Київський національний університет імені Тараса Шевченка
Міністерство освіти і науки України

Київський національний університет імені Тараса Шевченка
Міністерство освіти і науки України

Кваліфікаційна наукова

праця на правах рукопису

ІЛЬКУН ОЛЕКСАНДР ПЕТРОВИЧ

УДК 004.912

ДИСЕРТАЦІЯ

СИСТЕМА ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ В НЕЧІТКИХ УМОВАХ

121 Інженерія програмного забезпечення

12 Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ О. П. Ількун

Науковий керівник: Провотар Олександр Іванович, доктор фізико-математичних наук, професор

Київ – 2023

АНОТАЦІЯ

Ількун О.П. Система підтримки прийняття рішень в нечітких умовах – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії з галузі знань 12 Інформаційні технології за спеціальністю 121 Інженерія програмного забезпечення – факультет комп'ютерних наук та кібернетики, Київський національний університет імені Тараса Шевченка, Київ, 2023.

Дисертаційна робота присвячена розробці методів та алгоритмів побудови системи підтримки прийняття рішень на основі аналізу нечітких знань. Зокрема, йдеться про діагностичну систему для аналізу стану здоров'я пацієнта та формування первинного діагнозу на основі експертних нечітких правил і поточних симптомів.

У першому розділі дисертації акцентується увага на розгляді систем прийняття рішень та їх основних компонентів. Описані ключові етапи та діючі сутності таких систем. Особлива увага приділена деталізації проблеми дослідження та контекстуальному розгляду предметної області – акцентовано на представленні основних даних цієї області, особливостях сутностей системи та результатах її роботи. Розділ також містить аналіз систем управління інформацією, включаючи структуру системи підтримки прийняття рішень та структуру експертної системи, крім того, додатково розглянуто особливості застосування в таких системах елементів нечіткої логіки та нечіткого виведення. Проведено аналіз та надана класифікація знань за їх основними характеристиками. Детально розглянуто основні етапи життєвого циклу та придбання знань для систем підтримки прийняття рішень, зокрема, діючі ролі та моделі придбання знань. Також у розділі здійснено огляд існуючих рішень у даній області з точки зору їх можливостей, обмежень та областей застосування.

У другому розділі дослідження увага акцентується на використанні нечітких моделей для діагностування стану здоров'я пацієнта. Розглядаються основні поняття теорії нечітких множин, основні операцій на нечітких множинах, нечіткі відношення та системи нечіткого логічного виведення. Основну увагу приділено алгоритмам

знаходження нечіткої моделі хвороби та процесу формування нечітких правил системи, що дозволяє автоматизувати процес діагностування і підвищити його точність та надійність. Цей розділ є ключовим для розуміння механізмів моделювання та аналізу нечітких знань, що в свою чергу, є важливим для розробки ефективних систем діагностики та підтримки прийняття рішень в медичній галузі. Особливу увагу в розділі приділено детальному вивченню та аналізу методів нечіткого моделювання, які є основою для розробки сучасних систем медичної діагностики.

У третьому розділі основна увага приділена методам знаходження оцінок достовірності при нечіткому моделюванні. Обговорюється процес побудови систем нечітких правил та знаходження їх виходу на предмет правильного (адекватного, достовірного) визначення діагнозу. В розділі розглядаються елементи теорії ймовірностей нечітких подій та алгоритм знаходження достовірності виходу системи нечітких правил для прийняття рішень в умовах нечіткості та невизначеності. Наводяться приклади для скінченних просторів, де конкретизуються методи та процедури обчислення виходів систем нечітких правил для реальних систем діагностування. Усі розглянуті методи та інструменти дозволяють підвищувати точність і надійність медичних діагностичних систем та є фундаментом для подальших наукових досліджень і розробок в даній галузі.

У четвертому розділі роботи деталізовано програмні рішення для системи, що створюється. Розглянуті різні архітектури програмного забезпечення та специфікації сучасних десктопних програм та вебсистем. Розглянуті всі рівні системи, включаючи інфраструктурний рівень, рівень зберігання даних, презентаційний рівень, рівень бізнес-логіки та рівень моніторингу, що всебічно описують структуру та функціонування системи. Подано деталізацію процесу моделювання рівня зберігання даних відповідно до предметної області, що дозволяє оптимально використовувати потужності систем зберігання даних. Розглянуто застосування неперервної інтеграції та її основних компонентів в життєвому циклі сучасних систем підтримки прийняття рішень. Наведено приклади роботи системи підтримки прийняття рішень, які ілюструють практичну реалізацію розроблених методів і технологій. Висвітлено як технічні аспекти розробки, так і методологічні принципи, що лежать в її основі.

Робота завершується висновками (підсумовуються основні наукові та практичні результати, представлені в дослідженні) та оцінкою важливості систем, заснованих на знаннях у сфері медичної діагностики.

Ключові слова: системи підтримки прийняття рішень, експертні системи, нечітка логіка, системи нечіткого виведення, ймовірність нечітких подій, архітектура вебзастосунків, неперервна інтеграція, моделювання баз даних.

ABSTRACT

Ilkun O.P. Decision support system in fuzzy conditions – Qualification scientific work with manuscript rights.

Dissertation for obtaining the scientific degree of Doctor of Philosophy in the field of knowledge 12 Information technologies in the specialty 121 Software engineering – Faculty of Computer Sciences and Cybernetics, Taras Shevchenko National University of Kyiv, Kyiv, 2023.

The dissertation is focused on the research and creation of methods for developing advanced decision support systems using fuzzy logic, in particular, a diagnostic system for analyzing the patient's health status and forming a primary diagnosis based on expert fuzzy rules and current symptoms.

The first chapter of the dissertation focuses on the consideration of decision-making systems and their main components. The key stages and active entities of such systems are described. Particular attention is paid to the detailing of the research problem and the contextual examination of the subject area – the emphasis is on the presentation of the main data of this area, the features of the system entities and the results of its work. The chapter also contains an analysis of information management systems, including the structure of the decision support system and the structure of the expert system, in addition, the features of the application of elements of fuzzy logic and fuzzy inference in such systems are additionally considered. The analysis was carried out and the classification of knowledge according to its main characteristics was provided. The main stages of the life cycle and knowledge acquisition for decision support systems are considered in detail, in particular, the current roles and models of knowledge acquisition. Also, the section reviews existing solutions in this area from the point of view of their capabilities, limitations, and areas of application.

The second chapter of the study focuses on the use of fuzzy models to diagnose the patient's health status. The basic concepts of the theory of fuzzy sets, basic operations on fuzzy sets, fuzzy relations and fuzzy inference systems are considered. The main attention is paid to algorithms for finding a fuzzy model of the disease and the process of forming

fuzzy rules of the system, which allows to automate the diagnosis process and increase its accuracy and reliability. This section is key to understanding the mechanisms of fuzzy knowledge modeling and analysis, which in turn is important for developing effective diagnostic and decision support systems in the medical field. Special attention is paid in the section to the detailed study and analysis of fuzzy modeling methods, which are the basis for the development of modern medical diagnostic systems.

In the third chapter, the main attention is paid to the methods of finding reliability estimates for fuzzy modeling. The process of building systems of fuzzy rules and finding their solution to the subject of a correct (adequate, reliable) diagnosis is discussed. The section examines the elements of the theory of probabilities of fuzzy events and the algorithm for finding the reliability of the output of the system of fuzzy rules for decision-making under conditions of vagueness and uncertainty. Examples are given for finite spaces, where methods and procedures for calculating outputs of fuzzy rule systems for real diagnostic systems are specified. All the considered methods and tools make it possible to increase the accuracy and reliability of medical diagnostic systems and are the foundation for further scientific research and development in this field.

The fourth chapter of the work details software solutions for the system being created. Different software architectures and specifications of modern desktop programs and web systems are considered. All levels of the system are considered, including the infrastructure, data storage, presentation, business logic, and monitoring levels, which comprehensively describe the structure and functioning of the system. The details of the process of modeling the level of data storage according to the subject area are provided, which allows optimal use of the capacities of data storage systems. In addition, the application of continuous integration and its main components in the life cycle of modern decision support systems is considered. Examples of the operation of the decision support system are given, which illustrate the practical implementation of the developed methods and technologies. Both the technical aspects of the development and the methodological principles underlying it are highlighted.

The dissertation ends with conclusions (summarizing the main scientific and practical results presented in the study) and an assessment of the importance of knowledge-based systems in the field of medical diagnostics.

Keywords: decision support systems, expert systems, fuzzy logic, fuzzy inference systems, probability of fuzzy events, web application architecture, continuous integration, database modeling.

Список праць здобувача за темою дисертації

Статті у наукових фахових виданнях України:

1. Провотар О.І., Ількун О.П. Використання категорій як абстрактних структур для формалізації моделей обчислень // Зв'язок. – 2022. – № 5. С.46-49.
2. Ількун О.П. Вебзастосунки як один із сучасних способів реалізації систем підтримки прийняття рішень // Вісник Київського національного університету імені Тараса Шевченка. Серія: фізико-математичні науки. – 2023. – Випуск №1. С.94-100.
3. Провотар О.І., Ількун О.П. Про один підхід до знаходження оцінок достовірності при нечіткому моделюванні // Кібернетика та Системний Аналіз. – 2023. – Том 59, № 6. С.30-39.

Праці, які засвідчують апробацію матеріалів дисертації:

4. Провотар О.І., Ількун О.П., Провотар Т.М. Достовірність нечітких рішень в задачах розпізнавання // Матеріали міжнародного наукового симпозіуму «Інтелектуальні рішення». м. Ужгород, Україна, 2021. С. 66.
5. Ількун О.П. Оптимізація коефіцієнта конверсії вебсистеми через пошук найвпливовішого за швидкодією компонента // Матеріали XXI Міжнародної науково-практичної конференції «Шевченківська весна – 2023», Секція «Прикладна математика, комп'ютерні науки, інженерія програмного забезпечення, системний аналіз». м. Київ, Україна, 2023. С.82-83.
6. Провотар О.І., Ількун О.П. Категорії як формальні моделі обчислень // Міжнародний науковий симпозіум «Питання оптимізації обчислень (ПОО-XLVIII)», Фізико-математичне моделювання та інформаційні технології (37). м. Львів, Україна, 2023. С.98-102.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

СППР – система підтримки прийняття рішень

ЕС – експертна система

БЗ – база знань

СКБД – система керування базами даних

VPN – віртуальна приватна мережа (virtual private network)

DNS – система доменних імен (domain name system)

WAF – брандмауер вебзастосунків (web application firewall)

CDN – мережа розповсюдження контенту (content delivery network)

XSS – міжсайтовий скриптинг (cross site scripting)

IAAS – інфраструктура як послуга (infrastructure as a service)

API – прикладний програмний інтерфейс (application programming interface)

ВСТУП

Дисертаційне дослідження присвячене розробці методів та алгоритмів побудови системи підтримки прийняття рішень на основі аналізу та обробки нечітких знань.

Основна увага в дослідженні акцентується на розробці діагностичної системи, яка аналізує стан здоров'я пацієнта на основі його симптомів і формує початковий діагноз, використовуючи експертні нечіткі правила.

Обґрунтування вибору теми дослідження. У епоху глобалізації, коли інформація виступає ключовим ресурсом сучасного суспільства, системи підтримки прийняття рішень набувають вирішального значення, відіграючи важливу роль у багатьох сферах діяльності [1]. Сучасні системи підтримки прийняття рішень, що використовують машинне навчання, статистичний аналіз, оптимізацію, нечіткі логіки та інші методи, можуть ефективно обробляти величезні обсяги даних, сприяючи розумінню та оптимізації бізнес-процесів, а також підвищенню ефективності управління.

Спектр застосування таких систем включає бізнес, медицину, освіту, науку, інженерію, а також структури державного управління. Особливий інтерес представляє використання систем підтримки прийняття рішень у медицині, зокрема в процесах діагностики захворювань. Це можна пояснити великим обсягом нечіткої інформації, що виникає під час медичного огляду пацієнтів, де неоднозначність та розмитість симптомів створює значні труднощі для встановлення діагнозу пацієнта.

