   Цей етап включає інструменти для моделювання, моніторингу та графічного представлення поведінки системи. Він дозволяє оцінити, як система реагує на різні вхідні значення, та перевірити її відповідність очікуванням.

3. **Етап тюнінгу**

   Завдання цього етапу полягає у вдосконаленні системи шляхом ідентифікації, вивчення та спрощення алгоритмів. Це дозволяє оптимізувати роботу системи, забезпечуючи її ефективність та надійність.

4. **Етап синтезу**

   На заключному етапі інструменти розробки генерують код високого рівня на основі специфікації створеної системи. Цей код може бути використаний для інтеграції нечіткої логіки у програмні або апаратні рішення.

Mathlab Fuzzy Logic Toolbox

Fuzzy Logic Toolbox, доступний у середовищі MATLAB, забезпечує потужний набір інструментів для розробки та аналізу класичних нечітких систем. Toolbox дозволяє виконувати такі основні завдання:

- **Розробка та аналіз систем нечіткого виводу**: моделювання систем, що використовують нечітку логіку для прийняття рішень.
- **Розробка адаптивних систем нейронечіткого логічного висновку**: поєднання нечіткої логіки та штучних нейронних мереж для створення гібридних рішень.
- **Інтеграція з Simulink**: Toolbox надає блоки нечітких контролерів, які можна використовувати у Simulink для моделювання та імітації. Крім того, з Simulink можна генерувати **C-код** для використання у вбудованих додатках.

Fuzzy Logic Toolbox складається з кількох компонентів, які охоплюють усі аспекти розробки нечітких систем:

1. **FIS Editor**

   Забезпечує огляд загальної інформації про нечітку систему виводу (Fuzzy Inference System, FIS). Це центральний компонент, який дозволяє візуалізувати структуру системи.

2. **Редактор функцій належності**

   Надає можливість створення, редагування та перегляду функцій належності, пов'язаних із вхідними та вихідними змінними. Це важливий інструмент для налаштування параметрів системи.

3. **Редактор правил**

Дозволяє переглядати та редагувати нечіткі правила у трьох форматах:

- Повний англійський-подібний синтаксис.
- Скорочене символічне позначення.
- Індексоване позначення.

4. **Переглядач правил**

Сприяє детальному аналізу поведінки системи на основі заданих правил. Дозволяє діагностувати помилки або оцінити вплив змін у вхідних даних.

5. **Переглядач поверхонь**

Генерує тривимірну (3D) поверхню, яка відображає залежність між двома вхідними змінними та вихідним значенням системи. Це зручний інструмент для візуалізації складних взаємозв’язків у системі.

Fuzzy Logic Toolbox також виділяється своєю зручністю для візуального аналізу й оптимізації розроблених систем. За допомогою вбудованих графічних інструментів користувачі можуть легко налаштовувати функції належності, коригувати правила та переглядати вплив змін вхідних параметрів на вихід. Зокрема, Переглядач поверхонь (Surface Viewer) дозволяє створювати тривимірні графіки, які відображають залежність вихідних значень від двох вхідних змінних, забезпечуючи інтуїтивне розуміння роботи системи. Дизайн і основні компоненти Toolbox наочно представлені на рис. 3.3, що ілюструє його функціональність та взаємозв’язок між різними етапами проектування нечітких систем.

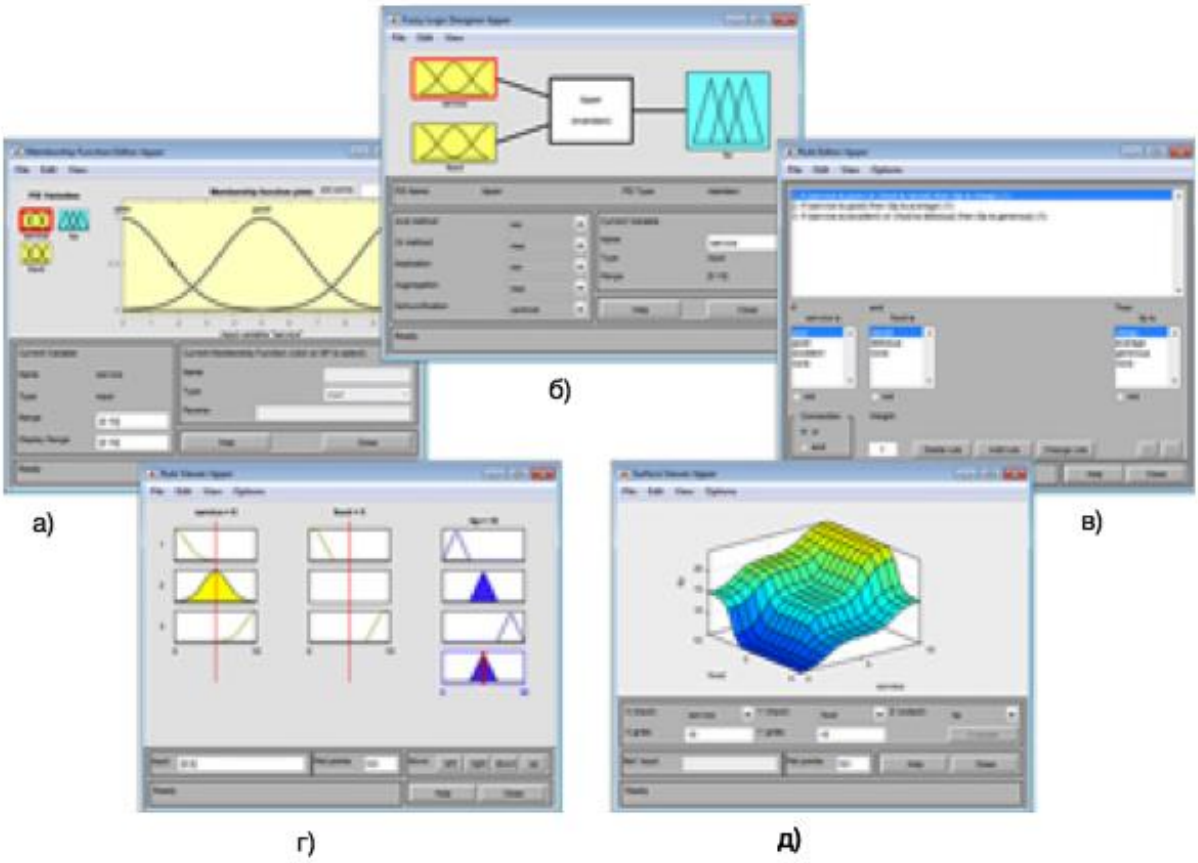

Рисунок 3.3 – Дизайн Fuzzy Logic Toolbox. а). Редактор Функцій Належності , б). FIS Едітор, в). Редактор Правил г). Переглядач Правил, д). Переглядач поверхні

3.4. Аналіз засобів розробки користувацьких інтерфейсів для створення конфігурацій контролерів нечіткої логіки

На сучасному етапі розвитку інтернет-технологій існує велика кількість підходів до розробки користувацьких інтерфейсів, орієнтованих на різні типи клієнтських платформ та сценарії використання. Основні підходи включають:

1. **Орієнтовані на обробку запитів**

   Цей підхід використовується переважно для високонавантажених інтернет-застосунків, де критичними є швидкість роботи та можливість масштабування. Приклади платформ на Java: **Struts 2**, **Spring MVC**, **Rails**, **Stripes**.

2. **Орієнтовані на роботу з компонентами**

Цей підхід зазвичай застосовується в інтранет-середовищах або захищених мережах із невеликою кількістю користувачів. Платформи, такі як **JSF**, **Tapestry**, **GWT**, забезпечують інтегровану підтримку компонентів та функціоналу.

3. **Багаті інтернет-застосунки (RIA)**

Підходи, такі як **Flash** та **Flex**, були популярними у минулому, але сьогодні вони поступаються місцем технологіям, базованим на **CSS3**, **HTML5** та інших стандартах. Новіші підходи є простішими у реалізації та краще інтегруються з сучасними браузерами.

Аналіз підходів

Популярні реалізації цих підходів на платформі Java були проаналізовані за низкою критеріїв, включаючи:

- Продуктивність розробників.
- Час на вивчення та сприйнятливість.
- Якість і стабільність проекту.
- Підтримка шаблонізації, AJAX, компонентів.
- Масштабованість і підтримка тестування.
- Локалізація та валідація даних.
- Підтримка роботи з веб-сервісами.
- Можливості роботи на мобільних платформах.
- Рівень ризиків і наявність аддонів.

Ці критерії забезпечують комплексну оцінку кожного підходу, допомагаючи обрати найкращий варіант для конкретних задач. Детальні результати аналізу представлені у таблиці 3.1, яка порівнює підходи за зазначеними характеристиками, дозволяючи зробити обґрунтований вибір для проекту.

Таблиця 3.1. Аналіз засобів розробки користувацьких інтерфейсів

Критерій	**Struts 2**	**Spring MVC**	**Wicket**	**JSF 2**	**Tapestry**	**Stripes**	**GWT**	**Grails**	**Rails**	**Flex**	**Vaadin**	**Lift**	**Play**
Продуктивність розробників	0.5	1.0	0.5	0.5	1.0	0.5	0.9	1.0	1.0	0.0	1.0	0.5	1.0
Сприйнятливість	0.5	0.9	1.0	0.5	0.5	1.0	0.9	1.0	1.0	1.0	1.0	1.0	1.0
Час Вивчення	1.0	0.9	0.5	0.5	0.5	1.0	0.9	1.0	1.0	1.0	1.0	0.5	1.0
Якість	0.5	1.0	1.0	1.0	1.0	0.5	0.9	1.0	1.0	0.5	1.0	1.0	1.0
Підтримка шаблонізації	1.0	0.9	1.0	0.5	1.0	1.0	0.4	1.0	1.0	0.5	0.5	0.5	0.5
Компоненти	0.0	0.9	1.0	1.0	1.0	0.0	0.4	0.5	0.5	1.0	1.0	0.0	0.0
Ajax	0.5	1.0	0.5	0.5	0.5	0.5	0.9	0.5	0.5	0.5	1.0	1.0	0.5
Наявність аддонів	0.5	0.9	1.0	1.0	0.5	0.0	0.9	1.0	1.0	1.0	1.0	0.5	1.0
Масштабованість	1.0	0.9	0.5	0.5	0.5	1.0	0.9	0.5	0.5	0.5	0.5	1.0	1.0
Підтримка тестування	1.0	1.0	0.5	0.5	1.0	1.0	0.4	1.0	1.0	0.0	0.5	0.5	1.0
Підтримка Локалізації	1.0	0.9	1.0	0.5	1.0	1.0	0.9	1.0	0.5	0.5	1.0	1.0	1.0
Підтримка Валідації	1.0	0.9	1.0	0.5	1.0	1.0	0.9	1.0	1.0	1.0	1.0	0.5	0.5
Підтримка кількох мов	0.5	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0

Підтримка роботи з web-сервісами	0.5	0.9	0.5	0.0	0.5	0.5	0.4	1.0	1.0	0.5	0.5	0.5	0.5
Можливості роботи на мобільних платформах	1.0	0.9	1.0	1.0	1.0	1.0	0.9	1.0	1.0	0.5	1.0	1.0	1.0
Агрегований процент ризиків	1.0	1.0	1.0	1.0	1.0	1.0	0.9	1.0	1.0	1.0	0.5	0.5	0.5
Загальна оцінка	11.5	15.0	13.0	10.5	13.0	12.0	11.5	14.5	13.0	9.5	13.5	10.0	12.5

Агреговані результати аналізу показано на рис.3.4.

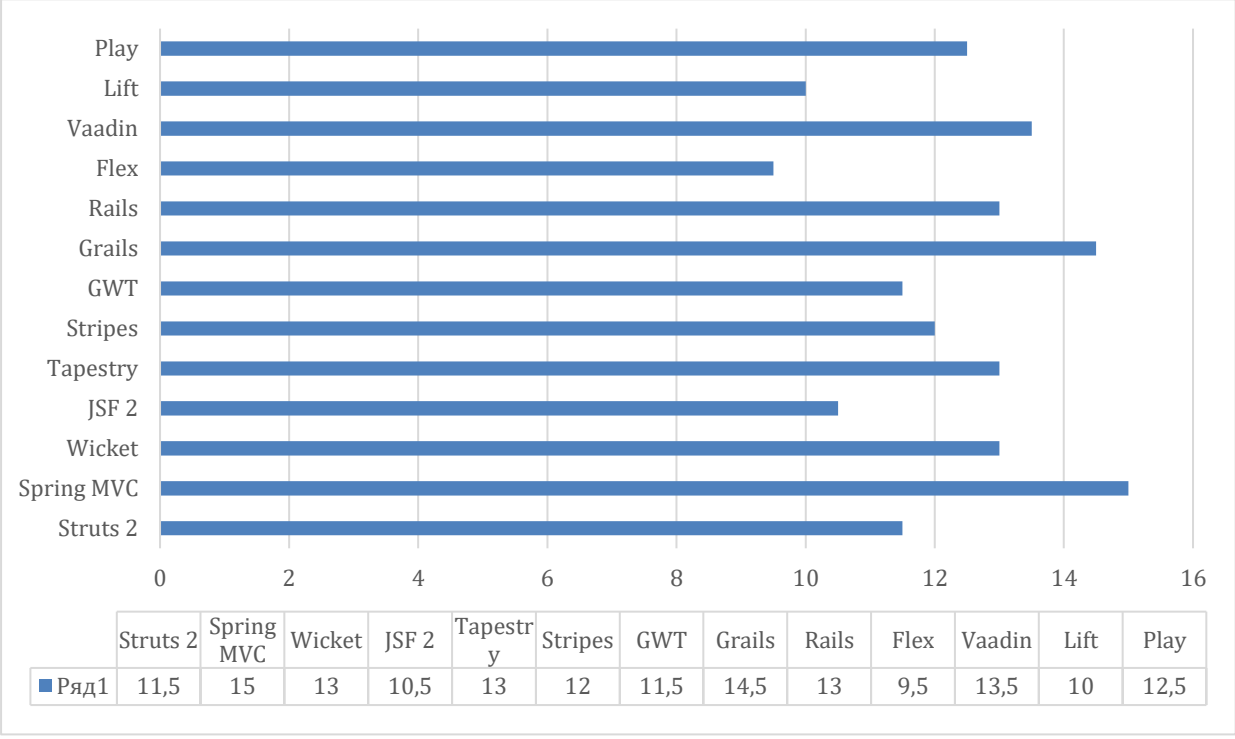

Рисунок3.4 – Агреговані результати аналізу засобів розробки користувацьких інтерфейсів

Як результат проведеного аналізу, використання **Spring MVC** визнано доцільним вибором для розробки веб-інтерфейсу, призначеного для

створення та конфігурації контролерів нечіткої логіки. Цей фреймворк забезпечує високу продуктивність розробки, масштабованість, а також підтримку сучасних технологій, таких як AJAX, шаблонізація, та інтеграція з RESTful веб-сервісами. Крім того, Spring MVC має добре організовану архітектуру, що дозволяє легко адаптувати систему до змін вимог, та забезпечує широкий спектр інструментів для перевірки даних і локалізації, що є важливим для розробки комплексних систем.

Таким чином, вибір **Spring MVC** є оптимальним рішенням для реалізації веб-застосунку, який поєднує в собі інтуїтивний користувацький інтерфейс та високоефективну платформу для керування нечіткими логічними системами.

3.5. Аналіз мехінізмів довготривалого зберігання конфігурацій контролерів нечіткої логіки

У веб-застосунках використовується різноманіття механізмів збереження даних, які обираються залежно від функціональних вимог, обсягу даних, частоти доступу та рівня безпеки. Основні механізми збереження даних включають:

1. **Збереження даних в оперативній пам'яті у вигляді об’єктів** Цей підхід забезпечує найвищу швидкість доступу до даних, оскільки вони зберігаються безпосередньо в оперативній пам’яті. Він підходить для тимчасових даних, які часто змінюються, але втрачаються при перезапуску сервера.

2. **Збереження файлів у сесії користувача** Використовується для зберігання тимчасових даних, специфічних для певного користувача. Даний метод дозволяє уникнути дублювання даних та зменшує навантаження на сервер, але обмежений у масштабованості при великій кількості користувачів.

3. **Збереження даних у текстових файлах певного формату**
Зберігання у файловій системі сервера є простим у реалізації і підходить для невеликих обсягів даних. Проте доступ до таких даних є менш ефективним у порівнянні з використанням баз даних, і забезпечення їхньої безпеки може бути складним.

4. **Збереження даних у файлах Cookies**
Цей механізм дозволяє зберігати невеликі обсяги даних безпосередньо у браузері користувача, що зменшує навантаження на сервер. Cookies ідеально підходять для зберігання налаштувань або ідентифікаторів сесії, але мають обмеження щодо обсягу даних і можуть бути видалені користувачем.

5. **Збереження даних у системах керування базами даних (СКБД)**
Використовується для зберігання великих обсягів структурованих даних з високими вимогами до цілісності, надійності та безпеки. СКБД забезпечують ефективність доступу, підтримку транзакцій і масштабованість, що робить їх ідеальним вибором для складних систем.

Порівняльна характеристика цих механізмів збереження даних, яка враховує швидкість доступу, складність реалізації, обсяги даних, що зберігаються, і рівень безпеки, наведена у таблиці 3.2. Ця таблиця дозволяє визначити оптимальний механізм збереження даних для конкретного веб-застосунку, враховуючи його вимоги та особливості реалізації.

Таблиця 3.2. Порівняння механізмів збереження даних у web-ужитках

Зберігання даних	**Оперативна пам'ять**	**Сесія користувача**	**Файлова система**	**Файли Cookies**	**СКБД**
Надійність	Мала надійність. Можливість втрати даних через	Мала надійність. Можливість втрати даних	Висока надійність	Обмежена надійність. Файли можуть бути випадково видалені	Висока надійність

Зберігання даних	**Оперативна пам'ять**	**Сесія користувача**	**Файлова система**	**Файли Cookies**	**СКБД**
	помилку сервера	через помилку сервера		користувачем або через помилки системи	
Простота	Дуже проста реалізація	Дуже проста реалізація	Потребує розробки спеціального механізму збереження	Проста у використанні	Потребує інтеграції з сервером бази даних
Період зберігання даних	Діє тільки під час активності програми	Тривалість користувацької сесії	Без обмежень	Без обмежень	Без обмежень
Зручність	Незручний підхід	Незручний підхід	Зручний підхід	Незручний підхід	Зручний підхід
Доступність з різних пристроїв	Так	Ні	Так	Ні	Так
Можливість зберігати складну інформацію	Ні	Ні	Так	Ні	Так

В результаті аналізу даних, наведених у таблиці 3.2, можна зробити висновок, що найбільш доцільним рішенням для реалізації механізму збереження даних у рамках цієї магістерської роботи є використання **систем керування базами даних (СКБД)**. Цей вибір зумовлений їхньою здатністю забезпечувати ефективне збереження великих обсягів структурованих даних, підтримку транзакцій, масштабованість, а також високий рівень безпеки.

Наступним етапом є вибір конкретної реалізації СКБД, яка найкраще відповідає потребам даного проекту. Для цього було проведено аналіз

найпоширеніших систем керування базами даних, включаючи комерційні та відкриті рішення. Оцінювання здійснювалося за такими критеріями, як продуктивність, простота інтеграції, підтримка сучасних стандартів, можливість масштабування та вартість володіння.

Результати аналізу представлені в таблиці 3.3, що дозволяє обрати оптимальну СКБД для реалізації проекту. Ця таблиця включає порівняння характеристик різних СКБД, зокрема **MySQL**, **PostgreSQL**, **Oracle Database**, та інших популярних систем, що використовуються у веб-застосунках.

Таблиця 3.3. Порівняльна характеристика СКБД

	MSSQL	**ORACLE**	**MySQL**	**MongoDB**
Надійність збереження даних	+	+	+	+
Швидкість SQL операцій	-	+	+	+
Простота використання	-	-	-	+
Зручність	+/-	-	+	+
Можливість безоплатного комерційного використання	-	-	+	+

Проаналізувавши базові характеристики розглянутих систем керування базами даних (СКБД), було визначено, що **MongoDB** є найбільш зручним і ефективним вибором для довготривалого збереження даних у рамках розробки веб-сайту. Цей вибір зумовлений кількома ключовими перевагами MongoDB:

- **Підтримка різноманітних мов програмування**: MongoDB інтегрується з багатьма мовами програмування, включаючи Java, що полегшує її використання у проекті.
- **Надійність та швидкість**: Сервер MongoDB забезпечує високу продуктивність, що є критично важливим для веб-застосунків, які потребують швидкого доступу до даних.
- **Простота встановлення**: MongoDB легко налаштовується, що знижує поріг входу для розробників.
- **Зручність у використанні**: Завдяки гнучкій моделі документів, MongoDB дозволяє легко працювати з даними, особливо у динамічних веб-застосунках.
- **Безпека та доступність**: MongoDB забезпечує захищений доступ до даних через Internet, що є важливим для веб-застосунків із глобальною аудиторією.

Таким чином, завдяки своїм характеристикам MongoDB повністю задовольняє вимоги до СКБД для розробки веб-сайту, забезпечуючи баланс між доступністю, швидкістю роботи та надійністю. Це робить її оптимальним вибором для реалізації поставлених завдань у магістерській роботі.

3.6. Аналіз архітектур web-сервісів

На сучасному етапі розвитку веб-технологій найбільш поширеними парадигмами розробки веб-сервісів є **SOAP** і **REST**. Кожен із цих стандартів має свої переваги та недоліки, які визначають їхнє використання залежно від потреб проєкту.

Основні відмінності SOAP і REST:

1. **Формати даних**
   - **SOAP** використовує **XML** для кодування запитів і відповідей, що забезпечує сувору типізацію даних і гарантує їх цілісність при передачі між

клієнтом і сервером. Цей підхід є корисним у системах, де критично важлива точність та надійність даних.

- **REST**, у свою чергу, підтримує різноманітні формати даних, зокрема **ASCII**, **XML**, **JSON**, або будь-які інші формати, які підтримуються і клієнтом, і сервером. Завдяки цьому пакети запитів та відповідей у REST мають менший розмір, ніж у SOAP, що сприяє швидшій передачі даних.

2. **Взаємодія з протоколом HTTP**

- У SOAP рівень передачі даних протоколу **HTTP** виконує роль пасивного каналу, який лише передає запити SOAP від клієнта серверу за допомогою методу POST. Усі деталі сервісного запиту (ім’я процедури, аргументи тощо) кодуються у тілі запиту, що ускладнює структуру взаємодії.
- **REST** інтегрується з HTTP на глибшому рівні, використовуючи його стандартні методи (GET, POST, PUT, DELETE) для позначення типів операцій. Цей підхід дозволяє значно спростити структуру запитів і полегшити їхнє розуміння та формулювання.

3. **Розмір і швидкість передачі даних**

- Через складність XML-запитів у SOAP їхній розмір значно більший, ніж у REST, що може впливати на швидкість передачі даних, особливо у високонавантажених системах.
- REST-архітектура дозволяє зменшити обсяг даних завдяки використанню компактних форматів, таких як JSON, що робить її більш ефективною для веб-застосунків.

4. **Простота розробки**

- SOAP забезпечує більшу формальність і сувору структуру, що може бути перевагою для складних інтеграцій, але створює додаткову складність для розробників.
- REST, завдяки використанню стандартних методів HTTP, є більш інтуїтивно зрозумілим для розробників і краще підходить для побудови простих і легких у підтримці сервісів.

Більш детальні результати аналізу наведені в табл.3.4.

Таблиця 3.4. Порівняльна характеристика архітектур web-сервісів

Параметр	**REST**	**SOAP**
Модель зв'язку	Передбачає модель зв’язку "point-to-point", яка не може ефективно працювати у розподіленому середовищі, де повідомлення проходять через кілька посередників	Розроблений для роботи у розподілених обчислювальних середовищах
Інструменти проміжного рівня	Потрібен мінімальний набір інструментів, лише підтримка HTTP	Вимагає використання широкого спектра інструментів проміжного рівня
Визначення дії запиту	URL зазвичай вказує на ресурс, до якого здійснюється доступ, або ж видаляється/оновлюється	Зміст повідомлення визначає операцію, яка має бути виконана, наприклад, через стрічкові літерали
Надійність	Низька надійність - HTTP DELETE може повернути статус OK, навіть якщо ресурс не видалено	Висока надійність
Засоби формального опису	Відсутні засоби для формального опису	Пропонує потужні механізми для опису сервісів, наприклад, WSDL + XSD, WS-Policy
Модифікації запитів	Не вимагає змінювати вміст повідомлень	Повідомлення повинні відповідати схемі SOAP

Параметр	**REST**	**SOAP**
Підтримка стандартів	Використовує лише найбільш популярні стандарти, такі як HTTP і SSL	Підтримує широкий спектр усталених стандартів
Обробка помилок	Містить вбудований механізм для обробки помилок	Відсутній механізм обробки помилок
Транспортна модель	Прив’язана до транспортної моделі HTTP	Підтримує протоколи SMTP і HTTP для транспорту повідомлень

В результаті порівняння двох стандартів розробки веб-сервісів — SOAP і REST — було прийнято рішення про доцільність використання архітектури **REST** для реалізації веб-сервісів у даному проекті. Це рішення обґрунтоване наступними перевагами REST:

- **Стислість повідомлень (запитів/відповідей)**: REST дозволяє використовувати компактні формати даних, такі як JSON, що зменшує розмір повідомлень та прискорює обмін даними.
- **Простота у порівнянні з SOAP**: REST є більш інтуїтивно зрозумілим завдяки використанню стандартних методів HTTP, таких як GET, POST, PUT і DELETE, що спрощує створення та підтримку веб-сервісів.
- **Простота інтеграції**: REST забезпечує легку інтеграцію з різноманітними сторонніми клієнтами, включаючи мобільні пристрої та інші веб-застосунки, завдяки підтримці універсальних форматів даних і стандартів.
- **Використання базових функцій HTTP**: REST максимально використовує можливості протоколу HTTP, зокрема механізми кешування, ідентифікації ресурсів через URI та підтримку безпечного з'єднання (HTTPS).