Варто зазначити, що специфіка людського мислення полягає в здатності аналізувати неоднозначність, неповноту та нечіткість інформації. Це, в свою чергу, ставить перед наукою завдання побудови та реалізації нечітких моделей подання знань в контексті розробки та впровадження інноваційних комп'ютерних систем.

Одним з важливих інструментів, що допомагає працювати з нечіткою інформацією є теорія нечітких множин і нечітка логіка, запропонована американським вченим Лотфі Заде в 1965 році [2]. Ці теорії представляють узагальнення класичної теорії множин і формальної логіки і є інструментом для обробки нечіткої, неоднозначної та невизначеної інформації.

Потреба в нечіткому моделюванні виникає в багатьох реальних задачах, розв'язання яких за допомогою точних методів є неможливим або недостатньо ефективним. Відповідні методи, засновані на різних способах представлення нечіткої інформації, стають важливою складовою розробки високоефективних та високотехнологічних систем.

В рамках даної роботи розглядаються сучасні підходи до розробки систем діагностики захворювань з використанням нечіткої логіки. Важливість дослідження систем підтримки прийняття рішень при діагностиці захворювань зумовлена їх роллю у підвищені ефективності діяльності організацій охорони здоров’я.

Сучасні системи підтримки прийняття рішень в медицині допомагають лікарям приймати ефективні та своєчасні рішення, засновані на точних та актуальних даних. Враховуючи різноманітність можливих симптомів та станів пацієнтів, системи підтримки прийняття рішень можуть значно покращити процес діагностики, шляхом формування індивідуальних планів лікування та профілактичних заходів. Для ефективного використання цих систем необхідно розробити архітектуру, що відповідає вимогам великих обсягів даних, швидкості обробки, складності моделей та нечіткості даних.

Крім того, величезні обсяги даних, що створюються у медичній галузі, поставили перед сучасними системами підтримки прийняття рішень новий виклик – розробка та впровадження ефективних методів аналізу даних. Використання таких методів, як нечітка логіка, дає можливість відкрити нові горизонти в діагностуванні та лікуванні різних захворювань, але також вимагає глибокого розуміння цих технологій та їх можливостей.

Також важливим є врахування контексту в діагностиці. Системи підтримки прийняття рішень повинні бути здатні обробляти не тільки клінічні дані, але і контекстуальну інформацію про пацієнтів, таку як їхній соціальний, психологічний і поведінковий контекст, що може впливати на їхній стан здоров’я. Це потребує розробки нових методів для інтеграції та аналізу таких даних.

Загалом, використання нечіткої логіки в системах підтримки прийняття рішень в медицині має величезний потенціал. Однак, для досягнення цього потенціалу

потрібно подальше дослідження та розробка нових методів, які пов’язані з обробкою великих обсягів даних, інтерпретацією результатів та інтеграцією контекстуальної інформації.

Також слід врахувати, що ефективність систем підтримки прийняття рішень значною мірою залежить від їхньої інтеграції в існуючі бізнес-процеси та їхньої спроможності адаптуватися до змін умов та вимог. Це стосується не лише технічної сторони питання, але і організаційних аспектів використання цих систем, включаючи питання безпеки даних, конфіденційності та етики.

Враховуючи все вищезгадане, можна з впевненістю зазначити, що інтелектуальні системи підтримки прийняття рішень на основі нечіткої логіки та машинного навчання мають великий потенціал для трансформації медичної галузі. Вони відкривають нові можливості для ефективної діагностики, індивідуалізованого лікування та попередження хвороби. Однак, щоб повною мірою використати ці переваги, необхідно подальше дослідження та вдосконалення цих систем, а також забезпечення їх інтеграції в існуючі медичні процеси. Це вимагає поєднання зусиль науковців, медичних фахівців, інженерів, а також організацій, які втілюють ці системи в життя. Такий інтегрований підхід дозволить створити більш ефективну, справедливу та інноваційну медицину майбутнього.

Зв’язок роботи з науковими програмами, планами, темами. Дисертаційна робота виконана в рамках науково-дослідної роботи № 16КФ015-01 «Моделі та технології інтелектуальних обчислень» кафедри інтелектуальних програмних систем факультету комп’ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка, яка виконувалася у межах кафедральної тематики.

Мета і завдання дослідження. Мета роботи полягає у створенні сучасної системи підтримки прийняття рішень (з використанням нечіткої логіки), яка допомагає проаналізувати поточну симптоматику хворого і надати попередній діагноз на підставі експертних нечітких знань.

Об’єкт дослідження – методи побудови систем підтримки прийняття рішень.

Предмет дослідження – система підтримки прийняття рішень для обробки і аналізу нечітких знань при діагностиці стану здоров'я пацієнтів.

Для досягнення мети пропонується здійснити комплексний підхід, що включає в себе кілька ключових завдань, наведених нижче.

- Першим кроком є детальний огляд та аналіз основ теорії нечітких множин. Це передбачає глибоке вивчення концепцій, математичних моделей та методології, що лежать в основі нечіткої логіки, з особливим акцентом на теоретичні засади та практичне використання.
- Далі, передбачається проведення огляду та аналізу існуючих експертно-діагностичних систем. Це включає дослідження структури, архітектури та функціональності цих систем, з метою зрозуміння їх потенціалу, а також обмежень.
- Третім завданням є імплементація алгоритму нечіткого логічного виведення. Зокрема, це включає розробку та кодування специфічного алгоритму, що використовує принципи нечіткої логіки для знаходження висновків нечітких правил на основі набору вхідних даних.
- Завершальним завданням є побудова функціональної експертно-діагностичної системи. Це передбачає розробку всіх необхідних компонентів, включаючи інтерфейс користувача, систему управління даними і модулі для обробки та аналітики даних. Крім того, система повинна бути запроектована з урахуванням принципів гнучкості та масштабованості, з тим щоб адаптуватися до змінюваних вимог та різних сценаріїв використання.

Розглядаючи ці завдання, важливо зазначити, що вони вимагають інтегрованого підходу, що поєднує глибокі теоретичні знання, аналітичні навички та досвід в області розробки програмного забезпечення.

Гіпотеза дослідження. Сучасний рівень прогресу в області обчислювальних технологій та інформаційних систем відкриває нові горизонти для створення методів та моделей для побудови систем управління інформацією в медицині. Такі системи

можуть сприяти не лише оптимізації медичних процесів та підвищенню ефективності медичного обслуговування, але і надавати можливість обробки значних обсягів даних, інтегрованих із різноманітних джерел, забезпечуючи високий ступінь гнучкості, точності та надійності. Це, в свою чергу, дозволить мінімізувати ризик помилок, зумовлених людським фактором, та підвищить стандарти медичної допомоги, роблячи її загалом більш точною, доступною та вчасною.

Методи дослідження. Метод нечіткого логічного виведення Мамдані; метод нечіткого логічного виведення Сугено; дефазифікація методом центру тяжіння; дефазифікація методом центру площі; гнучкі методи розробки програмних систем; методи моделювання програмних систем; діаграми об’єктно-орієнтованого аналізу і проектування; багаторівнева архітектура програмних вебсистем; принципи неперервної інтеграції та неперервної доставки; автоматичне масштабування ресурсів; модульне тестування; функціональне тестування для перевірки відповідності вимогам; інтеграційне тестування для перевірки взаємодії компонентів системи; навантажувальне тестування для визначення продуктивності системи; тестування безпеки для ідентифікації можливих загроз.

Наукова новизна отриманих результатів полягає у створенні та дослідженні методів розробки сучасних систем підтримки прийняття рішень з використанням нечіткої логіки та базується на таких положеннях:

вперше:

- Запропоновано нечіткі імплікації для моделювання процесу діагностування в гомеопатії.
- Запропоновано та реалізовано алгоритм знаходження нечіткого діагнозу пацієнта.
- Запропоновано та реалізовано алгоритм знаходження достовірності виходу – ймовірнісної характеристики діагнозу пацієнта.
- Запропоновано та реалізовано універсальну архітектуру програмного забезпечення системи підтримки прийняття рішень для веборієнтованого середовища.

удосконалено:

- Алгоритм Мамдані знаходження виходу системи нечіткого логічного виведення.
- Метод наповнення та актуалізації бази знань.

Особистий внесок здобувача. В опублікованій у співавторстві статті [14] здобувачу належить опис операцій нечіткої логіки, які описуються в рамках абстрактних структур – категорій, які, в свою чергу, можуть бути основою для розробки новітніх методів та технік створення сучасних систем підтримки прийняття рішень, в тому числі, на основі нечіткої логіки.

В статті [22], опублікованої у співавторстві, здобувачу належить розробка алгоритму для визначення числових оцінок достовірності нечітких знань у системах нечіткого логічного виведення.

Результати з розробки та впровадження універсальної архітектури програмного забезпечення системи підтримки прийняття рішень для вебсередовища були опубліковані здобувачем в [28].

Основні наукові положення, висновки та результати дослідження опубліковано у 3 наукових статтях фахових видань України.

Апробація результатів дослідження. Результати дослідження доповідалися на:

- Міжнародний науковий симпозіум «Інтелектуальні рішення» (28-30 вересня 2021 року).
- XXI Міжнародна науково-практична конференція «Шевченківська весна – 2023: Математика, статистика, механіка. Прикладна математика, комп'ютерні науки, інженерія програмного забезпечення, системний аналіз. Методика викладання математики» (14 квітня 2023 року).
- Міжнародний науковий симпозіум «Питання оптимізації обчислень (ПОО-XLVIII)» (19-22 вересня 2023 року).

Структура та обсяг дисертації. Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, додатків. Кожен розділ розбито на підрозділи, які, в

свою чергу, поділяються на пункти. Розділи мають власну нумерацію рисунків, таблиць. Обсяг роботи – 143 сторінки.

Практичне значення отриманих результатів. Системи підтримки прийняття рішень відіграють важливу роль в різних областях людської діяльності. Одна з ключових характеристик цих систем полягає у їхній здатності ефективно обробляти великі обсяги інформації за допомогою широкого діапазону методологічних технік. Ці техніки включають статистичний аналіз, аналіз даних, машинне навчання, оптимізацію, нечіткі множини та багато інших.

У світлі вищенаведеного, проблема розробки високоефективних систем прийняття рішень, які відповідали б сучасним потребам, є особливо актуальною. У цій роботі було розглянуто сучасні методи розробки програмного забезпечення та запропоновано архітектуру для реалізації систем підтримки прийняття рішень, яка може бути адаптована до широкого спектра задач. Було продемонстровано застосування цієї архітектури на прикладі конкретної системи підтримки прийняття рішень, розробленої для діагностики станів пацієнтів.

При виконанні роботи результати теоретичного дослідження систем нечіткого логічного виведення були впроваджені у практичну сферу шляхом побудови системи підтримки прийняття рішень. Ці дослідження не тільки підкреслюють значимість нечіткої логіки в сучасних медичних дослідженнях, але й відкривають нові горизонти для подальшого вивчення та розуміння діагностичних процесів.

Впровадження розробленої системи дозволить осучаснити сферу медичного обслуговування, зокрема, шляхом зниження рівня залежності від професійної компетентності медичних працівників при виборі лікування для пацієнтів. Крім того, це дозволить надавати допомогу пацієнтам в місцях, де медичний персонал може мати обмежені ресурси або недостатню підготовку. Ще однією важливою перевагою цієї системи є можливість усунення фактору людської помилки у процесі діагностики.

РОЗДІЛ 1: ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ І СИСТЕМ УПРАВЛІННЯ ІНФОРМАЦІЇ ДЛЯ ПРИЙНЯТТЯ РІШЕНЬ

1.1 Основні етапи і сутності системи прийняття рішень

1.1.1 Основні етапи

Робота системи підтримки прийняття рішень, яка автоматизує процес діагностики стану здоров'я пацієнта, складається з декількох основних етапів:

1. Збір даних. Збір всієї релевантної інформації про пацієнта. Це може включати медичну історію, симптоми, результати обстежень та тестів, рівень фізичної активності, дієтичні відомості, генетичні дані тощо.

2. Обробка та аналіз даних. Обробка даних, яка полягає у їх перетворенні для подальшого аналізу. Застосування методів інтелектуального аналізу даних, нечіткої логіки та інших методів дозволяє виявляти закономірності, тенденції та зв'язки в даних.

3. Моделювання та прогнозування. На основі аналізу даних створюються моделі, які допомагають прогнозувати можливі медичні стани або хвороби. Це може включати класифікацію пацієнтів за ризиком захворювань, прогнозування перебігу хвороби та визначення найефективніших методів лікування.

4. Прийняття рішення. На основі прогнозувань система рекомендує найкращі можливі дії або рішення. Це може бути визначення діагнозу, вибір лікування, рекомендації щодо зміни способу життя.

5. Постійне навчання та адаптація. На останньому етапі система постійно вивчає нові дані, адаптується до змін у стані пацієнта та покращує свої моделі та прогнози. Це дозволяє системі ставати все більш точною і надійною з часом.

Важливо пам'ятати, що така система не заміняє лікаря, а слугує інструментом для підтримки процесу прийняття рішень, забезпечуючи їх детальними, точними та своєчасними даними.

1.1.2 Основні діючі особи системи

До основних діючих осіб системи можна віднести лікаря та пацієнта. Сформуємо детальний опис функціональних можливостей кожної з діючих осіб і опишемо цей функціонал у вигляді історій користувача (User story) [3].

Високорівневі вимоги до функціоналу системи від лікаря можна описати наступним чином – лікар повинен мати змогу використовувати систему підтримки прийняття рішень для аналізу даних про стан здоров'я пацієнтів, щоб швидше та точніше ставити діагнози та розробляти плани лікування.

Більш детально ці вимоги можна описати наступними історіями користувача:

Історія користувача 1

Як лікар, я хочу мати можливість реєструватися в системі підтримки прийняття рішень та отримувати доступ до організаційних можливостей, щоб мені було зручно використовувати систему в своїй щоденній роботі.

Критерії приймання

- Я, як лікар, повинен мати можливість зареєструватися в системі, використовуючи мою професійну електронну адресу та іншу необхідну інформацію.
- Система повинна перевірити мою інформацію і надати мені доступ до системи, після чого я можу створити особистий пароль для безпечного входу.
- Система повинна надавати доступ одразу після реєстрації в системі, я повинен мати доступ до основних організаційних можливостей, включаючи створення та управління профілями пацієнтів, доступ до медичних даних пацієнтів.
- Система повинна допомагати мені у навігації та використанні різних функцій, надаючи корисні поради та підказки.

Історія користувача 2

Як лікар, я хочу, щоб система підтримки прийняття рішень могла автоматично проаналізувати медичні дані пацієнта та допомогти в діагностиці, щоб я міг більш точно та ефективно визначити стан здоров'я пацієнта.

Критерії приймання

- Я, як лікар, можу ввести медичні дані пацієнта в систему, включаючи симптоми, історію хвороб, результати обстежень та тестів.
- Система автоматично аналізує ці дані, використовуючи алгоритми аналізу даних, для визначення потенційного діагнозу.
- Система надає мені список потенційних діагнозів, впорядкований за ймовірністю, з посиланнями на відповідну медичну літературу для подальшого огляду.
- Я, як лікар, можу переглянути ці діагнози, порівняти їх з моїми власними спостереженнями та вирішити, який діагноз найбільш відповідає стану здоров'я пацієнта.

Історія користувача 3

Як лікар, я хочу мати можливість отримувати попередження від системи підтримки прийняття рішень, якщо стан здоров'я пацієнта погіршується або, якщо виявлені симптоми вимагають термінового втручання, щоб я міг вчасно відреагувати та надати необхідну допомогу.

Критерії приймання

- Я, як лікар, можу налаштувати систему на відправку сповіщень про критичні зміни в стані здоров'я пацієнтів.
- Я, як лікар, отримую автоматичні попередження про випадки, які вимагають термінової уваги.
- Я, як лікар, можу використовувати інформацію з цих попереджень для того, щоб швидко розробити план втручання.

Історія користувача 4

Як лікар, я хочу мати змогу навчати систему на основі моїх власних випадків і досвіду, щоб вона ставала все більш точною та корисною в моїй роботі.

Критерії приймання

- Я, як лікар, можу вводити в систему деталі випадків, над якими я працював, включаючи діагнози та методи лікування, які були застосовані.

- Я, як лікар, можу давати системі зворотний зв’язок щодо її рекомендацій, що допомагає їй навчитися і покращити свої прогнози в майбутньому.
- Я, як лікар, бачу, як мій вклад в систему допомагає підвищити її ефективність та точність.

Історія користувача 5

Як лікар, я хочу мати можливість обговорювати випадки з колегами через систему підтримки прийняття рішень, щоб залучати додаткову експертизу та поглиблювати свої знання.

Критерії приймання

- Я, як лікар, можу використовувати систему для обміну інформацією про конкретні випадки з іншими лікарями.
- Я, як лікар, можу запитувати поради у колег, додаючи до випадків анотації з примітками та запитаннями.
- Я, як лікар, можу використовувати відгуки та поради від колег для підвищення якості діагностики та лікування пацієнтів.

Історія користувача 6

Як лікар, я хочу мати змогу відслідковувати ефективність лікування пацієнта в режимі реального часу за допомогою системи підтримки прийняття рішень, щоб я міг швидко вносити корективи у план лікування, якщо це потрібно.

Критерії приймання

- Я, як лікар, можу бачити динаміку змін стану здоров’я пацієнта на основі даних, що надходять в систему.
- Я, як лікар, можу отримувати аналітичні звіти від системи, які допомагають оцінити ефективність лікування.
- Я, як лікар, можу швидко реагувати на зміни у стані здоров’я пацієнта, вносячи необхідні корективи у план лікування.

Високорівневі вимоги до функціоналу системи від пацієнта можна описати наступним чином – пацієнт повинен мати можливість реєструватися в системі підтримки прийняття рішень, ділитися своїми симптомами, переглядати свою

медичну історію та отримувати оновлення про свій стан здоров’я, щоб бути в курсі своєї ситуації та активно приймати участь в процесі діагностики та лікування.

Історія користувача 1

Як пацієнт, я хочу мати можливість вносити в систему детальний опис моїх симптомів, щоб мій лікар міг більш точно розуміти моє стан здоров’я і встановити правильний діагноз.

Критерії приймання

- Я повинен мати можливість внесення інформації про мої симптоми в систему.
- Я повинен мати можливість вказувати деталі щодо кожного симптому.
- Я повинен мати можливість оновлювати та редагувати ці внесені мною симптоми в будь-який час.
- Ця інформація має автоматично зберігатися в моєму медичному профілі і бути доступною для аналізу моїм лікарем.

Історія користувача 2

Як пацієнт, я хочу мати змогу отримувати сповіщення про нагадування про прийом ліків або виконання процедур, які мені призначено, щоб не пропустити жодного кроку у моєму лікуванні.

Критерії приймання

- Я повинен мати можливість отримувати нагадування на основі мого плану лікування.
- Я повинен мати можливість отримувати сповіщення на моєму пристрої з деталями про прийом ліків або процедури, які мені потрібно виконати.
- Я повинен мати можливість налаштувати час і формат нагадувань відповідно до своїх потреб.

Історія користувача 3

Як пацієнт, я хочу мати можливість спілкуватися з моїм лікарем або медичною командою через систему, щоб уточнити будь-які питання щодо мого стану здоров’я або плану лікування.

Критерії приймання

- Я повинен мати можливість відправляти повідомлення моєму лікарю або медичній команді.
- Я повинен бути впевненим, що мої повідомлення захищені, і лише відповідний персонал має доступ до них.
- Я повинен отримувати відповіді на мої запити вчасно і ефективно.
- Я повинен мати можливість переглядати історію моїх розмов з медичним персоналом.

Історія користувача 4

Як пацієнт, я хочу мати можливість переглядати мою медичну історію в системі, щоб зрозуміти мою попередню медичну історію і мати змогу планувати подальше лікування.

Критерії приймання

- Я повинен мати можливість переглядати збережену системою мою медичну історію, включаючи всі діагнози, призначені лікування та результати тестів.
- Я повинен мати можливість переглядати ці дані в хронологічному порядку.
- Я повинен мати доступ до інформації, яка має бути представлена зрозуміло, з використанням простих медичних термінів.

1.2 Деталізація проблеми дослідження і її контексту

1.2.1 Деталізація предмету наукового дослідження

Центральним предметом дослідження є концепція, розробка і впровадження системи підтримки прийняття рішень в області медичної діагностики. Система є цифровим інструментом, основною метою якого є обробка суб’єктивних відчуттів пацієнта, результатів медичних досліджень та інших діагностичних даних для створення деталізованого списку можливих захворювань, що могли б відповідати даним симптомам і оцінкам.

1.2.2 Деталізація завдання дослідження і розробки системи

Проект має кілька стратегічних цілей, які відображають бачення потенціалу цієї системи:

- Мінімізація ризику помилки при встановленні діагнозу шляхом впровадження в систему ефективних алгоритмів.
- Покращення діагностичної точності, підвищення відсотка правильно встановлених діагнозів за допомогою більш ґрунтовного аналізу медичних даних і симптомів пацієнтів.
- Оптимізація процесу діагностики, полегшення роботи медичних працівників завдяки автоматизації рутинних задач.
- Зміцнення довіри пацієнтів до медичних працівників шляхом підвищення прозорості процесу діагностики та покращення комунікації між лікарями та пацієнтами.

Для досягнення цих цілей потрібно вирішити ряд важливих завдань:

- Створити ефективні механізми для визначення можливого захворювання пацієнта на основі детального аналізу симптомів та результатів медичного обстеження.
- Розробити надійну систему реєстрації для лікарів, що дозволить їм використовувати систему для діагностики і взаємодії з пацієнтами.
- Створити зручний і безпечний механізм реєстрації для пацієнтів, забезпечуючи їм доступ до своїх медичних записів і інформації про свій стан здоров’я.
- Розробити механізм оновлення і розширення бази даних системи новими симптомами і асоційованими з ними захворюваннями, створюючи більш об’ємну і точну базу знань для діагностики.
- Розробити функціональність для детального перегляду медичної історії пацієнта, дозволяючи лікарям проводити більш глибокий аналіз їх здоров’я.
- Надавати пацієнтам можливість активно участі в процесі діагностики, дозволяючи їм ділитися власними відчуттями і оцінками.
- Забезпечити неперервне вдосконалення системи на основі зворотного зв’язку від лікарів та пацієнтів, що використовують систему.

1.2 Представлення даних предметної області

1.2.1 Деталізація сутностей системи

Система підтримки прийняття рішень для діагностики стану здоров'я пацієнта містить в собі цілий набір сутностей, кожна з яких відіграє ключову роль у її функціонуванні:

- Користувач – це особа, що має можливість взаємодіяти з системою. Це може бути пацієнт, лікар, адміністратор або інший медичний працівник, який використовує систему для різних потреб, включаючи діагностику, моніторинг, аналіз або звітність.
- Роль користувача – це специфічна категорія користувача, заснована на його професійних обов'язках і потребах. Вона визначає доступ користувача до функцій системи, що включають різноманітні дії, від збору симптомів до аналізу медичних даних.
- Повноваження ролі – це набір дозволів або обмежень, що визначає до яких конкретних ресурсів системи має доступ користувач з певною роллю. Наприклад, лікар може мати доступ до медичних даних усіх своїх пацієнтів, адміністратор – до конфігурації системи, а пацієнт – лише до своїх особистих даних.
- Симптом – це медичний параметр, що відображає фізичний або психологічний стан пацієнта. Симптоми використовуються для ідентифікації потенційних хвороб або розладів.
- Хвороба – це патологічний стан, що характеризується специфічним набором симптомів. Хвороби в системі пов'язані з симптомами, що дає змогу здійснювати диференційовану діагностику.
- Правило – це логічний структурний компонент, який визначає взаємозв'язки між різними сутностями у системі. Правила використовуються для моделювання взаємозв'язків між хворобами та симптомами, що сприяє точнішому прогнозуванню діагнозів за допомогою математичних моделей.