- **Механізм обробки помилок**: REST надає інструменти для виявлення та обробки помилок через стандартизовані коди стану HTTP (наприклад, 404 для "Не знайдено", 500 для "Помилка сервера"), що спрощує налагодження і діагностику проблем.

Завдяки цим перевагам архітектура REST є оптимальним вибором для побудови веб-сервісів, що потребують високої швидкості роботи, масштабованості та простоти у підтримці.

РОЗДІЛ 4. КОНЦЕПТУАЛЬНА МОДЕЛЬ

Концептуальна модель системи “Web-ужиткок для роботи з контролерами нечіткої логіки” наведена на рис. 4.1.

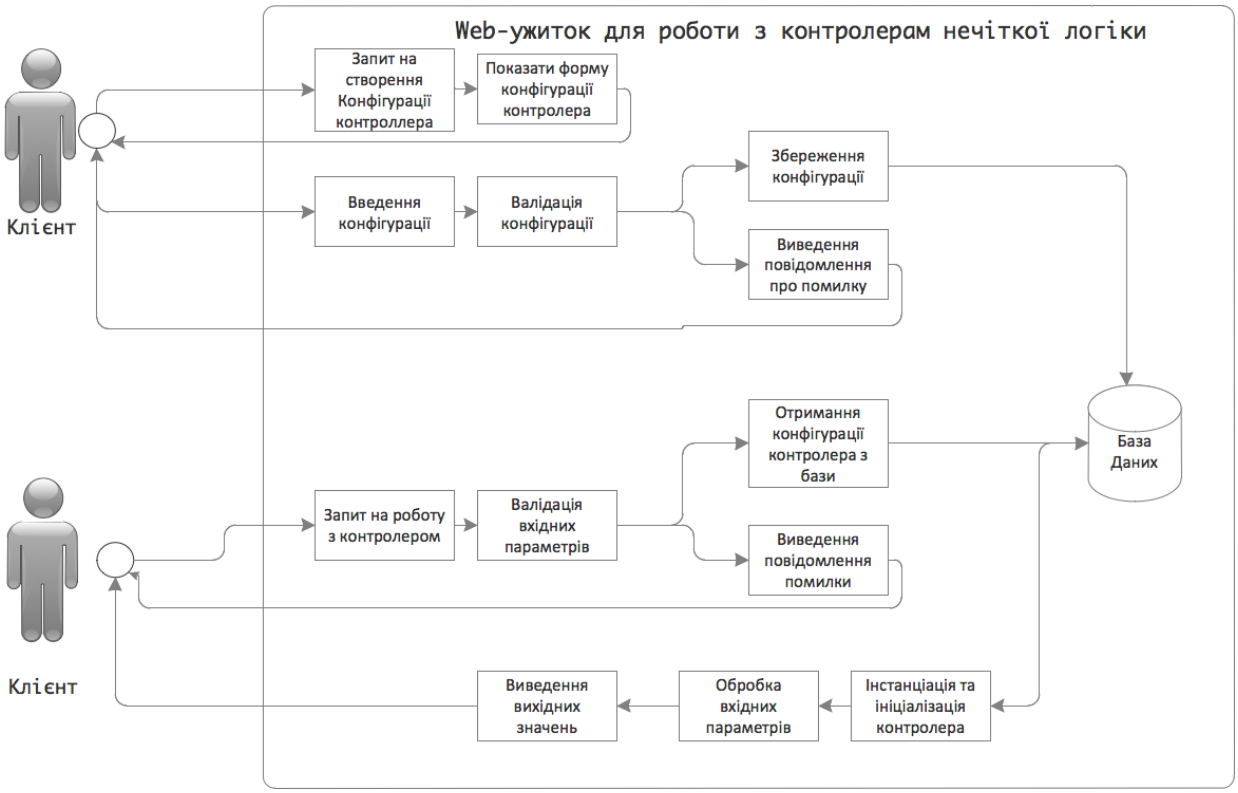

Рисунок 4.1 – Коцептуальна модель системи “Web-ужиткок для роботи з контролерами нечіткої логіки”

У роботі із системою нечіткої логіки можна чітко виділити два основні напрямки:

1. Робота з конфігураціями контролера нечіткої логіки

Цей напрямок охоплює наступні операції:

- **Створення**: користувач має можливість створювати нові конфігурації, які представляють декларативний опис контролера.
- **Редагування**: після створення конфігурацій користувач може змінювати їх відповідно до вимог.

- **Перегляд**: наявні конфігурації доступні для перегляду, що дозволяє аналізувати поточний стан системи.
- **Видалення**: за необхідності користувач може видалити непотрібні конфігурації.

Усі ці дії виконуються через інтуїтивно зрозумілий графічний користувацький інтерфейс. Результати дій безпосередньо впливають на стан даних у базі даних: відповідні записи змінюються або видаляються. Робота з конфігураціями є першочерговим етапом, без якого неможливий перехід до взаємодії з контролером.

2. Робота з контролером нечіткої логіки

Після завершення конфігураційного етапу користувач може працювати з контролером у режимі реального часу. У цьому напрямку система автоматично виконує такі функції:

- **Інстанціювання контролера**: створення екземпляра контролера на основі конфігурації.
- **Ініціалізація контролера**: підготовка контролера до роботи відповідно до переданих параметрів.

Користувачеві потрібно лише визначити:

- вхідні параметри та їх значення;
- конфігурацію контролера, з яким він хоче працювати.

Валідація

Під час роботи із системою здійснюється валідація вхідних даних, яка включає:

- Перевірку на наявність порожніх значень.
- Зіставлення типів введених даних із формальними типами, задекларованими у конфігурації.
- Перевірку відповідності конфігурацій бізнес-вимогам, що забезпечує коректну роботу контролера (відсутність помилок при ініціалізації або передачі параметрів).

Обробка помилок

У разі виникнення помилок валідації чи роботи контролера система інформує користувача через відповідні повідомлення. Ці повідомлення містять опис помилок, що дозволяє швидко ідентифікувати проблему та вжити необхідних заходів.

Такий підхід до роботи з конфігураціями та контролерами забезпечує простоту використання, високу адаптивність системи та мінімізацію ризиків, пов’язаних із некоректним введенням даних або порушенням бізнес-вимог. Це дозволяє ефективно реалізувати інтерактивну взаємодію між користувачем і системою в рамках сучасного веб-інтерфейсу.

РОЗДІЛ 5. РОЗРОБКА WEB-УЖИТКУ

5.1. Розробка структури web-ужитку.

Основні етапи розробки веб-застосунку "Web-ужитку для роботи з контролерами нечіткої логіки" можна структурно розподілити за рівнями розробки, кожен із яких охоплює конкретні завдання та функціональні аспекти:

1. **Загальна структура**

На цьому етапі визначається базова архітектура проекту, яка включає модулі та взаємозв’язки між ними. Це забезпечує чітке розмежування відповідальностей і підвищує гнучкість системи.

2. **Початкова конфігурація**

Включає налаштування проекту, визначення середовища виконання та підключення необхідних залежностей. Цей етап є ключовим для подальшої інтеграції компонентів.

3. **Доступ і робота з базами даних**

Цей рівень забезпечує взаємодію з базою даних, включаючи збереження, оновлення, видалення і отримання даних. Реалізовано за допомогою інструментів ORM (наприклад, Hibernate або JPA).

4. **Рівень бізнес-логіки (сервісний рівень)**

Містить основні алгоритми роботи системи та обробки даних, зокрема керування конфігураціями контролерів нечіткої логіки та їхніми операціями.

5. **Графічний користувацький інтерфейс і логіка навігації між сторінками**

Забезпечує зручний інтерфейс для користувачів, дозволяючи їм створювати, редагувати, переглядати та видаляти конфігурації. Навігація реалізується з урахуванням інтуїтивної зручності.

6. **Web-сервіси для роботи з контролерами**

Реалізуються RESTful або SOAP-сервіси, які забезпечують доступ до функціональності контролерів, зокрема їх ініціалізацію та управління в реальному часі.

7. **Механізм валідації даних**

Реалізує перевірку введених даних на коректність, зокрема перевірку типів, заповнення обов’язкових полів і відповідності бізнес-вимогам.

8. **Механізм представлення роботи системи**

Включає інструменти для візуалізації даних та результатів роботи системи, що дозволяють користувачеві оцінити ефективність налаштувань контролера.

Для управління залежностями проекту, такими як сторонні бібліотеки, а також для його структуризації, було обрано систему **Apache Maven**. Проект поділено на модулі відповідно до їх функціонального призначення, що забезпечує зручність розробки, тестування та підтримки. Кожен модуль проекту відповідає конкретному рівню архітектури і має окремий набір залежностей. Це дозволяє зосередитись на певному аспекті функціональності без впливу на інші частини системи. Структура веб-застосунку представлена у вигляді дерева модулів на рис. 5.1, що наочно демонструє ієрархію компонентів та їхній взаємозв’язок.

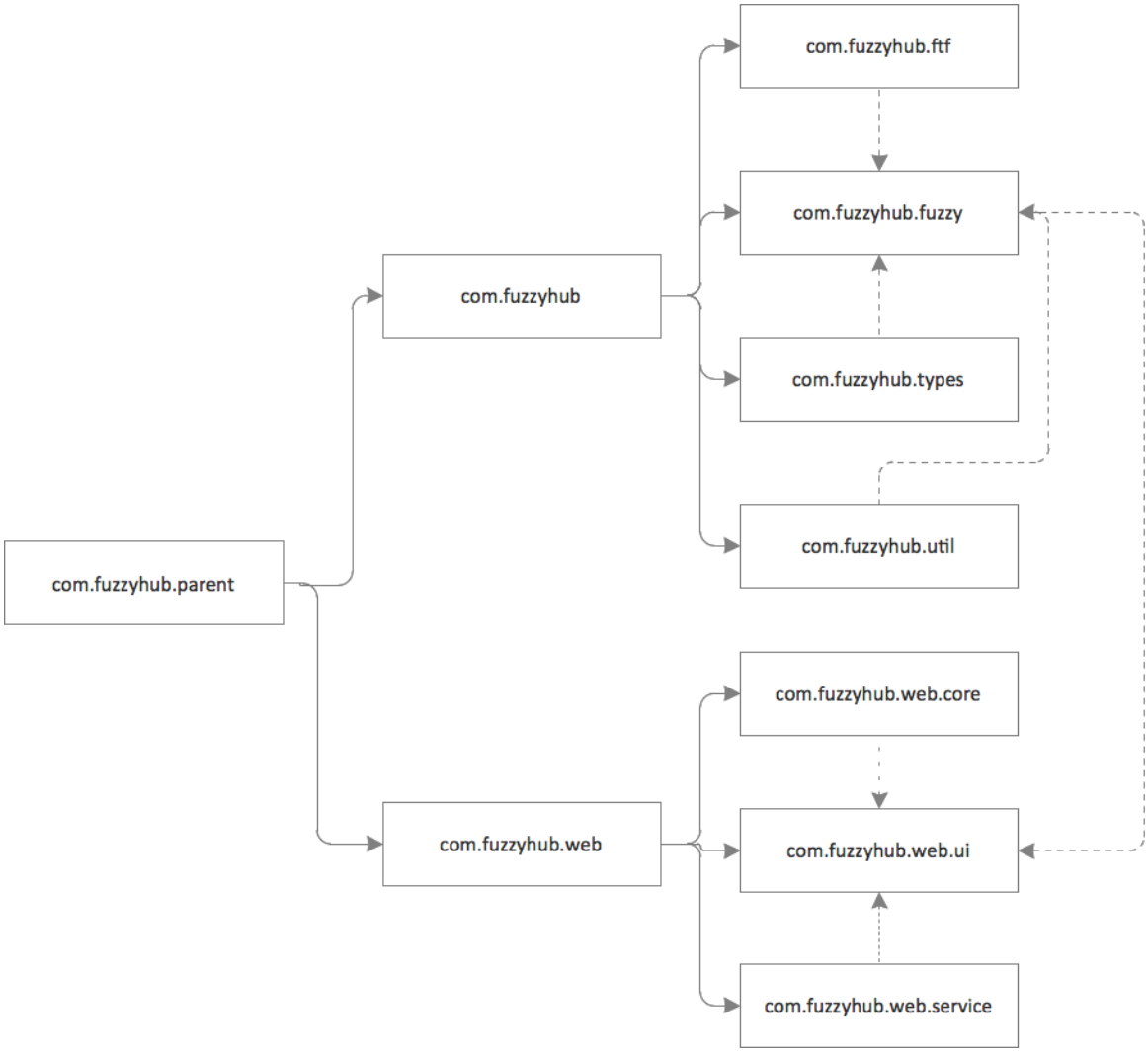

Рисунок 5.1 – Структура web-ужитку у вигляді дерева модулів

Проект веб-застосунку структуровано у вигляді окремих модулів, кожен із яких відповідає за певний функціональний аспект системи. Ось опис кожного з них:

1. **com.fuzzyhyb.parent**

   Основний модуль проекту, який містить усі суб-модулі. Цей модуль визначає загальні залежності, конфігурацію проекту та його структуру.

2. **com.fuzzyhyb**

   Модуль для побудови та роботи з контролерами нечіткої логіки. Відповідає за основну функціональність контролерів, включаючи їх ініціалізацію, конфігурацію та виконання.

3. **com.fuzzyhyb.ftf**

   Модуль, призначений для реалізації системи дефазифікації даних у контролерах нечіткої логіки. Забезпечує механізм перетворення нечітких множин у чіткі значення.

4. **com.fuzzyhub.fuzzy**

   Містить реалізацію самого контролера нечіткої логіки, включаючи його алгоритми, моделі та логіку роботи.

5. **com.fuzzyhub.types**

   Відповідає за визначення системи типів для контролера нечіткої логіки. Містить типи змінних, функцій належності, а також структури даних, які використовуються у системі.

6. **com.fuzzyhub.util**

   Модуль для утилітарних функцій. Забезпечує повторно використовувані інструменти та допоміжні класи, які спрощують роботу з іншими модулями проекту.

7. **com.fuzzyhub.web**

   Базовий модуль, який надає засоби для інтеграції контролерів нечіткої логіки з веб-середовищем. Забезпечує доступ до функціональності контролерів через веб-інтерфейси.

8. **com.fuzzyhub.web.core**

   Модуль, відповідальний за доступ до бази даних, об’єкти доменної логіки та сервісні класи. Реалізує логіку роботи із збереженням, обробкою та доступом до даних.

9. **com.fuzzyhub.web.service**

   Містить веб-сервіси, які надають програмний доступ до контролерів нечіткої логіки. Цей модуль реалізує RESTful або SOAP-сервіси для взаємодії з контролерами.

10. **com.fuzzyhub.web.ui**

    Модуль, що включає розроблені веб-інтерфейси для створення та

редагування конфігурацій контролерів нечіткої логіки, а також для роботи з ними. Саме цей модуль генерує **WAR-файл**, який розгортається на сервері для забезпечення роботи веб-застосунку.

Початкова конфігурація web-ужитку. Наступним етапом після створення структури модулів і визначення їх функціональних ролей є базове налаштування веб-компонентів. Для забезпечення можливості роботи у веб-середовищі у проекті використовується Spring Framework.

Основу Spring Framework складає концепція Inversion of Control (IOC) Container, яка передбачає передачу відповідальності за створення об’єктів та управління їх життєвим циклом від розробника до контейнера. Контейнер контролює процеси:

1. **Створення об’єктів**: Контейнер автоматично створює екземпляри об’єктів відповідно до заданої конфігурації.
2. **Управління залежностями**: Всі необхідні залежності ін’єктуються у компоненти автоматично.
3. **Життєвий цикл об’єктів**: Контейнер визначає, коли і як створюються, використовуються та знищуються об’єкти.
4. **Опис конфігурації**: Налаштування об’єктів задається декларативно у вигляді XML-файлів, Java-класів або через анотації.

Конфігурація Spring IOC Container починається з декларування у web-дескрипторі DispatcherServlet’у:

server: port: 8080 # Define the server port servlet: context-path: /myapp # Base application path spring: mvc: servlet: path: / # Map DispatcherServlet to root path

Далі необхідно задекларувати ім’я файлу в якому розміщені декларації класів для додання в контейнер:

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      classpath:basetControllerContext.xml
```

```
    </param-value>
  </context-param>
```

Після налаштування базового контейнера необхідно конфігурувати "дочірній" контейнер, який відповідатиме за управління класами веб-компонентів (веб-контролерів):

```
import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherSe
rvletInitializer;

public class WebAppInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
      return new Class[] { RootConfig.class }; // Application context
configuration
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
      return new Class[] { WebConfig.class }; // Web-specific configuration
    }
    @Override
    protected String[] getServletMappings() {
      return new String[] { "/" }; // Map the DispatcherServlet to the root path
    }
  }
```

Батьківський контейнер керує всіма іншими компонентами, задіяними у роботі веб-застосунку, і налаштовується наступним чином:

```
import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import
org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import
org.springframework.web.servlet.config.annotation.ViewResolverRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {

   @Override
   public void configureViewResolvers(ViewResolverRegistry registry) {
      registry.jsp("/WEB-INF/views/", ".jsp");
   }

   @Override
   public void addResourceHandlers(ResourceHandlerRegistry registry) {
      registry.addResourceHandler("/resources/**")
            .addResourceLocations("/resources/");
   }

   @Bean
   public InternalResourceViewResolver viewResolver() {
      InternalResourceViewResolver resolver = new
InternalResourceViewResolver();
      resolver.setPrefix("/WEB-INF/views/");
      resolver.setSuffix(".jsp");
      return resolver;
   }
}
```

5.2. Розробка логіки роботи web-ужитку

Для реалізації довготривалого збереження даних у рамках даної магістерської роботи була обрана база даних **MongoDB**. Доступ до бази даних здійснювався за допомогою бібліотеки **mongo-java-driver**, яка є офіційним засобом доступу, наданим розробниками MongoDB.

Основна функціональність mongo-java-driver:

- Створення з'єднання з базою даних.
- Аутентифікація користувача (за необхідності).
- Базова конфігурація параметрів доступу до бази.
- Робота з документами на низькому рівні.

Проте пряме використання mongo-java-driver виявляється складним через необхідність повторення значних обсягів коду. Для спрощення роботи з базою даних було вирішено використовувати бібліотеку **Spring Data MongoDB**.

Spring Data MongoDB є інструментом для об’єктно-документного мапінгу (ODM), який дозволяє працювати з базою даних на рівні об’єктів предметної області. Ця бібліотека інтегрується зі Spring IOC Container, що забезпечує зручність у використанні та керуванні залежностями.

Основні концепції Spring Data MongoDB:

1. **Репозиторії**

   Репозиторії відповідають за управління об’єктами доменної логіки і створюються у співвідношенні один до одного. Кожен репозиторій є інтерфейсом, що наслідується від базового інтерфейсу MongoRepository<T, ID extends Serializable>.

2. **Параметри репозиторіїв**
   - **T** — клас доменної області, з яким працюватиме репозиторій.
   - **ID** — тип унікального ідентифікатора цього об’єкта в базі даних.
3. **Базовий набір операцій**

Інтерфейс MongoRepository надає стандартний набір CRUD-операцій (створення, читання, оновлення, видалення). Завдяки цьому, розробник може зосередитися на описі специфічних методів, таких як пошук за певними критеріями або фільтрація.

Переваги використання Spring Data MongoDB:

- **Простота в роботі**: Бібліотека автоматично генерує основний код для роботи з базою даних, що значно знижує складність розробки.
- **Інтеграція зі Spring**: Підтримка Spring IOC Container дозволяє легко керувати залежностями та впроваджувати репозиторії у сервіси.
- **Мапінг на об'єкти**: Робота з документами бази даних у вигляді об'єктів доменної логіки, що спрощує взаємодію між різними рівнями системи.
- **Розширюваність**: Можливість додавання спеціалізованих методів для роботи з базою даних без повторення базового коду.

Приклад одного з розроблених репозиторіїв наведено нижче:

```
public interface TControllerRepository extends
MongoRepository<TControllerEntity,BigInteger> {
   TControllerEntity findByName(String name);
}
```

Конфігурація SpringDataMongoDB використана в даному проекті наведена нижче:

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.data.mongodb.core.MongoClientFactoryBean;
import
org.springframework.data.mongodb.repository.config.EnableMongoRepositories
;

@Configuration
```

```
@EnableMongoRepositories(basePackages =
"com.fuzzyhub.web.core.repository")
public class MongoConfig {

   @Bean
   public MongoClientFactoryBean mongoClient() {
      MongoClientFactoryBean mongoClientFactoryBean = new
MongoClientFactoryBean();
      mongoClientFactoryBean.setHost("${OPENSHIFT_MONGODB_DB_HOST}");

mongoClientFactoryBean.setPort(Integer.parseInt("${OPENSHIFT_MONGODB_D
B_PORT}"));
      return mongoClientFactoryBean;
   }

   @Bean
   public MongoTemplate mongoTemplate() throws Exception {
      return new MongoTemplate(mongoClient().getObject(),
            "${OPENSHIFT_MONGODB_DB_NAME}");
   }
}
```

Розробка рівня сервісів. Після створення логіки для роботи з базою даних наступним кроком є реалізація рівня сервісів. Ці сервіси представлені у вигляді класів, які інкапсулюють бізнес-логіку роботи з об’єктами доменної області, забезпечуючи чітке розмежування між доступом до даних і бізнес-процесами. Нижче наведено приклад одного з них.

```
@Service
public class ControllerService {
private ProjectsRepositoryBean projectRepository;
   private ControllerRepositoryBEan controllerRepository;

   @Autowired
   public ControllerService(ProjectsRepositoryBean projectRepository,
ControllerRepositoryBean controllerRepository) {
      this.projectRepository = projectRepository;
      this.controllerRepository = controllerRepository;
   }
..// java code here //...
}
```

Даний клас також інстанціюється та ініціалізується за допомогою Spring IOC Container, що дозволяє інтегрувати його в архітектуру системи з мінімальними зусиллями. Він виконує такі ключові функції:

- Інкапсуляція логіки роботи з конфігураціями контролерів нечіткої логіки: Клас забезпечує централізоване управління конфігураціями, дозволяючи уникнути дублювання коду і підвищити зручність роботи з ними.
- Можливість повторного використання бізнес-логіки: Реалізована логіка роботи з конфігураціями може бути застосована в різних контекстах, таких як веб-сервіси, графічний користувацький інтерфейс, віддалений виклик процедур (RPC) тощо.
- Єдиний інтерфейс для роботи з конфігураціями: Клас надає уніфікований API, який дозволяє працювати з конфігураціями без необхідності знання деталей їхнього збереження чи створення. Це спрощує використання сервісу розробниками інших компонентів системи.

5.3. Розробка графічного користувацького інтерфейсу та логіки навігації між сторінками

Перед початком розробки графічного інтерфейсу була розроблена стратегія навігації сторінками web-ужитку (рис 5.2).