Потрібно зазначити, що цей опис сутностей є лише основним, і він може бути розширений додатковими сутностями залежно від специфічних вимог, технічних особливостей та потреб системи. Наприклад, можуть бути введені сутності для відстеження історії змін даних, для моделювання прогностичних сценаріїв або для інтеграції з іншими системами або джерелами даних. Така адаптивність є ключовим фактором для успішної реалізації та оптимізації системи, що підтримує прийняття інформованих та своєчасних рішень щодо діагностики стану здоров'я пацієнта.

1.2.2 Результат роботи системи

Вихідні дані, що випливають із функціонування системи, охоплюють широкий спектр діагностичних та інформаційних показників. Основним з них є набір можливих діагнозів, які система видає на основі вказаних пацієнтом симптомів. Кожен із цих потенційних діагнозів супроводжується числовою оцінкою впевненості, яка відображає ймовірність його відповідності фактичному стану здоров'я пацієнта. Зазначені діагнози ранжуються в порядку зменшення впевненості, починаючи від найбільш ймовірного.

У доповненні до переліку хвороб система генерує вичерпний звіт, що включає історію відвідувань пацієнта, деталі проведених діагностичних сесій, а також записи оцінок симптомів, які були зроблені під час кожного відвідування. Цей інформаційний ресурс створює можливість для динамічного аналізу стану здоров'я пацієнта, відстежуючи зміни в його самопочутті з часом.

Важливим також є дотримання принципу мінімальних привілеїв у доступі до цих даних. Зважаючи на чутливість медичної інформації, доступ до деталізованих звітів і персональних даних пацієнтів повинен бути строго контрольованим і доступним лише для тих осіб, для яких ця інформація необхідна у виконанні їх професійних обов'язків.

1.3 Системи управління інформацією

Системи управління інформацією, як правило, розробляються для керування і обробки інформації в організаціях. Існує кілька основних видів систем інформаційного управління, зокрема:

Системи управління базами даних. Ці системи забезпечують засоби для організації, зберігання, відновлення та аналізу даних. Вони забезпечують інтерфейс для користувачів або інших програм для отримання та маніпулювання даними.

Системи управління відносинами з клієнтами. Ці системи фокусуються на управлінні відносинами між компанією та її клієнтами. Вони зазвичай збирають та аналізують дані про клієнтів, щоб поліпшити обслуговування та відповідати їх потребам.

Системи планування ресурсів підприємства. Ці комплексні системи інтегрують різні процеси управління в одній системі, включаючи управління людськими ресурсами, фінансами, ланцюгом постачання та інше.

Системи підтримки прийняття рішень. Це інформаційні системи, розроблені для підтримки процесу прийняття рішень в організаціях. Вони зазвичай забезпечують відповідні дані або аналітику, які можуть допомогти користувачам у вирішенні складних проблем.

Експертні системи. Це тип системи штучного інтелекту, яка імітує роботу експерта, аналізуючи вхідні дані і використовуючи базу знань для вирішення специфічних проблем. Вони зазвичай використовуються в областях, де спеціалізовані знання є особливо важливими, наприклад, в медицині, науці, інженерії тощо.

Системи управління знаннями. Ці системи спрямовані на зберігання, відновлення та розповсюдження знань в межах організації. Вони можуть включати в себе віртуальні простори для співпраці, інструменти для управління документами, пошукові системи та інше.

Інформаційні системи. Ці системи використовуються для обробки інформації, включаючи зберігання, відновлення, передачу та маніпуляцію даними.

Кожна з цих систем виконує спеціалізовані функції і може бути важливою частиною стратегії інформаційного управління організації.

В даному розділі ми розглянемо особливості та взаємозв’язків між експертними системами та системами підтримки прийняття рішень. Вони є двома видами систем інформаційного управління, кожна з яких має свою власну особливість, але вони тісно пов’язані. Взаємозв’язок між ними полягає в тому, що експертні системи часто

використовуються в рамках більшої системи підтримки прийняття рішень для надання спеціалізованої експертизи або для автоматизації певних аспектів процесу прийняття рішень. Іншими словами, експертні системи можуть бути розглянуті як частина більшої системи підтримки прийняття рішень, хоча вони можуть також існувати і функціонувати незалежно.

1.3.1 Структура системи підтримки прийняття рішень

Структуру системи підтримки прийняття рішень (СППР) можна описати шляхом визначення основних підсистем або компонентів. Один зі підходів розділяє СППР на три основні підсистеми – діалог, дані та моделі (рис. 1.1) [4].

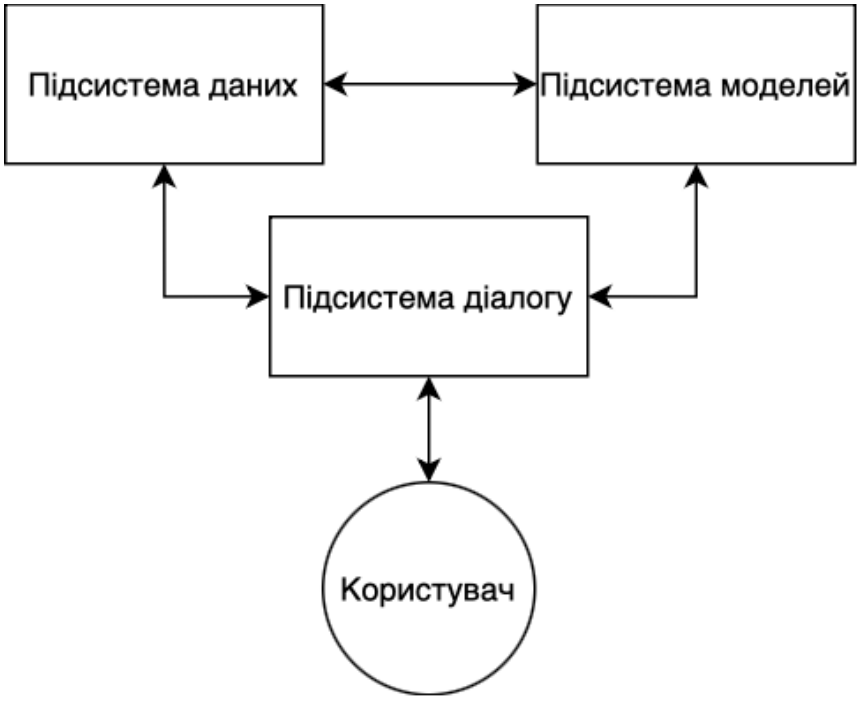

Рис. 1.1. Основні підсистеми СППР

Підсистема даних. Цей компонент зберігає та обробляє всі дані, що пов'язані з пошуком рішення. Він може включати історичні дані, транзакційні дані, інтегровані дані з різних джерел, а також дані, зібрані в реальному часі. Наприклад, для СППР, що використовується в медичній області, підсистема даних може зберігати медичні записи пацієнтів, дані про лікування, дані про результати клінічних досліджень тощо.

Підсистема моделей. Цей компонент використовує моделі або алгоритми для обробки даних з підсистеми даних та виробляє висновки або рекомендації. Моделі

можуть бути статистичними, фінансовими, оптимізаційними або іншими типами моделей, в залежності від конкретної області використання СППР. Для медичної СППР модель може включати моделі прогнозування хвороби або моделі оптимізації лікування.

Підсистема діалогу. Цей компонент відповідає за взаємодію з користувачем. Підсистема забезпечує інтерфейс для користувача (де він може вводити свої параметри або запити) і відображає результати, отримані від підсистеми моделей. Для медичної СППР це може бути інтерфейс, де лікар вводить симптоми пацієнта, а потім система показує можливі діагнози або рекомендації щодо лікування.

1.3.2 Структура експертної системи

Основні компоненти експертної системи (ЕС) можна визначити за допомогою діаграми (рис. 1.2) [4].

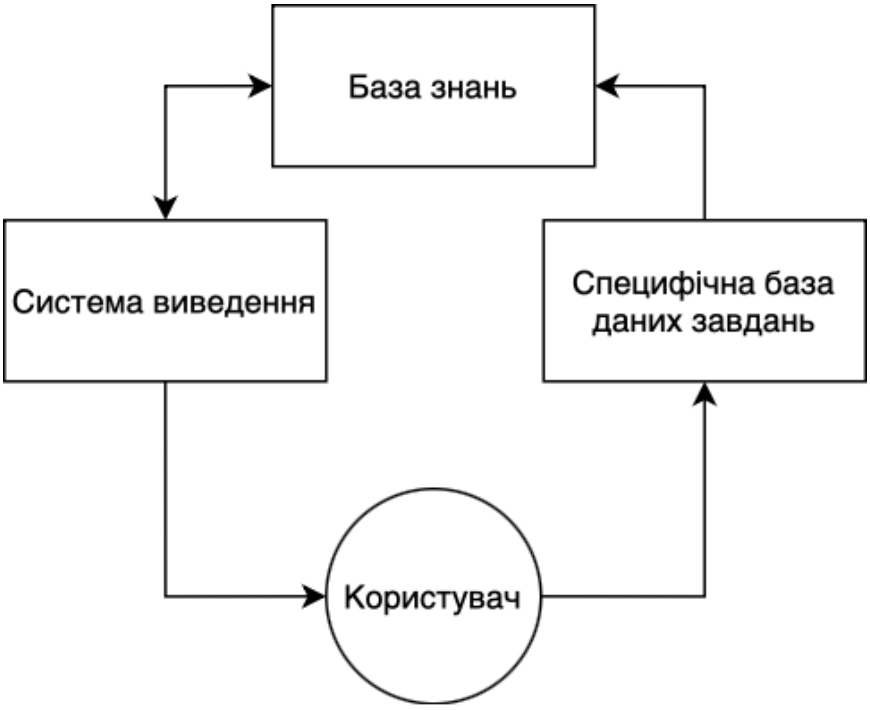

Рис. 1.2. Основні компоненти ЕС

Специфічна база даних завдань. Цей компонент містить дані, що відносяться до конкретної області, де система має бути застосована. Наприклад, для експертної системи в медицині, це можуть бути дані про симптоми різних хвороб, діагностику, лікування тощо.

База знань. Це основний компонент експертної системи, який зберігає знання та досвід експертів в області, яка охоплюється системою. Це знання представлене в формі "правил", що містять взаємозв’язки між різними фактами і концепціями в області. Наприклад, в медичній експертній системі, правило може бути таким: "Якщо пацієнт має симптоми A, B і C, тоді він може мати хворобу X".

Система виведення. Це компонент системи, що використовує знання з бази знань для вирішення конкретних проблем, які представлені користувачем. Він застосовує правила до вхідних даних та виводить рекомендації або рішення. В медичній експертній системі система виведення може використовувати знання про різні симптоми та хвороби для діагностики пацієнтів.

Крім того, враховуючи потреби в постійному покращенні і вдосконаленні експертної системи в процесі роботи, виникає необхідність в побудові більш складних архітектур ЕС. В цих архітектурах додаткова увага приділяється базі знань та процесу її життєвого циклу. Наприклад, розглянемо наступну архітектуру ЕС на рис. 1.3 [5]:

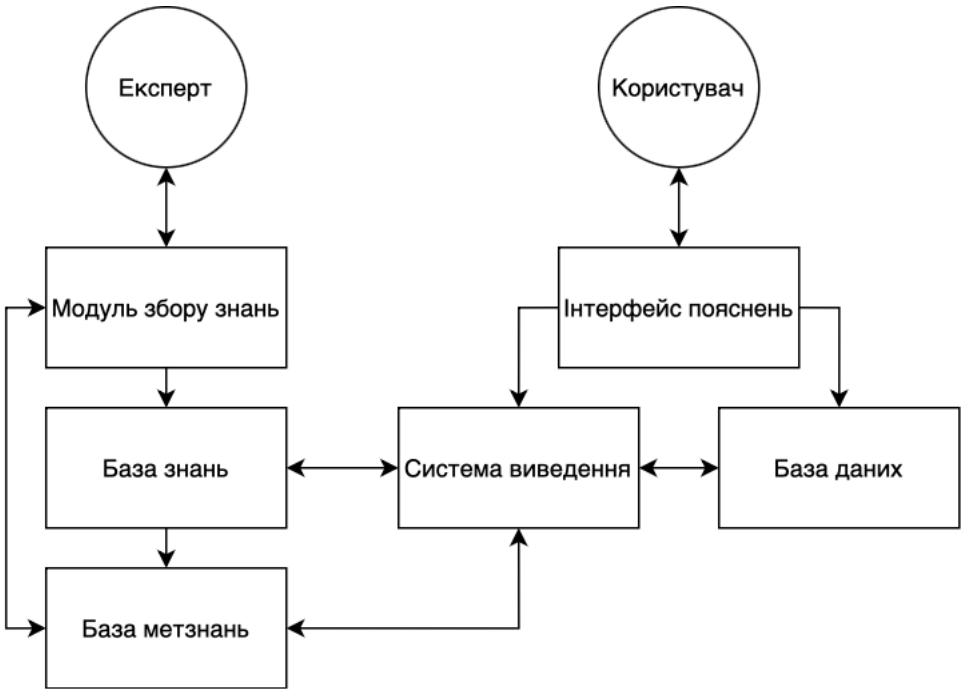

Рис. 1.3. Розширена архітектура ЕС

Основні компоненти такої експертної системи можна описати наступним чином:

Модуль збору знань. Цей модуль відповідає за збір інформації від експертів і конвертацію цієї інформації в формат, який може бути використаний в рамках системи. Зазвичай це включає процеси інтерв'ювання експертів, виявлення знань та формулювання правил, що будуть використовуватись в системі.