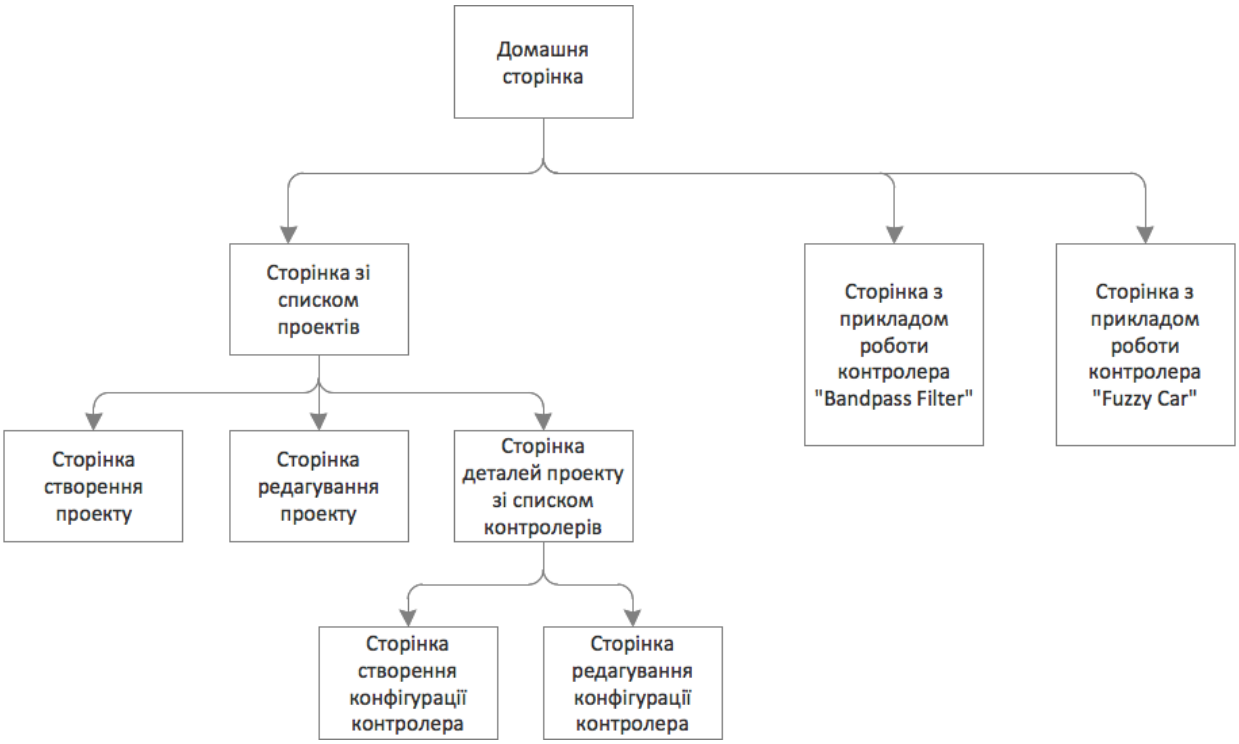

Рисунок 5.2 – Стратегія навігації сторінками web-ужитку

Домашня сторінка слугує початковою точкою взаємодії користувача із системою. Це головний навігаційний центр, з якого користувач може виконати перехід до двох основних розділів:

- **Сторінки проектів**, що містять конфігурації контролерів нечіткої логіки.
- **Сторінок із готовими прикладами застосування розроблених контролерів**, які демонструють можливості та функціональність системи.

Функціональність сторінок:

1. **Сторінка проектів**

Ця сторінка забезпечує управління проектами, що є контейнерами для конфігурацій контролерів нечіткої логіки. Її функціонал включає:

- **Створення нових проектів**: Користувач може ініціювати новий проект із нуля, визначивши його основні параметри.
- **Редагування існуючих проектів**: Надається можливість змінювати назву проекту, його опис чи інші атрибути.
- **Перегляд списку контролерів, пов'язаних із проектом**: Користувач отримує доступ до детальної інформації про всі контролери, створені в рамках конкретного проекту, що спрощує навігацію та пошук.

2. **Сторінка контролерів**

Цей розділ орієнтований на управління конфігураціями контролерів. На сторінці доступні такі дії:

- **Створення нових конфігурацій контролерів**: Користувач може визначити нові параметри або структуру контролера, що дозволяє налаштовувати його для конкретних задач.
- **Редагування існуючих конфігурацій**: Дозволяє вносити зміни до вже створених контролерів, що забезпечує гнучкість і адаптивність до змінюваних вимог.

Технології, використані для розробки інтерфейсу:

1. **Для розробки веб-сторінок**:
   - **JSP (Java Server Pages)**: Ця технологія дозволяє динамічно генерувати веб-сторінки, такі як HTML, XML тощо. Вбудований Java-код забезпечує прямий доступ до серверних даних, що зберігаються в базі даних, дозволяючи створювати інтерактивний інтерфейс.
   - **JSTL (JavaServer Pages Standard Tag Library)**: Надає додаткові можливості для роботи із циклами, умовами,

обробкою URL та інтернаціоналізацією, що значно спрощує створення динамічних сторінок.

- **Spring Forms**:

Розширює функціонал JSTL, дозволяючи безпосередньо працювати з Java-об’єктами на стороні сервера та автоматично генерувати HTML-форми.

2. **Для дизайну сторінок**:
   - **Twitter Bootstrap Framework**:

Цей фреймворк забезпечує створення адаптивного дизайну сторінок, який коректно відображається на пристроях із різними розмірами екранів. У комплекті з Bootstrap надаються готові CSS-стилі та JavaScript-бібліотеки, які використовуються для додавання анімацій, інтерактивних елементів і макетів.

Рис. 5.3 демонструє форму конфігурації нечітких лінгвістичних змінних, яка дозволяє користувачеві налаштовувати входи, виходи та функції належності для конкретного контролера. Рис. 5.4 ілюструє форму для конфігурації правил роботи контролера, що є ключовою частиною роботи з нечіткою логікою.

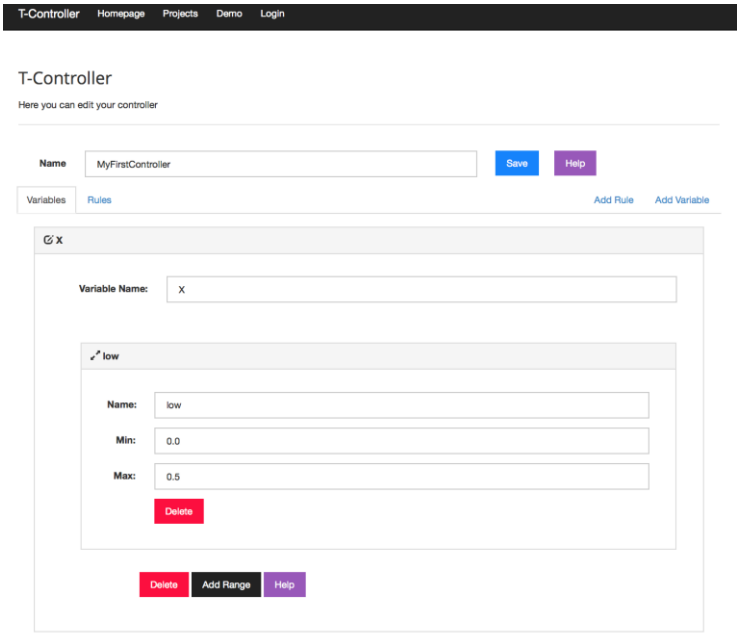

Рисунок 5.3 – Форма конфігурації нечітких лінгвістичних змінних

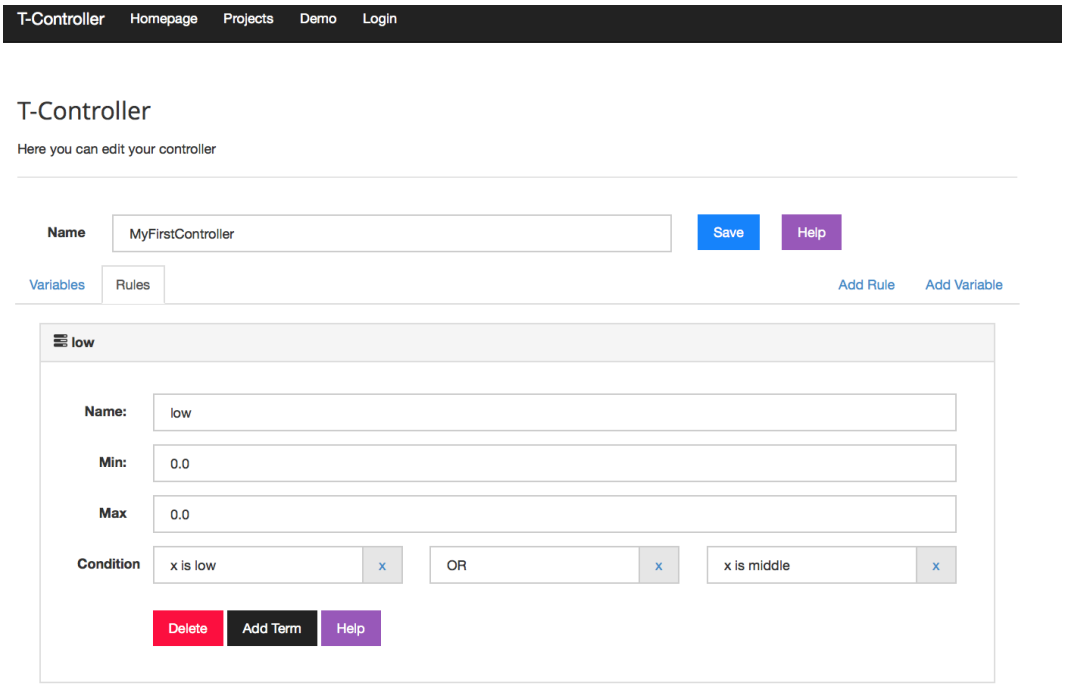

Рисунок5.4 – Форма конфігурації правил роботи контролера нечіткої логіки

Для забезпечення навігації між сторінками та передачі даних між ними було використано ще один компонент Spring — Spring MVC. Spring MVC є реалізацією архітектурного патерну Model-View-Controller (MVC) у веб-застосунках. При використанні Spring MVC:

- Модель (Model) представлена Java POJO (Plain Old Java Objects), які містять дані та бізнес-логіку.
- Контролери (Controller) реалізуються як спеціально анотовані Java-класи, що відповідають за обробку запитів і взаємодію з моделлю.
- Відображення (View) здійснюється через JSP-сторінки зі Spring Forms, які відповідають за представлення даних користувачу.

Приклад одного з контролерів ужитку:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.ModelAndView;
```

```
@Controller
@RequestMapping("/projects/{projectName}/users")
public class UsersController {

   @Autowired
   private UserManager userManager;

   @Autowired
   private ProjectManager projectManager;

   @RequestMapping(method = RequestMethod.GET)
   public ModelAndView showProjectUsers(@PathVariable("projectName")
String projectName) {
      ModelAndView usersPage = new ModelAndView("UsersPage");
      usersPage.addObject("project", projectManager.get(projectName));
      usersPage.addObject("users",
userManager.getUsersByProject(projectName));
      return usersPage;
   }

   @RequestMapping(method = RequestMethod.POST)
   public String addUserToProject(@PathVariable("projectName") String
projectName, @RequestParam("userName") String userName) {
      userManager.addUserToProject(projectName, userName);
      return "redirect:/projects/" + projectName + "/users";
   }

   @RequestMapping(value = "/{userId}", method = RequestMethod.DELETE)
   public String removeUserFromProject(@PathVariable("projectName") String
projectName, @PathVariable("userId") Long userId) {
      userManager.removeUserFromProject(projectName, userId);
      return "redirect:/projects/" + projectName + "/users";
   }
}
```

Контролери в Spring MVC повинні бути позначені анотацією @Controller, яка вказує на те, що клас виконує роль контролера у патерні MVC. Запити, які обробляються контролером, визначаються за допомогою анотації @RequestMapping, де описується шлях URL та метод HTTP (наприклад, GET або POST).

Кожен метод контролера в результаті своєї роботи повертає:

- Або ім’я відображення (назву JSP-сторінки), на яку потрібно перенаправити користувача.
- Або об’єкт ModelAndView, що містить як ім’я відображення, так і дані, які потрібно передати для відображення.

Для виконання бізнес-логіки контролери взаємодіють із об’єктами сервісного рівня, що інкапсулюють основну логіку роботи з даними. Такий підхід забезпечує чітке розмежування відповідальностей та спрощує підтримку системи.

5.4. Розробка механізму валідації даних

Валідація даних у Java реалізована відповідно до стандарту **JSR-303**, який підтримує кілька популярних імплементацій. У рамках цієї роботи було обрано **JbossValidator**, оскільки він забезпечує найповніший функціонал і є одним із найпростіших у використанні.

Валідація виконується у контролерах Spring Framework під час процесу прив’язування даних, отриманих від клієнта, до POJO-класу, який слугує моделлю даних. Це дозволяє автоматично перевіряти правильність введених значень, дотримання обов’язкових полів та відповідність вказаним критеріям.

Приклад розробленого контролера який здійснює валідацію наведено нижче:

```
@RequestMapping(value = "create", method = RequestMethod.POST)
public String createNewUser(
    @Valid @ModelAttribute("user") User user,
    BindingResult bindingResult,
    @RequestParam("projectName") String projectName) {

  if (bindingResult.hasErrors()) {
    return "EditUserPage";
  }

  if (userService.createUserInProject(projectName, user) == null) {
```

```
        bindingResult.addError(new ObjectError("user", "A user with this name
already exists in the project"));
        return "EditUserPage";
    }

    return "redirect:/projects/" + projectName + "/users";
}
```

Для валідації параметрів необхідно анотувати поле за допомогою @Valid. Також можна визначити правила валідації для об’єкта, анотуючи відповідні поля з використанням відповідних обмежень, як показано нижче:

```
    @Size(min = 1, message = "Опис повинен містити принаймні один
символ")
    private String description;

    @NotNull(message = "Статус має бути вказаний")
    private String status;

    @Valid
    @NotNull
    private List<TaskEntity> tasks;

    @Valid
    @NotNull
    private UserEntity assignedUser;
```

У цьому прикладі:

- @Size(min = 1) забезпечує, що поле description містить принаймні один символ.
- @NotNull перевіряє, щоб поля status, tasks та assignedUser не були порожніми.
- @Valid активує рекурсивну валідацію для складних полів, таких як tasks та assignedUser, перевіряючи їхні внутрішні властивості відповідно до їхніх власних правил.

5.5. Розробка web-сервісів для роботи з контролерами

Для реалізації веб-сервісів у цьому проекті була обрана архітектура REST, яка забезпечує простоту, масштабованість і ефективність обміну даними між клієнтом і сервером. Основним стандартом, що дозволяє використовувати REST у Java, є JAX-RS (Java API for RESTful Web Services). JAX-RS має кілька популярних імплементацій, серед яких найбільш популярними є Apache CXF, Jersey та JBoss RESTEasy.

У рамках даної роботи було обрано Jersey. Основною причиною вибору є те, що Jersey підтримує інтеграцію зі Spring Framework, що дозволяє використовувати вже налаштовані Spring-компоненти. Крім того, Jersey забезпечує можливість використання об'єктів, які конфігуруються через Spring IOC Container, безпосередньо всередині веб-сервісів. Це значно спрощує розробку, інтеграцію та налаштування веб-сервісів.

Реалізовані веб-сервіси у проекті повторно використовують код сервісних класів для виконання бізнес-логіки. Це дозволяє уникати дублювання коду, забезпечуючи централізоване управління логікою роботи із даними. Завдяки такому підходу веб-сервіси виступають як посередники, що координують запити від клієнта та передачу результатів, залишаючи обробку даних сервісному рівню.

Перед початком роботи з Jersey його потрібно сконфігурувати через web-дескриптор. Конфігурація використана в даному проекті наведена нижче.

```
import com.fuzzyhub.web.cors.ResponseCorsFilter;
import org.glassfish.jersey.server.ResourceConfig;
import org.springframework.context.annotation.Configuration;

import javax.ws.rs.ApplicationPath;

@Configuration
@ApplicationPath("/service")
```

```
public class JerseyConfig extends ResourceConfig {
   public JerseyConfig() {
      // Register packages containing REST resources
      packages("com.fuzzyhub.web.service");
      // Enable POJO Mapping for JSON serialization/deserialization
      property("jersey.config.server.disableMoxyJson", true);
      // Register custom filters and other components
      register(ResponseCorsFilter.class);
   }
}
```

Далі наведено один з розроблених web-сервісів.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import java.util.List;

@Component
@Path("/users")
public class UserService {

   @Autowired
   private UserManager userManager;

   @GET
   @Produces(MediaType.APPLICATION_JSON)
   public List<User> getUsers() {
      return userManager.getAll();
   }
```

```
    @GET
    @Path("/{userId}")
    @Produces(MediaType.APPLICATION_JSON)
    public User getUserById(@PathParam("userId") Long userId) {
      return userManager.getById(userId);
    }

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.TEXT_PLAIN)
    public String addUser(User user) {
      userManager.create(user);
      return "User created successfully";
    }

    @DELETE
    @Path("/{userId}")
    @Produces(MediaType.TEXT_PLAIN)
    public String deleteUser(@PathParam("userId") Long userId) {
      userManager.delete(userId);
      return "User deleted successfully";
    }
  }
```

Веб-сервіси, розроблені за допомогою Jersey, також повинні бути додані до Spring IOC Container для забезпечення інтеграції з іншими компонентами системи. Для цього класи веб-сервісів анотуються за допомогою

@Component, що дозволяє Spring автоматично виявити та керувати ними як частиною контейнера.

Щоб визначити типи HTTP-запитів, які має обробляти конкретний метод сервісу, використовуються відповідні анотації: @GET, @POST, @DELETE тощо. Вони вказують, який метод HTTP протоколу відповідає даному методу сервісу. Для налаштування відносного шляху ресурсу, з яким працює веб-сервіс, застосовується анотація @Path, яка визначає URL для доступу до даного ресурсу.

Як було зазначено раніше, інтеграція Jersey з Spring дозволяє веб-сервісам отримувати доступ до всіх об'єктів, які містяться в Spring IOC Container. Це забезпечує просту взаємодію з іншими компонентами, такими як сервіси або репозиторії. Для отримання об'єкта з контейнера використовується анотація @Autowired, яка автоматично ін’єктує потрібний об'єкт. Достатньо вказати клас, об’єкт якого має бути отриманий, і Spring самостійно подбає про його надання веб-сервісу.

Механізм представленя роботи сиетеми. Для демонстрації роботи системи було створено інтерактивне демо з використанням **FrozenFramework**, HTML5-фреймворку, призначеного для розробки ігор. Це рішення дозволило візуалізувати функціональність розробленого веб-застосунку у зрозумілому та інтерактивному вигляді.

У рамках демо було реалізовано контролер нечіткої логіки, який керує рухом автомобіля на площині. Контролер динамічно обчислює оптимальний кут руху транспортного засобу для ефективного уникнення перешкод.

Для забезпечення інтеграції між демо та веб-застосунком використовується простий **JavaScript-клієнт**. Він взаємодіє з веб-сервісами застосунку, отримуючи дані про поточний стан автомобіля та відправляючи необхідні параметри для розрахунку руху. На основі отриманих результатів клієнт змінює кут руху автомобіля, що дозволяє йому об'їжджати перешкоди.

```
define([
    "dcl",
    "dojo/Deferred"
], function (dcl, Deferred) {
    'use strict';
    class FuzzyClient {
        constructor() {
            this.serverHost = "/service/DemoProject/CarController/query";
            this.distanceKey = "distance";
            this.angleKey = "angle";
            this.angle = 0.5;
        }
        async getAngle(distance, angle) {
            const url = `${this.serverHost}?${this.distanceKey}=${distance}&${this.angleKey}=${angle}`;
            try {
                const response = await fetch(url, { method: 'GET' });
                if (response.ok) {
                    const value = await response.text();
                    this.angle = parseFloat(value);
                    console.log("fuzzyAngle = ", this.angle);
                } else {
                    console.error(`Error fetching angle: ${response.status} ${response.statusText}`);
                }
            } catch (error) {
```

```
                console.error("Request failed", error);
            }
        }
    }

    return FuzzyClient;
});
```

РОЗДІЛ 6. ЕКОНОМІЧНА ОЦІНКА ПРОЕКТНОГО РІШЕННЯ

6.1. Економічна характеристика проектного рішення

На вітчизняному ринку комерційних систем на основі нечіткої логіки перші розробки почали з'являтися в середині 1995 року. З того часу найбільш популярними серед замовників стали такі програмні пакети:

- **CubiCalc 2.0 RTC** – одна з найпотужніших експертних систем, створених на базі нечіткої логіки, яка дозволяє користувачам розробляти власні прикладні експертні системи.
- **CubiQuick** – спрощена й доступніша версія пакета CubiCalc, орієнтована на навчальні заклади.
- **RuleMaker** – програма, що автоматично витягує нечіткі правила з вхідних даних.
- **FuziCalc** – електронна таблиця з підтримкою нечітких полів, яка дозволяє проводити швидкі оцінки за умов неточної або часткової інформації без нагромадження похибки.
- **OWL** – пакет, що містить вихідний код для різних видів нейронних мереж, нечіткої асоціативної пам'яті тощо.

Головними споживачами систем на основі нечіткої логіки у країнах СНД є банкіри, фінансисти, а також фахівці у галузях політичного та економічного аналізу. Наприклад, **CubiCalc** активно застосовується для моделювання економічних, політичних і біржових ситуацій. У той же час, пакет **FuziCalc** отримав популярність серед банкірів і спеціалістів із надзвичайних ситуацій завдяки можливості швидких розрахунків в умовах неповноти або неточності вхідних даних.

Незважаючи на вже значні досягнення, можна з упевненістю сказати, що розквіт прикладного використання нечіткої логіки на вітчизняному ринку ще попереду.

Елементи нечіткої логіки сьогодні впроваджені у десятках промислових виробів. Наприклад, вони застосовуються в системах керування

електропоїздами, бойовими вертольотами, а також у побутовій техніці, такій як пилососи та пральні машини. Багато компаній, особливо японських, активно використовують цей напрямок як конкурентну перевагу у своїх рекламних кампаніях.

Нечітка логіка стала основою для створення сучасних ситуаційних центрів керівників західних країн, у яких приймаються важливі політичні рішення та моделюються різноманітні кризові ситуації. Одним із найбільш вражаючих прикладів масштабного застосування нечіткої логіки стало моделювання системи охорони здоров'я та соціального забезпечення у Великобританії (National Health Service – NHS). Це рішення дозволило вперше точно оцінити та оптимізувати витрати на соціальні потреби, зробивши значний внесок у розвиток цієї галузі.

6.2. Розрахунок витрат на розроблення та впровадження проектного рішення

1) Витрати на розробку і впровадження програмного засобу (К) визначаються як:

$$K_{заг}=K_1+K_2, \qquad (6.1)$$

де $К_1$ - витрати на розробку програмного засобу, грн.; $К_2$ - витрати на налагодження і дослідну експлуатацію програмного засобу на ЕОМ, грн.

Витрати на розробку програмного засобу включають в себе:

1. витрати на оплату праці розробників ($В_{ОП}$);
2. ЄСВ ($В_{єсв}$);
3. вартість додаткових виробів, що закуповуються ($В_Д$);
4. накладні витрати ($Н_В$);
5. інші витрати (Ів).

Для розрахунку витрат на оплату праці враховуємо чисельність працівників, їхню середньоденну заробітну плату та трудомісткість у людино-днях. У проекті залучено п'ять осіб: керівник проекту, двоє інженерів-програмістів та двоє тестувальників (інженерів із забезпечення якості). Заробітна плата становить 5000 грн для керівника проекту, 3600 грн для інженерів-програмістів та 2000 грн для тестувальників. Трудомісткість робіт становить 5 днів для керівника проекту, 36 днів для кожного з інженерів-програмістів та 7 днів для кожного з тестувальників.Середньоденна заробітна плата *i*-го розробника ($ЗП_{Дi}$) обчислюється за формулою:

$$ЗП_{Дi},= ЗП_i / Фм, \quad (6.2)$$

де $ЗП_i$ - основна місячна заробітна плата розробника *i*-ої спеціальності, грн.; *Фм* - місячний фонд робочого часу, днів (24 дні).

$$ЗП_{д1} = 5000 / 24=208,33 \text{ грн.}$$
$$ЗП_{д2} = 3600 / 24 =150,00 \text{ грн.}$$
$$ЗП_{д3} = 2000 / 24 = 83, 33 \text{ грн.}$$

Розрахунок витрат на оплату праці усіх розробників проекту обчислюємо за формулою:

$$В_{ОП} = \sum_{i=1}^{N} n_i \cdot t_i \cdot ЗП_{Дi}, \quad (6.3)$$

де n_i - кількість розробників проекту *i*-ої спеціальності, чол.; t_i - час, витрачений на розробку проекту працівником *i*-ої спеціальності, дні; $ЗП_{Дi}$ - денна заробітна плата розробника *i*-ої спеціальності, грн.;

$В_{ОП} = (1 \cdot 5 \cdot 208,33) + 2 \cdot (1 \cdot 36 \cdot 150,00) + 2 \cdot (1 \cdot 7 \cdot 83,33) =13008,27$ грн.

Розрахунок витрат на оплату праці розробників зводиться у (табл. 6.1).