База знань. Це місце, де зберігається вся інформація, яка була отримана від експертів. Вона може включати факти, правила, процедури та інші типи знань.

База метазнань. Це база знань про знання, яке використовується в системі. Метазнання може включати інформацію про те, як використовувати та інтерпретувати знання в базі знань.

Інтерфейс пояснень. Це інтерфейс, через який користувач може отримати пояснення від системи. За допомогою цього інтерфейсу система може відповідати на запитання користувача, пояснюючи, як було отримано певне рішення або висновок.

Система виведення. Це серцевина системи, яка використовує правила, записані в базі знань, для виведення висновків на основі вхідних даних.

База даних. Це частина експертної системи, де зберігаються дані, що використовуються для аналізу. Дані можуть включати історії випадків, статистику, вимірювальні дані тощо. Ці дані використовуються при виведенні висновків.

Дана система дозволяє розв'язувати такі задачі, як отримання нових знань, управління знаннями та пояснення виведення.

1.3.3 Структура нечіткої системи виведення

Нечіткі системи виведення або системи на основі нечіткої логіки являють собою підтип експертних систем, де знання представлені за допомогою нечітких моделей, а логіка виведення є узагальненням поняття виведення класичної логіки на нечіткий випадок.

Розглянемо типові компоненти нечіткої системи виведення на рис. 1.4:

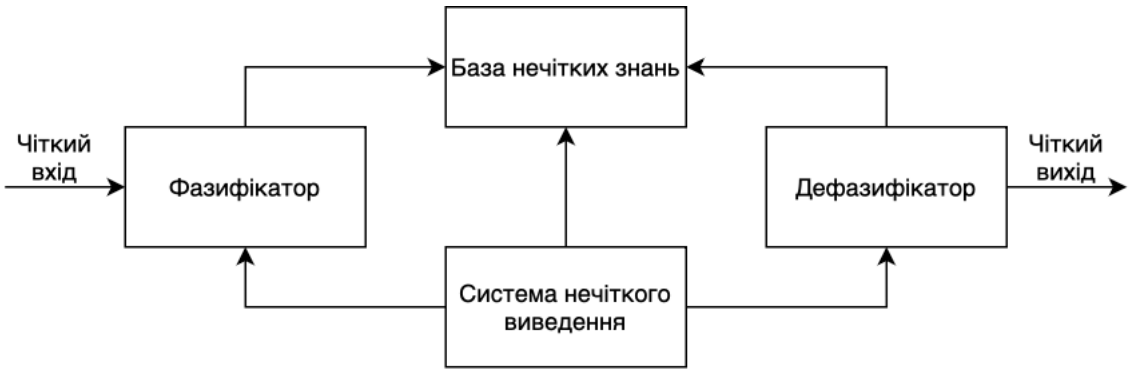

Рис. 1.4. Типова архітектура нечіткої системи виведення

База нечітких знань. Ця база знань містить нечіткі правила, які формуються на основі досвіду експертів або засновані на даних. Нечітке правило може виглядати так: "Якщо температура пацієнта є "Дуже високою", тоді збільшити дозу прийому ліків".

Система нечіткого виведення. Ця система виведення використовує нечіткі правила з бази знань. На вхід системи подаються нечіткі вхідні дані. Використовуючи правила, одержуємо нечіткий вихідний результат.

Особливої уваги заслуговують два додаткових компонента, які притаманні саме системам нечіткого логічного виведення.

Фазифікатор. Цей модуль конвертує вхідні дані в нечіткі величини. Нечіткі величини описуються нечітким множинами, кожному елементу яких приписується ступінь належності до множини. Наприклад, температуру пацієнта можна описати нечіткими величинами "Низька", "Нормальна", "Висока", "Дуже висока", які описуються нечіткими множинами у відповідних просторах.

Дефазифікатор. Після отримання виходу системи цей модуль конвертує його в конкретне числове значення для подальшого застосування. Є кілька різних методів дефазифікації, які можуть бути використані в залежності від конкретної системи і її вимог.

Отже, у нечіткій системі логічного виведення інформація обробляється обома компонентами, переходячи від точних чисел до нечітких множин і навпаки.

1.3.4 Класифікація знань

Окремого деталізованого розгляду заслуговує класифікація знань і їх застосування у вищезгаданих системах.

Основні типи знань, що зберігаються в базі знань, можуть включати [6]:

Інтенсіональні знання. Це абстрактні, загальні правила або принципи, що є основою для прийняття рішень. Наприклад, "Якщо температура висока, то прийняти ліки від температури". Інтенсіональні знання – це знання, які мають загальне значення та можуть бути застосовані до різних ситуацій.

Екстенсіональні знання. Це конкретні факти або дані, що стосуються певної області або ситуації. Екстенсіональні знання – це знання, які стосуються конкретних екземплярів чи випадків. Наприклад, конкретна температура пацієнта на даний момент.

Метазнання. Це знання про знання. Вони допомагають системі краще розуміти, як використовувати та інтерпретувати інші типи знань в базі знань. Метазнання можуть включати інформацію про те, як знання були отримані, які методи використовувались для виведення цих знань, які знання більш важливі у порівнянні з іншими знаннями тощо.

Існують також інші типи знань, що можуть бути включені в базу знань в залежності від конкретної системи, включаючи процедурні знання (інформація про те, як виконати певні дії або задачі), евристичні знання (накопичуються інтелектуальною системою в процесі її функціонування, а також закладаються в ній апріорі, але не мають статусу абсолютної істинності в даній проблемній області).

Метазнання, інтенсіональні знання та екстенсіональні знання взаємодіють у складному балансі. Метазнання визначає, як інтенсіональні та екстенсіональні знання мають бути використані та інтерпретовані. Інтенсіональні знання використовуються для виведення нових екстенсіональних знань або для прийняття рішень на основі екстенсіональних знань.

Розглянемо більш детально кожен з цих типів.

В експертній системі для діагностики хвороб інтенсіональні знання можуть бути такими (приклади):

Діагностика. Інтенсіональні знання включають загальні принципи і методи, які використовуються для діагностики хвороби. Наприклад, "Якщо у пацієнта висока температура, сухий кашель і втрата смаку, ймовірність, що у пацієнта COVID-19, є висока".

Симптоми хвороби. Інтенсіональні знання включають описи симптомів різних хвороб, їх спільні та відмінні ознаки. Наприклад, "Жовтяниця є частим симптомом гепатиту, але вона може бути присутня і в інших станах".

Правила інтерпретації тестів. Інтенсіональні знання можуть включати правила інтерпретації результатів тестів. Наприклад, "Якщо рівень глюкози в крові на голодний шлунок перевищує 126 мг/дл, це може бути ознакою діабету".

Стратегії лікування. Інтенсіональні знання також можуть включати інформацію про типові стратегії лікування для різних хвороб. Наприклад, "Антибіотики є стандартним лікуванням для бактеріальних інфекцій".

Зв'язки між симптомами та хворобами. Інтенсіональні знання включають зв'язки між різними симптомами та хворобами. Наприклад, "Серцевий напад часто супроводжується болем в грудях".

Ці та інші інтенсіональні знання створюють основу для роботи експертної системи, вони є загальними і можуть бути застосовані до широкого діапазону конкретних ситуацій.

В експертній системі для діагностики хвороб екстенсіональні знання можуть бути такими (приклади):

Дані про пацієнтів. Екстенсіональні знання включають конкретні дані про пацієнтів, такі як їх вік, стать, медична історія, симптоми, результати медичних тестів тощо. Наприклад, "Пацієнтка, 25 років, скаржиться на високу температуру, біль у горлі та сухий кашель".

Історії випадків. Екстенсіональні знання можуть також включати конкретні історії випадків або сценарії, які допомагають встановити діагноз. Наприклад, "Пацієнтка, 50 років, яка раніше страждала від високого кров'яного тиску, пройшла тест на гіпертонію, який показав підвищений рівень".

Результати тестів. Екстенсіональні знання включають конкретні результати діагностичних тестів. Наприклад, "Результати крові пацієнта показали підвищені рівні лейкоцитів, що може вказувати на інфекційний процес".

Конкретні діагнози. Екстенсіональні знання можуть також включати конкретні діагнози, які були встановлені в результаті діагностичного процесу. Наприклад, "На основі симптомів та результатів тестів, пацієнтові було діагностовано пневмонію".

Екстенсіональні знання, такі як інформація про конкретних пацієнтів, їх симптоми, медична історію, результати тестів тощо, є фундаментальними для експертних систем і використовуються для встановлення конкретних діагнозів.

В експертній системі для діагностики хвороб метазнання можуть бути такими:

Правила вибору алгоритмів діагностики. Метазнання можуть включати інформацію про те, які алгоритми або методи діагностики слід використовувати в залежності від характеристик пацієнта або специфіки його симптомів. Наприклад, "Якщо пацієнт молодий і здоровий, використовуйте алгоритм А для діагностики. Якщо пацієнт у віці і має хронічні захворювання, використовуйте алгоритм Б".

Правила оцінки достовірності діагнозу. Метазнання можуть також вказувати, як оцінити надійність потенційного діагнозу. Наприклад, "Якщо всі основні симптоми хвороби присутні і не мають альтернативних пояснень, діагноз можна вважати високодостовірним".

Правила оновлення бази знань. Метазнання можуть містити інструкції про те, як і коли оновлювати базу знань системи з новою інформацією про хвороби, симптоми, тестування тощо.

Ці та інші типи метазнань допомагають керувати процесом діагностики і використовувати інтенсіональні та екстенсіональні знання якомога ефективніше.

Крім того, існує також класифікація знань з точки зору зміни у часі. Нижче приведені приклади таких знань.

Статичні знання. Це знання, які залишаються незмінними протягом часу. Наприклад, закони фізики, математичні теореми, або факти, які не змінюються. ("Симптоми грипу включають високу температуру, головний біль, сухий кашель, біль

у м’язах, слабкість та нездужання"). Ці знання можуть бути закодовані в базі знань і використовуватися без змін протягом тривалого часу.

Динамічні знання. Це знання, які можуть змінюватися в часі. Наприклад, актуальні життєві показники конкретного пацієнта, які відстежується в реальному часі. Динамічні знання вимагають постійного оновлення і часто залежать від зовнішніх джерел інформації.

Інтенсіональні, екстенсіональні та метазнання можуть бути як статичними, так і динамічними. Наприклад, інтенсіональні знання, що базуються на сталих наукових принципах, будуть статичними, а інтенсіональні знання, що базуються на постійно оновлюваній статистиці, будуть динамічними. Теж саме стосується екстенсіональних знань (наприклад, статичні факти про історичні події проти динамічних даних про поточний стан системи) і метазнань (наприклад, сталі правила обробки даних проти адаптивних стратегій, які змінюються відповідно до контексту).

Розглянемо декілька прикладів для системи діагностики хвороб:

Статичні Інтенсіональні Знання. Це можуть бути стабільні медичні принципи або діагностичні критерії. Наприклад, "Якщо пацієнт має підвищену температуру, кашель і втрату смаку, це може бути індикатором COVID-19".

Динамічні Інтенсіональні Знання. Це можуть бути останні медичні дослідження або статистика, яка регулярно оновлюється. Наприклад, "Нові дослідження показують, що симптомом Delta-варіанту COVID-19 може бути різкий головний біль".

Статичні Екстенсіональні Знання. Це можуть бути конкретні дані про пацієнта, які не змінюються, наприклад, дата народження, група крові, вік тощо.

Динамічні Екстенсіональні Знання. Це можуть бути дані, які регулярно оновлюються, наприклад, поточний стан пацієнта, останні результати аналізів, тиск, серцевий ритм тощо.

Статичні Метазнання. Це можуть бути загальні принципи або процедури, які система використовує для діагностики. Наприклад, "Якщо пацієнт має два або більше симптоми COVID-19, порекомендуйте йому зробити тест".

Динамічні Метазнання. Це можуть бути адаптивні стратегії, які система використовує, наприклад, "Якщо в області, де проживає пацієнт, виявлено новий варіант COVID-19, збільшити кількість потенційних симптомів для рекомендації тестування".

1.3.5 Розширена структура системи підтримки прийняття рішень

Враховуючи попередню інформацію, можна сформувати наступну розширену архітектуру системи для діагностики стану здоров’я пацієнтів (рис. 1.5).

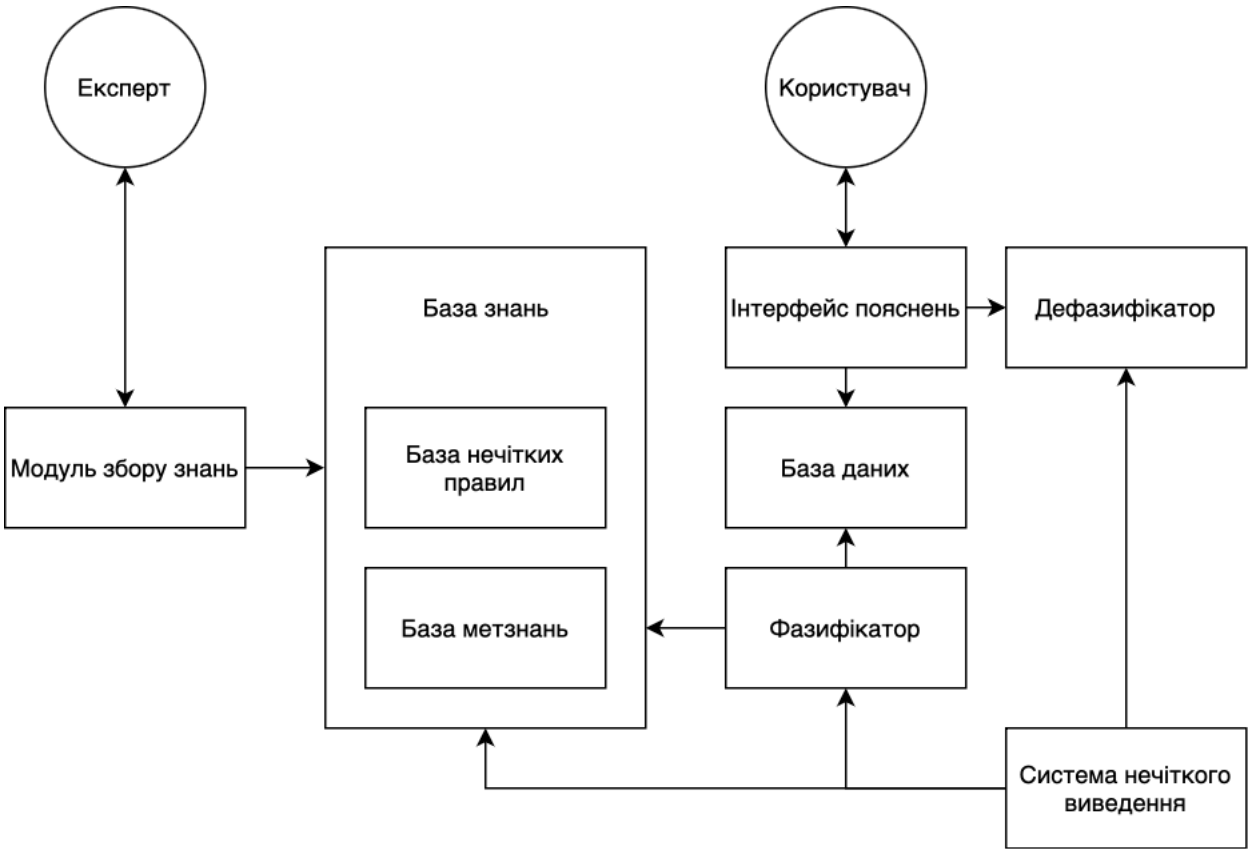

Рис. 1.5. Розширена архітектура системи підтримки прийняття рішень

Дана архітектура може мати значні переваги при вирішенні завдань діагностики стану здоров’я пацієнтів за рахунок наступних функціональностей:

Комплексний аналіз. Використання різних видів аналізу для вирішення однієї задачі, що забезпечує більш глибоке та багатогранне розуміння проблеми.

Впевненість і точність. Використання знань експертів для створення більш точних та впевнених діагнозів. Система нечіткого виведення дозволяє обробляти

нечітку інформацію та невизначеність, які часто виникають при медичній діагностиці.

Розширена підтримка прийняття рішень. Використання статистичних та математичних моделей для підтримки прийняття рішень з метою забезпечення обґрунтованих рекомендацій.

Адаптивність. Поступове вдосконалення за допомогою механізмів навчання та адаптації з метою покращення своїх висновків та рекомендацій на основі отриманого досвіду.

Об'єднання даних та знань. Можливість об'єднувати як кількісні дані (наприклад, медичні показники), так і якісні дані (наприклад, описи симптомів або відгуки пацієнтів).

Зручність для користувача. Можливість більшої інтерактивності та зрозумілості інтерфейсу користувача з метою забезпечення наочності та простоти використання.

1.4 Життєвий цикл та придбання знань системи підтримки прийняття рішень

Взагалі, ефективний життєвий цикл розробки системи повинен гарантувати створення якісної системи, що відповідає потребам клієнтів. Він реалізується у встановлені терміни та виділений бюджет і забезпечує надійну роботу у діючій та майбутній IT-інфраструктурі.

Життєвий цикл розробки системи представляє собою концептуальну схему, яка визначає норми та методики для створення або модифікації систем на протязі їхнього існування і складається з наступних етапів:

- Етап планування
- Етап аналізу та збирання вимог
- Етап проектування та прототипування
- Етап розробки програмного забезпечення
- Етап тестування програмного забезпечення
- Етап впровадження та інтеграції

- Етап експлуатації та технічного обслуговування

1.4.1 Деталізація етапів життєвого циклу системи підтримки прийняття рішень

З точки зору системи підтримки прийняття рішень, яка має свою додаткову специфіку, ці етапи можуть бути розширені додатковими пунктами, які відповідають за роботу зі знаннями.

Пропонується розглянути наступну типову схему життєвого циклу для системи підтримки прийняття рішень, що враховує управління знаннями (рис. 1.6) [7].

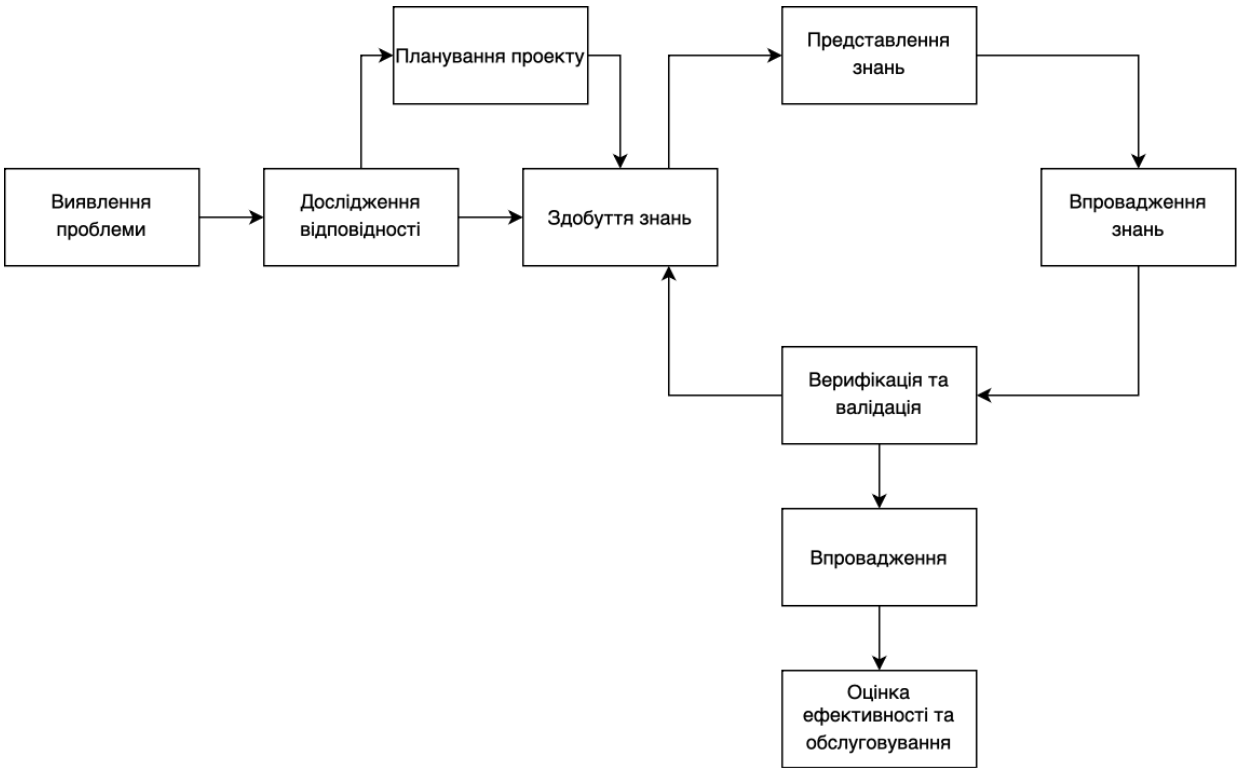

Рис. 1.6. Етапи життєвого циклу системи підтримки прийняття рішень

На рис. 1.6 окремі блоки описуються наступним чином:

1. **Виявлення проблеми.** Цей етап призначений для виявлення існуючих проблем та можливостей, де установа (організація) може отримати вигоду від впровадження експертної системи. Основна мета – з'ясувати, де система може принести максимальну користь, і допомогти визначити області, які потребують удосконалення або оптимізації. Процес включає в себе аналіз поточного стану організації, визначення її ключових потреб та вимог. Також на цьому етапі встановлюються загальні цілі та очікування від проекту, формулюються гіпотези

щодо потенційних вигод та розробляються критерії для оцінки успішності проекту. Важливо залучити до процесу виявлення проблеми ключових експертів та інших зацікавлених сторін, таких як керівники, співробітники та клієнти, щоб забезпечити всебічне та об'єктивне розуміння проблеми.

2. **Дослідження відповідності.** На цьому етапі проводиться детальна оцінка потенційного проекту з точки зору його технічної, економічної та оперативної відповідності. Основна мета – визначити, чи може бути реалізована експертна система в умовах конкретної організації.

   Розглядаються наступні типи відповідностей:

   - Технічна відповідність – чи є достатньо технологій, знань та ресурсів для розробки системи та чи зможе вона інтегруватися з уже існуючими системами організації.
   - Економічна відповідність – чи є проект вартісно ефективним і чи буде він приносити користь організації у фінансовому плані.
   - Експлуатаційна відповідність – чи може організація ефективно експлуатувати систему з урахуванням її поточної структури, культури та операційних процесів.