Таблиця 6.1. Розрахунок витрат на оплату праці

№	Спеціальність розробника	Кількість розробн.	Час роботи, дні	Денна заробітна плата розробника, грн.	Витрати на оплату праці, грн.
1	Керівник проекту	1	5	208,33	1041,65
2	Інженер-програміст	2	36	150,00	10800
3	Тестер	2	7	83, 33	1166,62
Всього					13008,27

2) Витрати на оплату праці працівникам тягнуть за собою додаткові зобов'язання підприємства перед державними фондами обов'язкового страхування $В_Ф$. Підприємство зобов'язане здійснювати відрахування ЄСВ (36,3 % від фонду оплати праці для ІТ-працівників):

$$В_Ф = 8274,47 \cdot 0,363 = 4722,01 \text{ грн.}$$

3) Витрати на додаткові вироби, що закуповуються $В_Д$ (папір, канцелярські вироби тощо, HDD – носій інформації) визначаються за їхніми фактичними цінами з врахуванням найменування, номенклатури та необхідної їх кількості в проекті. Вихідні дані та результати розрахунків оформлено у (табл. 6.2).

$$В_{Д1} = 1 \cdot 66,98 = 66,98 \text{ грн.}$$

$$В_{Д2} = 1 \cdot 2\ 500,00 = 2\ 500,00 \text{ грн.}$$

Таблиця 6.2 Розрахунок витрат на закуплені вироби

№ п/п	Найменування виробу	Марка, тип	К-ть, шт.	Ціна за од., грн.	Сума, грн.
1	Папір для принтера	A4 80г/м IQ Economy /Pol Speed / пачка	1	66,98	66,98
2	Server	Impression HomeServer 0111	1	2 500,00	2 500,00
Всього:					2566.98

4) Витрати на придбання спеціального обладнання (ВСО) для проведення експериментальних робіт розраховуються у випадках, коли для розробки та впровадження проектного рішення необхідно придбати додаткові технічні засоби. У цьому проекті передбачено використання програмного засобу, для роботи якого потрібен маршрутизатор для доступу до мережі Інтернет.

Зважаючи на економічну доцільність, було прийнято рішення про придбання безпровідного маршрутизатора. Використання цього типу обладнання дозволяє уникнути додаткових витрат на придбання та встановлення дротових маршрутизаторів. Для реалізації проекту обрано маршрутизатор **TP-LINK TL-WR740N**, який відповідає вимогам щодо технічних характеристик та вартості.

Вартість спеціального обладнання для виконання проекту розраховується на основі специфікації в потребі технічних засобів і фактичних цін, включаючи заготівельні витрати. Вихідні дані та результати розрахунків представлені у таблиці 6.3, яка демонструє підсумкову вартість придбаного обладнання та обґрунтовує вибір моделі маршрутизатора.

Таблиця 6.3 Розрахунок вартості спецобладнання

№ п/п	Найменування обладнання	Марка, тип	Кількість на проект, шт.	Ціна за одиницю, грн.	Сума витрат грн.
1.	Маршрутизатор	TP-LINK TL-WR740N	1	247,00	247,00
Всього:					247,00

5) Накладні витрати ($В_н$) проектних організацій включають три групи видатків: витрати на управління; загальногосподарські; невиробничі витрати. Вони розраховуються за встановленими 20% до витрат на оплату праці.

$$В_н = 13008,27 \cdot 0,2 = 2601,66 \text{ грн.}$$

6) Інші витрати ($В_{ін}$) складають видатки, які не враховані в попередніх статтях витрат. Вони розраховуються за встановленими 7% до витрат на оплату праці.

$$В_{ін} = 13008,27 \cdot 0,07 = 910,58 \text{ грн.}$$

7) Витрати на розробку проектного рішення обчислюємо за формулою:

$$К_1 = В_{ОП} + В_Ф + В_Д + В_н + В_{ін} + В_{СО}; \quad (6.4)$$

$К_1 = 13008,27+4722,01+2566.98+1654,91+579,22+910,58= 23441.97$ грн.

8) Витрати на відлагодження і дослідну експлуатацію системи визначаємо згідно формули:

$$K_2 = S_{ms} \cdot t_{від}, \quad (6.5)$$

де S_{ms} - вартість однієї години роботи ПК, яка становить 4,5 грн./год; $t_{від}$ - кількість годин роботи ПК на відлагодження програми. Для відлагодження програми потрібно 36 день для програмістів та 7 днів для тестерів по 8 год/день.

$$K_2 = 4{,}5 \cdot (2 \cdot 7 \cdot 8 + 2 \cdot 36 \cdot 8) = 3096{,}00 \text{ грн.}$$

Отже, витрати на розробку і впровадження програмного засобу становлять:

$$K_{заг} = K_1 + K_2 = 23441{,}97 + 3096{,}00 = 26537{,}97 \text{грн.}$$

Результати розрахунків зведено у (табл.6.4).

Таблиця 6.4 Кошторис витрат на розробку проектного рішення

№ п/п	Найменування елементів витрат	Сума витрат, грн.
1	Витрати на оплату праці	13008,27
2	Відрахування у спеціальні державні фонди	4722,01
3	Витрати на додаткові вироби, що купуються	2566.98
4	Накладні витрати	2601,66
5	Витрати на придбання спецобладнання	247,00
6	Інші витрати	910,58
7	Витрати на відлагодження і дослідну експлуатацію системи	3096,00
Всього		26537,97

6.3. Визначення комплексного показника якості

Комплексний показник якості *(Пя)* визначається шляхом порівняння показників якості проектованої системи і вибраного аналогу.

За аналог вибирається прилад та програма, які використовуються для тестування якості відбитків . Для визначення *Пя* використовується система показників технічного рівня і якості, яка містить в собі наступні групи:

- показники призначення (актуальність, реальність, доступність);
- показники надійності (ступінь інформативності);
- показники безпеки (рівень захисту);
- патентно-правові показники (застереження авторських прав, відповідальність нормативним вимогам);
- ергономічні показники (легкість експлуатації).

Комплексний показник якості проектованої системи визначаємо методом арифметичного середньозваженого з формули:

$$П_Я = \sum_{i=1}^{m} C_i * q_i, \quad (6.6)$$

де m - кількість одиничних показників (параметрів), прийнятих для оцінки якості проектованої системи; q_i - коефіцієнт вагомості кожного з параметрів щодо їхнього впливу на технічний рівень та якість проектованої системи (встановлюється експертним шляхом), причому:

$$\sum_{i=1}^{m} q_i = 1.0 \quad (6.7)$$

C_i - часткові показники якості, визначені порівнянням числових значень одиничних показників проектованої системи і аналога за формулами:

$$C_i = \frac{П_{прі}}{П_{аі}} \text{ або } C_i = \frac{П_{аі}}{П_{прі}} \quad (6.8)$$

де $П_{прі}$, $П_{аі}$ - кількісні значення i-го одиничного показника якості відповідно проектованої системи і аналога.

З попередніх двох формул вибирається та, в якій збільшення відповідає покращенню показника якості проектованої системи. Результати розрахунку зводимо в (табл. 6.5).

Таблиця 6.5 Визначення комплексного показника якості

Показники	Числове значення показників, бали		Частковий показник якості, C*i*	К-нт вагомості qi	C*i*·q*i*
	Аналог	Проект. прогр. продукт			
1. Показники призначення					
1.1. Швидкість завантаження проекту	3	10	3,3	0,3	0,99
1.2. Керованість	6	10	1,67	0,15	0,25
1.3. Можливість розширення і модифікації функцій програмного забезпечення	10	7	0,7	0,1	0,07
2. Показники надійності					
2.1. Стійкість системи до некоректних дій користувача	10	9	0,9	0,1	0,09
2.2. Стійкість до збоїв системи	10	9	0,9	0,05	0,045
3. Показники безпеки					
3.1. Рівень захисту	7	10	1,42	0,05	0,071
4. Патентно-правові показники					
4.1 Застереження авторський прав	5	10	2	0,05	0,1
4.2 Відповідність нормативним вимогам	10	10	1	0,05	0,05
5. Ергономічні показники					
5.1. Легкість в експлуатації системи	4	10	2,5	0,15	0,375
Разом				**1**	**2,041**

Отже, комплексний показник якості дорівнює $П_Я$ = 2,041.

6.4. Визначення експлуатаційних витрат

При порівнянні програмних засобів в експлуатаційні річні витрати включають вартість підготовки даних (E_1) і вартість годин роботи ПК (E_2) і визначаються за формулою:

$$E_{П(А)}=E_{1П(А)}+E_{2П(А)} \ , \tag{6.9}$$

де $E_{П(А)}$ - одноразові експлуатаційні витрати на проектне рішення (аналог), грн.; $E_{1П(А)}$ - вартість підготовки даних для експлуатації проектного рішення (аналогу), грн.; $E_{2П(А)}$ - вартість машино-годин роботи ПК для проектного рішення (аналогу), грн.

Річні експлуатаційні витрати визначаються за формулою:

$$B_{(е)П(А)}= E_{П(А)} \cdot N_{П(А)} \ , \tag{6.10}$$

де $B_{(е)П(А)}$ - експлуатаційні річні витрати проектного рішення; $N_{П(А)}$ - періодичність експлуатації проектного рішення (аналогу), раз/рік.

Вартість підготовки даних для роботи на ПК (E_1) визначаються за формулою:

$$E_1 = \sum_{i=1}^{N} n_i \cdot t_i \cdot ЗП_{2i} \tag{6.11}$$

де i - номери категорій персоналу, які беруть участь у підготовці даних; n_r- кількість співробітників і-ї категорії, чол.; t_i - трудомісткість роботи співробітників i-ї категорії, дні; $ЗП_{2i}$- середньогодинна ставка робітника i-ї категорії з врахуванням відрахувань ЄСВ, грн./год.

Середньогодинна ставка оператора визначається за формулою:

$$ЗП_{г_i} = \frac{ЗП_{г0i}(1+b)}{Ф_г} \quad (6.12)$$

де $ЗП_{г_i}$ - основна місячна зарплата працівника *i*-ї категорії, грн.; *b* - коефіцієнт, який враховує ЄСВ (b=0,363); *Фг* - місячний фонд робочого часу, год.

Отже, для проектного рішення середньогодинна ставка становить:

$$ЗП_{г1} = 5000 \cdot (1+0,363) / 24 \cdot 8 = 35,45 \text{ грн.}$$
$$ЗП_{г2} = 3600 \cdot (1+0,363) / 24 \cdot 8 = 25,56 \text{ грн.}$$
$$ЗП_{г3} = 2000 \cdot (1+0,363) / 24 \cdot 8 = 14,19 \text{ грн.}$$

Вартість підготовки даних для роботи на ПК дорівнює:

$$E_{1П} = (1 \cdot 5 \cdot 35,45) + 2 \cdot (1 \cdot 36 \cdot 25,56) + 2 \cdot (1 \cdot 7 \cdot 14,19) = 2216,23 \text{ грн.}$$

Вартість машино-годин роботи ПК для проектного рішення була визначена у п.5.1. і становить 3096,00 грн. Тоді одноразові експлуатаційні витрати на проектне рішення становлять:

$$E_П = 2216,23 + 3096,00 = 5312,23 \text{ грн.}$$

А річні експлуатаційні витрати з урахуванням того, що періодичність експлуатації проектного рішення дорівнює 20 разів/рік, становлять:

$$B_{(е)П} = 5312,23 \cdot 20 = 106244,6 \text{ грн.}$$

Над проектом-аналогом працює 1 керівник проекту, 2 інженери-програмісти, 1 апаратний інженер та 2 тестери. Їхні місячні заробітні ставки відповідно 5000 грн., 3600 грн., 3900 грн., та 2100 грн. Тоді середньогодинна ставка для кожного з них становить:

$$ЗП_{г1} = 5000 \cdot (1+0,363) / 24 \cdot 8 = 35,87 \text{ грн.}$$

$$ЗП_{г2} = 3600 \cdot (1+0,363) / 24 \cdot 8 = 25,56 \text{ грн.}$$

$$ЗП_{г3} = 3900 \cdot (1+0,363) / 24 \cdot 8 = 27,69 \text{ грн.}$$

$$ЗП_{г4} = 2100 \cdot (1+0,363) / 24 \cdot 8 = 14,91 \text{ грн.}$$

Вартість підготовки даних для роботи на ПК дорівнює:

$E_{1A} = (1 \cdot 10 \cdot 35,87) + (2 \cdot 40 \cdot 25,56) + (1 \cdot 15 \cdot 27,69) + (2 \cdot 10 \cdot 14,91) =$ 3117,05грн.