3. **Планування проекту.** На цьому етапі встановлюються основні параметри проекту, а саме:

   - Команда проекту. Обрання учасників команди відбувається з огляду на їхні навички та досвід, визначаються ролі та обов'язки кожного.
   - Робоче середовище. Вибір та налаштування необхідних технологій та інструментів для ефективної роботи команди.
   - Графік і бюджет. Розробка чіткого плану робіт з визначеними термінами та обсягами, а також розрахунок бюджету проекту з урахуванням всіх можливих витрат.
   - Оцінка ризиків. Ідентифікація та аналіз потенційних ризиків, розробка стратегій їх мінімізації.

Кожен з цих параметрів є важливим для забезпечення успішного виконання проекту і досягнення запланованих результатів.

4. **Здобуття знань.** На цьому етапі проводиться збір інформації та знань від експертів у предметній області. Головна мета – визначити ключові деталі та особливості предметної області, які необхідні для розробки системи. Цей етап реалізується за допомогою наступних кроків:
    - Збір знань. Використовуючи інтерв'ю та аналіз документації, здійснюється збір знань та інформації від експертів.
    - Формулювання вимог до знань. На основі отриманих знань формуються вимоги до системи, які стають основою для наступних етапів розробки.

Цей етап дозволяє забезпечити точне та повне розуміння предметної області та визначити, які саме знання та інформація будуть необхідні для впровадження ефективної експертної системи.

5. **Представлення знань.** Цей етап фокусується на структуризації та організації зібраних на попередньому етапі знань, а також на представленні цих знань у формі, яка придатна для розробки системи. Етап реалізується за допомогою кроків:
    - Моделювання знань. Створення моделей та діаграм, які відображають ключові поняття предметної області та їх взаємозв'язки, для кращого розуміння структури та сутності предметної області.
    - Формалізація знань. Перетворення неструктурованих знань у структуровану форму (використовуючи відповідні мови представлення знань та методології) з метою підготовки їх до імплементації в системі.
    - Верифікація моделей знань. Перевірка створених моделей та формалізованих знань на предмет їх правильності, повноти та консистентності відносно реального світу.
    - Оптимізація. Вдосконалення моделей та формалізованих знань для забезпечення їхньої точності, відповідності та ефективності у системі.

Цей етап важливий для підготовки та оптимізації знань для подальшої імплементації у експертну систему і виступає гарантом того, що система буде базуватися на вірних, зрозумілих та узгоджених знаннях.

6. **Впровадження знань.** На цьому етапі формалізовані знання перетворюються у робочий прототип. Ключове завдання – адекватно транслювати зібрані знання у програмні рішення. Етап містить наступні кроки:
    - Розробка алгоритмів. Створення алгоритмів на основі формалізованих знань для реалізації конкретних завдань системи.
    - Кодування. Трансформація розроблених алгоритмів та моделей у програмний код за допомогою відповідних мов програмування та технологій.
    - Тестування функціональності. Виконання тестів для перевірки правильності реалізації знань у коді та для виявлення та виправлення будь-яких помилок та проблем.
    - Оптимізація та налаштування. Поліпшення ефективності коду, налаштування параметрів та вдосконалення реалізації на основі результатів тестування.

Мета цього етапу – забезпечити правильність імплементації в системі зібраних знань перед подальшою її верифікацією та валідацією.

7. **Верифікація та валідація.** На цьому етапі відбувається детальна оцінка робочого прототипу, щоб гарантувати, що він відповідає всім вимогам та очікуванням, визначеним у процесі планування та розробки. Етап реалізується кроками:
    - Перевірка відповідності. Проведення тестів і оцінок для переконання, що система вірно реалізує задумані функції та придбані знання.
    - Валідація коректності. Переконання, що система робить те, що вона повинна робити, виконує правильні операції та видає правильний результат в реальних умовах.
    - Аналіз відгуків. Вивчення відгуків та зауважень від експертів та користувачів для ідентифікації можливих покращень та корекцій.
    - Корекція та оптимізація. Внесення необхідних змін та покращень до прототипу на основі результатів верифікації та валідації.

Цей етап гарантує, що кінцевий продукт буде якісним, надійним та відповідатиме всім вимогам і стандартам, перед тим як він буде введений в експлуатацію.

8. **Впровадження.** На цьому етапі відбувається встановлення фінального прототипу в експлуатаційному середовищі, проводиться навчання користувачів і розробляється необхідна документація. Етап реалізується кроками:
    - Розгортання системи. Запуск та налаштування системи в реальному середовищі, забезпечення її готовності до експлуатації.
    - Навчання користувачів. Проведення інструктажу та навчальних сесій для майбутніх користувачів системи.
    - Розробка документації. Створення інструкцій, керівництв та іншої супровідної документації, необхідної для користувачів та обслуговуючого персоналу.
9. **Оцінка ефективності та обслуговування.** Експлуатація системи, моніторинг її роботи та продуктивності, і проведення необхідного обслуговування та удосконалень, де:
    - Моніторинг та аналіз. Постійне відстеження та оцінка роботи системи для забезпечення її стабільності та ефективності.
    - Технічне обслуговування. Проведення регулярних перевірок, оновлень та ремонтних робіт для забезпечення надійності та довговічності системи.
    - Оптимізація системи. Вдосконалення системи на основі зібраних даних та відгуків користувачів для покращення її функціональності та продуктивності.

1.4.2 Діючі ролі

Розробка експертної системи — це багатоетапний процес, що вимагає злагодженої роботи команди спеціалістів різних профілів. Здебільшого для реалізації даного завдання працює команда з трьох ключових експертів:

- Експерт з предметної галузі
- Інженер з інженерії знань

- Програміст-розробник інструментальних засобів

Експерт з предметної області є ключовою ланкою в цьому процесі. Цей фахівець має глибокі знання у своїй предметній галузі, для якої розробляється відповідна система. Він надає інформацію, базуючись на своєму професійному досвіді, який він отримав протягом багатьох років практики.

Інженер з інженерії знань — це особа, яка будує міст між експертом предметної області та програмістами. Він застосовує сучасні методології для збору, структуризації та формалізації знань, отриманих від експерта предметної області, для їх подальшого використання у системі.

Програміст-розробник, в свою чергу, фокусується на створенні інструментальних засобів для експертної системи. Ці інструменти не тільки спрощують розробку системи, але і забезпечують її інтеграцію з потрібним програмним та апаратним середовищем.

1.4.3 Моделі придбання знань

Процес збору знань від фахівця предметної області (чи з інших доступних джерел інформації) та їх інтеграції в експертну систему відомий як придбання знань. Зазвичай основним постачальником знань є експерт, однак інформацію також можна отримати з емпіричних даних та спеціалізованих текстів, що містять деталі про цю предметну область.

Цей процес є важливим і складним етапом розробки, оскільки успіх експертної системи значною мірою визначається якістю та обсягом інтегрованих в неї знань. Якість рішень, які приймає система, та її здатність ефективно вирішувати задачі безпосередньо залежать від глибини та правильності інтегрованих знань.

Складність процесу придбання знань походить від величезного обсягу даних, з якими оперує експерт, а також від того, що деяка частина знань може бути не повністю прийнята до уваги самим фахівцем. Отже, придбання знань вимагає від розробників високого професійного рівня та аналітичних здібностей для того, щоб забезпечити правильність та повноту зібраної інформації.

Розрізняють 3 основні фази придбання знань [8]:

1. **Попередня фаза** – дана фаза визначається відсутністю остаточно сформованої експертної системи. Знання на цій стадії добуваються інженером знань через взаємодію з експертом. Обов'язки інженера знань на цьому етапі концентруються на зборі від експерта ключової інформації з конкретної предметної області (такої як основні концепції, відносини) та на формуванні (на базі цієї інформації) загального уявлення про структуру даних і принципи конструювання експертної системи. Цей етап вимагає детального аналізу та систематизації отриманої інформації для розробки ефективної структури знань та вибору оптимальних методів побудови системи, що забезпечать високий рівень її функціональності та адаптивності.
2. **Початкова фаза** – на даному етапі здійснюється інтеграція системи з певними початковими знаннями, які характеризують організацію, структуру та механізм представлення бази знань. Враховуючи те, що формулювання та інтеграція таких знань вимагають глибокого розуміння основ програмування та специфіки функціонування проектованої експертної системи, цей процес може ефективно виконуватися лише інженером знань, оскільки йому, на відміну від експерта, властиве технічне розуміння системи.
3. **Фаза накопичення** – на етапі накопичення відбувається процес збору знань з обраної області експертизи. У контексті сучасного стану розвитку, процес збору знань на даному етапі часто впроваджується експертом та інженером знань у спільній комунікації. Завдання, які реалізуються на цій стадії, включають: виявлення та аналіз неточностей, неповноти чи конфліктів у знаннях, що інтегруються в експертну систему; збір додаткових знань для вирішення виявлених проблем та неузгодженностей; конвертацію отриманих знань до формату, доступного для експертної системи; інтеграцію нових набутих знань із вже існуючими. Варто підкреслити, що в ході даного етапу покращуються різноманітні форми знань, необхідні для оптимізації роботи та підвищення ефективності експертної системи. Передбачається, що реалізація зазначених завдань сприятиме якісній та безперебійній роботі експертної системи.

Розглянемо декілька типових моделей придбання знань:

1. **Рання модель придбання знань** – в ранніх моделях придбання знань програміст мав занурюватися у предметну область за допомогою експерта, а після отримання необхідних знань сам виконувати роль "експерта" під час розробки системи (рис. 1.7). Обмежене розуміння предметної області не дозволяло програмісту гарантувати абсолютну повноту та консистентність зібраних знань. До того ж всі неминучі модифікації системи, що відбувалися без чіткого розмежування бази знань та механізму виведення, створювали проблеми у збереженні первісної консистентності знань. Зміни в системі часто призводили до того, що вже досягнута консистентність знань втрачалася, що загрожувало втратою надійності та функціональності системи.

Рис. 1.7. Рання модель придбання знань

2. **Модель придбання знань за допомогою інженера знань** – подальші досягнення в області систем штучного інтелекту призвели до необхідності відокремлення знань від програмного коду (рис. 1.8). Знання стали подавати у формі інформаційних структур, відомих як бази знань. У такому контексті експерт може взаємодіяти із системою безпосередньо або опосередковано, через інженера знань. Перевагою даного підходу над попередньою моделлю є те, що структурування знань у базі дозволяє більш гнучко модифікувати та адаптувати знання, полегшуючи процес оновлення та оптимізації системи.

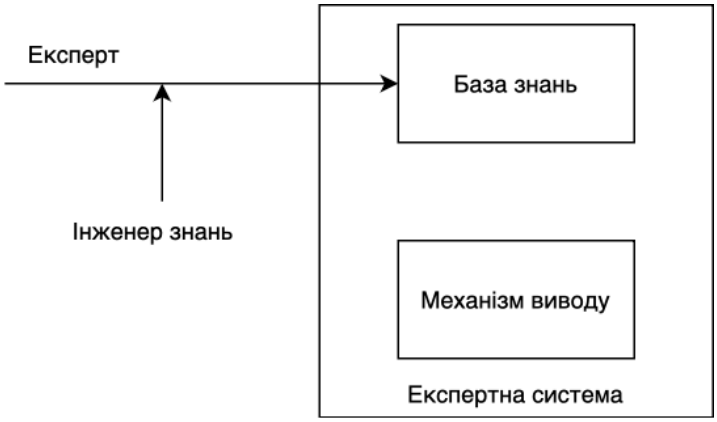

Рис. 1.8. Модель придбання знань за допомогою інженера знань

3. **Модель придбання знань ЕС за допомогою інтелектуального редактора** – в даній моделі інтегровано інтелектуальний редактор, що має розвинуті діалогові властивості та знання (відомі як метазнання) щодо структуризації бази знань (рис. 1.9). Цей інтелектуальний редактор автоматично конвертує знання до формату, який оптимально сприймається експертною системою, а також адаптує базу знань, інтегруючи новостворені знання із вже існуючими. Дана технологія дозволяє забезпечити більш гнучкий та ефективний процес синтезу знань, що сприяє постійному розвитку та вдосконаленню системи.