Вартість машино-годин роботи ПК для проекту-аналогу згідно формули (5.5) становить 4140 грн. Тоді одноразові експлуатаційні витрати на аналог становлять:

$$E_A = 3117,05 + 4140 = 7257.05 \text{ грн.}$$

А річні експлуатаційні витрати з урахуванням того, що періодичність експлуатації аналогу дорівнює 20 раз/рік, становлять:

$$B_{(e)A} = 7257.05 \cdot 20 = 145141 \text{грн.}$$

Вихідні дані та результати розрахунків витрат на підготовку даних для експлуатації на ЕОМ зводяться у (табл.6.6)

Таблиця 6.6 Розрахунок витрат на підготовку даних для роботи на ЕОМ

Категорія персоналу	Чисельність співробітників і-ої категорії, люд.,	Час роботи співробітників і-ої категорії, год.	Середньогодинна ЗП співробітника і-ої категорії, грн.	Витрати на підготовку даних, грн.
Проектне рішення				
Керівник проекту	1	5	35,45	177.25
Інженер-програміст	2	36	25,56	1840.32
Тестер	2	7	14,19	208.6
Всього	-	-	-	**2216,23**
Аналог				
Керівник проекту	1	10	35,87	358.7
Інженер-програміст	2	40	25,56	2044.8
Апаратний інженер	1	15	27,69	415.35
Тестер	2	10	14,91	298.2
Всього	-	-	-	**3117,05**

6.5. Розрахунок ціни споживання проектного рішення

Ціна споживання *(Цс)* – це витрати на придбання і експлуатацію проектного рішення за весь строк його служби:

$$Ц_{С(П)} = Ц_П + B_{(E)NPV} \quad (6.13)$$

де $Ц_П$ – ціна придбання проектного рішення, грн.; $B_{(E)NPV}$ – теперішня вартість витрат на експлуатацію проектного рішення (за весь час його експлуатації), грн.:

$$Ц_П = K * \left(1 + \frac{П_Р}{100}\right) * \left(1 + C_{ПДВ}\right) + K_0 + K_k \quad (6.14)$$

де $П_р$ - норматив рентабельності (приймаємо 30%); K_0 - витрати на прив'язку та освоєння проектного рішення на конкретному об'єкті, грн.,

K_0= 300 грн.;

$K_к$ - витрати на доукомплектування технічних засобів на об'єкті, грн.,

$K_к$ = 0 грн.;

$C_{ПДВ}$ - ставка податку на додану вартість (20 %).

$Ц_П$ = 26537,97 · (l + 30/100) · (l + 0,2) + 300 + 0 = 41699,23грн.

Ціна аналогу у переводі з доллара дорівнює:

$Ц_{П(А)}$ = 75400,00 грн.

Теперішня вартість витрат на експлуатацію проектного рішення розраховується за формулою:

$$B_{(E)NPV} = \sum_{t=1}^{T} \frac{B(E)nt}{(1+R)^t} \quad (6.15)$$

де $B_{(E)Пt}$ - річні експлуатаційні витрати в t-ому році, грн.; T- строк служби проектного рішення, 5 років; R - річна ставка проценту банків (30%).

$$B_{(E)NPV} = 106244,6 / (1 + 0,3)^1 + 106244,6 / (1 + 0,3)^2 + 106244,6 / /(1+ 0,3)^3 + 106244,6/ (1+ 0,3)^4 + 106244,6/(1+ 0,3)^5 = 258766,9 \text{грн.}$$

Таким чином ціна споживання проектного рішення становить:

$$Ц_{С(П)} = 41699,23 + 258766,9 = 300466,13 \text{ грн.}$$

Аналогічно визначається ціна споживання для аналогу. Визначимо теперішню вартість витрат на експлуатацію аналогу. Термін експлуатації аналогу становить 5 років, тоді за формулою 5.15:

$$B_{(E)NPV} = 145141/ (1 + 0,3)^1 + 145141/ (1 + 0,3)^2 + 145141/ (1 + +0,3)^3 + 145141/ (1 + 0,3)^4 + 145141/ (1+ 0,3)^5 = 353501.394 \text{грн.}$$

Таким чином ціна споживання аналогу становить:

$$Ц_{С(А)} = 75400,00 + 353501.394 = 428901,39 \text{грн.}$$

6.6. Визначення показників економічної ефективності

1) Показник конкурентоспроможності:

$$K_{кс} = \frac{Ц_{С(П)} * П_{Я}}{Ц_{С(А)}} \quad (6.16)$$

$$K_{КС} = 300466,13 \cdot 2,041 / 428901,39 = 1,43$$

2) Економічний ефект в сфері експлуатації:

$$Е_{ЕКС} = В_{(Е)А} - В_{(Е)П} \quad (6.17)$$

$$E_{ЕКС} = 145141 – 106244,6 = 38896,4 \text{ грн.}$$

3) Економічний ефект в сфері проектування:

$$E_{ПР} = Ц_{А} - Ц_{П} \quad (6.18)$$

$$E_{ПР} = 75400,00 - 41699,23 = 33700,77 \text{ грн.}$$

Оскільки $Еекс > 0$ та $E_{ПР} > 0$ то розраховуємо:

- додатковий економічний ефект в сфері експлуатації:

$$Е_{ЕКС\,Д} = \sum_{i=1}^{Т} Е_{ЕКС} * (1 + R)^{Т-i} \quad (6.19)$$

$$E_{ЕКС\,Д} = 38896,4 \cdot (1,3^0 + 1,3^1 + 1,3^2 + 1,3^3) = 205761.96 \text{грн.}$$

- додатковий економічний ефект в сфері проектування:

$$Enp_{Д} = Enp \times (1 + R)^{T} \quad (6.20)$$

$$Е_{ПР\,Д} = 33700,77 \cdot 3,71 = 125029,86$$

- термін окупності витрат на проектування рішення:

$$T_{ОК} = \frac{К}{Е_{еск}} \quad (6.21)$$

$Т_{ОК}$ = *26537,97*/ 38896,4 = 0,7 року або 9 місяців.

Результуючі показники економічної ефективності зводяться у (табл. 6.7).

Таблиця 6.7 Показники економічної ефективності проектного рішення

Найменування показників	Одиниці вимірювання	Значення показників	
		Аналог	Проектне
1 Капітальні вкладення	грн.	-	26537,97
2 Ціна придбання	грн.	75400,00	41699,23
3 Річні експлуатаційні витрати	грн.	145141	106244,6
4 Ціна споживання	грн.	428901,39	300466,13
5 Економічний ефект в сфері експлуатації	грн.	-	*38896,4*
6 Додатковий економічний ефект в сфері експлуатації	грн.	-	*205761.96*
7 Економічний ефект в сфері проектування	грн.	-	*33700,77*
8 Додатковий економічний ефект в сфері проектування	грн.	-	*125029,86*
9 Термін окупності витрат на	місяці	-	9
10 Коефіцієнт конкурентоспроможності	-	-	*1,43*

6.7. Висновки економічної ефективності

Таким чином, загальна сума витрат на розробку проектного рішення склала 26 537,97 грн. Проведені розрахунки підтверджують, що використане

в даній кваліфікаційній роботі програмне середовище є значно вигіднішим у порівнянні з аналогами за економічними показниками. Зокрема, помітних результатів було досягнуто у сферах експлуатації (38 896,4 грн.) та проектування (33 700,77 грн.). Ці дані свідчать про економічну доцільність і раціональність розробки такої системи.

Однією з ключових переваг розробленого програмного середовища є його висока якісна оцінка, яка свідчить про надійність, ефективність та відповідність сучасним вимогам у сфері проектування та впровадження програмних рішень. Крім того, позитивним фактором є короткий термін окупності витрат на проектування, який становить лише 9 місяців. Це демонструє швидке повернення інвестицій, що є важливим критерієм для будь-якого комерційного проекту.

Додатковою перевагою розробленого середовища є його вузькоспеціалізований характер. У порівнянні з універсальними аналогами, таке середовище спрямоване на вирішення конкретних задач, що дозволяє значно знизити його вартість. Завдяки цьому програмне забезпечення стає доступним для широкого кола користувачів, включаючи невеликі фірми-провайдери. Це відкриває можливість для впровадження інновацій навіть у компаніях із обмеженим бюджетом, сприяючи їхньому розвитку та конкурентоспроможності.

Таким чином, розроблене програмне середовище не лише відповідає економічним та якісним критеріям, але й демонструє ефективність у сфері практичного застосування. Воно є прикладом рішення, яке поєднує функціональність, доступність і економічну вигоду, що робить його особливо привабливим для малого та середнього бізнесу.

ВИСНОВКИ

Магістерська робота зосереджена на актуальній темі вдосконалення систем роботи з нечіткою логікою через впровадження сучасних web-технологій, які забезпечують підвищення ефективності та зручності використання таких систем.

1. Проведено огляд літератури, який підтвердив актуальність використання web-ужитків для налаштування контролерів нечіткої логіки, визначивши цю задачу як важливу науково-практичну проблему.
2. Виконано системний аналіз, у межах якого розглянуто сучасні алгоритми та засоби побудови контролерів нечіткої логіки, а також обрано оптимальні технології для розробки web-ужитків.
3. Розроблено концептуальну модель, яка відображає структурну побудову об’єкта дослідження та визначає шляхи взаємодії між компонентами системи та зовнішнім середовищем.
4. Проведено економічну оцінку проектного рішення, яка показала, що розроблений web-ужиток перевершує аналоги за економічними показниками.
5. Створено алгоритм, який забезпечує паралельний доступ до сервера для 100 користувачів.
6. Адаптовано алгоритм нечіткого логічного виведення для масового використання спеціалістами широкого профілю.
7. Розроблено інтуїтивно зрозумілий користувацький інтерфейс для налаштування контролерів нечіткої логіки.
8. Реалізовано web-технологію, яка забезпечує швидкий доступ до контролерів та їхню роботу в режимі реального часу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Бучек Г. ASP.NET. Навчальний курс. – СПб.: Київ, 2002. 512с.
2. Гери Д., Хорстманн К. JavaServer Faces – К.: Наукова думка, 2008. 576с.
3. Мальтцер Кевін, Михальски Брент. Розробка CGI-додатків на Perl. – К.: Видавничий дім., 2001. 400с.
4. Котеров Д.В. PHP 4. – СП.: БХВ-Донецьк, 2004. 576с.
5. Кузнець М., Симдянов И. MySQL 5. СП.: БХВ-Донецьк, 2010. 1024с.
6. Jason Price, Oracle Database 11g SQL. McGraw-Hill Osborne Media, 2007. 656с.
7. Трухаїв Р.І. Моделі прийняття рішень в умовах невизначеності. К.: Наука, 1981. 258 с.
8. Орловський С. А. Проблеми прийняття решень при нечеткій вихідній інформації. К.: Наука, 1981. 208 с.
9. Алтунін А.Е. Моделі та алгоритми прийняття рішень в нечітких умовах: Монографія. Донецьк, ДДУ, 2000. 352 с.
10. Zadaeh L. A.. Fuzzy Sets, Information and kontrol. 1965. 8. P. 338 – 353.
11. Леонець А. В. Нечітке моделювання в MATLAB и fuzzyTECH. БХВ-Донецьк, 2005. 736 с.
12. Головчук А.Ф. Електронна система паливоподачі тракторного дизеля. Вісник НТУ. 2012. №25 72с.
13. Блюмін С.Л., Шуйкова І.А., Сараев П.В. Нечітка логіка: алгебраїчні основи та додатки: Монографія. Донецьк, ДДУ, 2002. 113 с.
14. Асаї К. Прикладні нечіткі системи [пер. з японської] / Під ред. Т. Терано, К. Асаї, М.Сугено. К.: Вища школа, 1993. 368 с.

15. Додонов С. Б., Квачов В. Г., Петрушенко Л. А. Система обробки нечіткої та неповної информації для підтримання прийняття решень. Управляючі системи і машини. 2001. № 1. С. 90-92.

16. Смирнов В. С. Синтез електромеханічних систем з використанням нечіткої логіки та аналіз їх стійкості. Вісн. Вінниц. політехн. ін-ту. 1999. № 1. С. 52-60.

17. Лозинський А. О. Формування керуючих впливів електромеханічних систем на основі принципів нечіткої логіки. Радіоелектроніка. Інформатика. Управління. 2000. № 1. С. 156-161.

18. Кирик В. В., Костерєв М. В., Лукаш М. П., Бардик Є. І. Нечітка логіка в управлінні та експлуатації об'єктів електроенергетики. НАН України. Ін-т електродинаміки. Нац. техн. ун-т України "КПІ". К., 2007. 76 с.

19. Хандецький В. С., Редько В. І., Новак П. Я., Пастушкін Т. В. Нечітка логіка. Навч. посіб. для студ. вищ. навч. закл. Дніпропетр. нац. ун-т. Д., 2005. 230 с.

20. Kosko B. Fuzzy engineering. Prentice Hall, 1996. 549 p.

21. MacVicar-Whelan P.J. Fuzzy sets for man machine interaction. Int. J. Man-Mach. Stud. 1976. N 8. P. 687–697.

22. Tsukamoto Y. Advances in fuzzy set theory and application. North-Holland, 1979.

23. Yager R.R. Essentials of fuzzy modeling and control. New York: John Wiley & Sons, 1994.