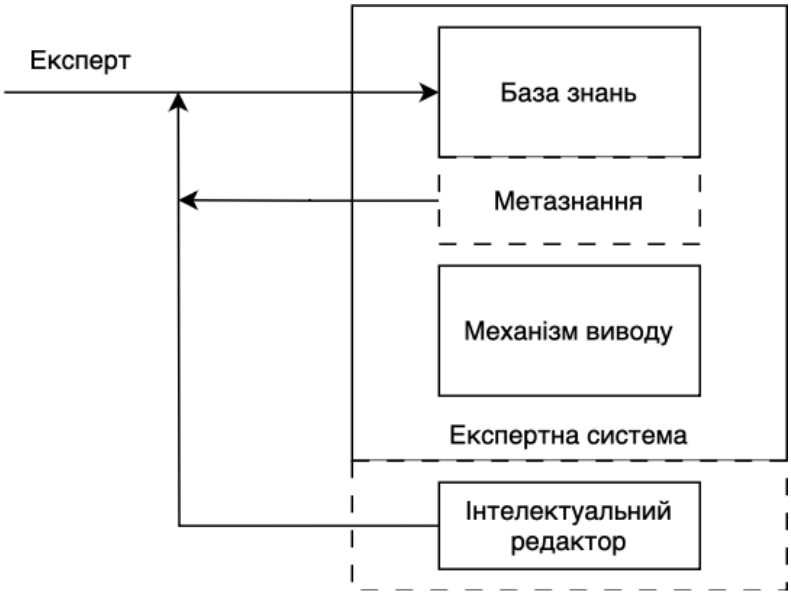

Рис. 1.9. Модель придбання знань ЕС за допомогою інтелектуального редактора

4. **Модель придбання знань ЕС за допомогою індуктивної програми.** Індуктивна програма вивчає дані з конкретної області, формуючи автоматично відношення та правила, які дозволяють детально описати цю область (рис. 1.10). Завдяки цьому можна створювати точні моделі, які відображають основні характеристики та закономірності обраної предметної області.

Рис. 1.10. Модель придбання знань ЕС за допомогою індуктивної програми

5. **Модель придбання знань ЕС за допомогою програми розуміння тексту.** У даній моделі значну роль відіграє спеціалізована програма, яка здатна інтерпретувати текстові дані для придбання знань (рис. 1.11). Завдання інтерпретації тексту є вкрай складним, оскільки воно вимагає обробки природної мови та спроможності відтворити модель специфічної предметної області на основі текстової інформації. Правильно виконана інтерпретація тексту дозволяє системі вивчати конкретні предметні області та конструювати точні, цілісні моделі знань.

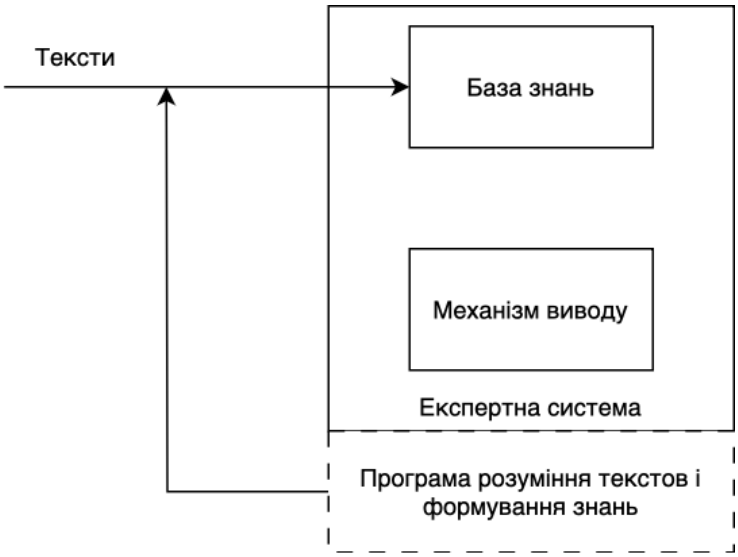

Рис. 1.11. Модель придбання знань ЕС за допомогою програми розуміння текстів

1.5 Огляд існуючих рішень

Зростаюча популярність та широкий діапазон застосування систем підтримки прийняття рішень ініціюють їх невпинне покращення та подальше інтегрування у різні сфери людської діяльності, навіть ті, де вони раніше не були представлені. Сьогоднішні системи підтримки прийняття рішень використовуються в різних галузях, включаючи, зокрема, медицину, де вони стали невід'ємною частиною технологічного обладнання лабораторій, незважаючи на те, що вони часто залишаються власністю вузького кола фахівців і не виходять за межі відповідного професійного співтовариства.

Тим не менш, відомо про деякі відкриті експертні системи, деталі реалізації яких були доступні для загального огляду. Розробники цих систем, керуючись різними мотивами, вирішили відкрити деякі елементи своїх продуктів, що дозволило вченим і інженерам по всьому світу вивчити їх архітектуру та методи роботи. Це, в свою чергу, стимулювало подальший розвиток експертних систем, сприяло обміну знаннями і досвідом у цій області.

Розглянемо одну з класичних відомих експертних систем **MYCIN**

MYCIN була однією з перших і найвідоміших експертних систем, розроблених у 1970-х роках для діагностики бактеріальних інфекцій крові та призначення антибіотиків [9].

Основні можливості. Система, заснована на аналізі внесених симптомів, здатна формувати вірогідний діагноз та пропонувати оптимальний курс медикаментозної терапії для кожного із потенційних інфекційних захворювань. Структура системи включає приблизно 600 унікальних правил, розроблених за консультацією з командою спеціалістів із інфекційних хвороб зі Стенфордського університету.

Переваги. MYCIN була зосереджена на дуже специфічній медичній предметній області (інфекційні захворювання крові), що дозволило створити високоспеціалізовану систему з глибоким розумінням області.

MYCIN використовувала систему правил для логічного виведення та знаходження рекомендацій, що забезпечувало прозорість та обґрунтованість прийнятих рішень.

Система надавала можливість взаємодії з лікарями через зручний інтерфейс, що підтримував діалогові сесії для введення даних пацієнта та отримання рекомендацій.

Недоліки. MYCIN була спрямована виключно на інфекційні хвороби крові, тому її область використання була досить обмеженою і система не могла бути застосована для інших медичних галузей без значних модифікацій.

MYCIN не була інтегрована з іншими медичними системами або базами даних, що ускладнювало обмін медичною інформацією та зменшувало її практичну цінність для медичних спеціалістів.

MYCIN пропонувала рекомендації з лікування, але не мала можливості моніторингу пацієнта та адаптації своїх рекомендацій відповідно до змін стану пацієнта.

Також розглянемо деякі відомі сучасні комплексні системи медичного обслуговування:

Aidoc

Це система підтримки прийняття рішень в медицині, яка використовує штучний інтелект та машинне навчання для аналізу медичних зображень. Розроблена компанією Aidoc, ця система допомагає лікарям швидко і точно діагностувати невідкладні стани, такі як крововилив в мозок, через своєчасний аналіз та виявлення аномалій на комп’ютерних томографіях [10].

Основні можливості. Aidoc може автоматично аналізувати великі набори даних щодо медичних зображень (використовуючи алгоритми глибокого навчання) для виявлення аномалій, що можуть вказувати на критичні стани. Система прагне до максимізації ефективності діагностичного процесу, прискорюючи час виявлення таких станів і забезпечуючи більш точну діагностику.

Переваги. Швидкість і точність. Aidoc може аналізувати медичні зображення набагато швидше, ніж спеціаліст, при цьому підтримуючи високий рівень точності.

Покращення ефективності. Aidoc допомагає розподіляти робочий час спеціалістів більш ефективно, автоматизуючи рутинні процеси аналізу зображень.

Інтеграція з існуючими системами. Aidoc легко інтегрується з існуючими системами медичної інформації, що дозволяє лікарям легко впроваджувати технологію в практику.

Недоліки. Залежність від якості даних. Як і всі системи, що базуються на машинному навчанні, Aidoc залежить від якості вхідних даних. Якщо якість сканування або зображення погана, результати можуть бути недостовірними.

Відсутність клінічної інтерпретації. Aidoc може виявляти аномалії на зображеннях, але в кінцевому підсумку потрібен лікар для інтерпретації цих результатів та встановлення остаточного діагнозу.

Можливість помилок. Хоча система і має високу точністю, існує ризик помилок або пропущених аномалій, що можуть мати серйозні наслідки для пацієнтів.

NextGen Healthcare

Це комплексна система підтримки прийняття рішень в сфері охорони здоров'я, яка надає широкий спектр сервісів і програмних продуктів. Наступні аспекти обумовлюють основні можливості, переваги та недоліки системи [11].

Основні можливості. Ця система дозволяє зберігати, управляти і відстежувати медичну інформацію пацієнтів в цифровому форматі і проводити детальний аналіз та інтеграцію з різними підсистемами охорони здоров'я.

Переваги. Інтеграція. NextGen надає цілісне рішення, що інтегрує різні аспекти охорони здоров'я, включаючи електронні медичні записи.

Доступність. Платформа доступна через хмарні сервіси, що дозволяє доступ до системи з будь-якого пристрою і в будь-який час.

Гнучкість. Можна налаштувати систему відповідно до специфіки конкретної медичної області або закладу.

Недоліки. Складність використання. Деякі користувачі відзначають, що система може бути складною для використання, особливо для нових користувачів, що не знайомі з медичними програмами.

Вартість. Ціна системи може бути досить високою, особливо для невеликих медичних закладів.

Технічна підтримка. Деякі користувачі відзначають, що вони стикалися з проблемами з технічною підтримкою, такі як повільний відгук або недостатня допомога при вирішенні проблем.

CloudMedx

Система підтримки прийняття рішень CloudMedx використовує технології штучного інтелекту та машинного навчання для аналізу клінічних даних та прогнозування результатів у різних медичних ситуаціях. Це прогресивний інструмент, який змінює підхід до оцінки стану здоров'я пацієнта, але як і будь-який інший інструмент, він має свої особливості [12].

Основні можливості. Система CloudMedx розроблена таким чином, що вона може аналізувати великі масиви даних, включаючи електронні медичні записи, лабораторні результати, записи пацієнтів та іншу інформацію. За допомогою алгоритмів машинного навчання, система визначає шаблони та кореляції, що допомагають у діагностиці та прогнозуванні перебігу захворювань.

Переваги. Ефективність аналізу. CloudMedx може аналізувати великі об'єми даних швидше та ефективніше, ніж це здатен зробити лікар.

Точність прогнозування. Використовуючи дані з різних джерел, система може прогнозувати різні медичні сценарії з високою точністю

Недоліки. Залежність від якості даних. Як і більшість систем машинного навчання, CloudMedx має бути навчена на великому обсязі високоякісних даних, щоб гарантувати точність своїх прогнозів. Це означає, що результати можуть бути спотворені у випадку некоректних, неповних або упереджених даних.

Потреба в постійному оновленні. Медична наука є динамічною галуззю, в якій постійно з'являються нові дослідження та відкриття. Це означає, що система має регулярно оновлюватись і перенавчатися, щоб враховувати актуальні дані.

РОЗДІЛ 2: НЕЧІТКІ МОДЕЛІ ДЛЯ ДІАГНОСТУВАННЯ СТАНУ ЗДОРОВ'Я ПАЦІЄНТА

2.1 Елементи теорії нечітких множин

Як відомо, в класичній теорії множин непорожня підмножина A універсальної множини X визначається характеристичною функцією

$$\chi_A(x) = \begin{cases} 1, x \in A \\ 0, x \notin A \end{cases}$$

Але як описати, наприклад, множину великих або малих чисел з проміжку [a,b]? Можна жорстко поділити відрізок на ліву та праву частини і числа, які належать до лівої частини відрізка вважати малими, а до правої – великими. Ще один природній поділ чисел відрізка на великі та малі – класифікувати їх, відповідаючи на питання: "Якою мірою те чи інше число можна віднести до "великого" ("малого")"?

Аналогічна картина спостерігається, коли хочуть описати поняття чистої або забрудненої води, холодного і гарячого, світлого і темного і т. д.

За допомогою нечітких множин можна формально визначати неточні і багатозначні поняття, такі як "висока температура", "молода людина", "середній зріст" або "велике місто". Перед формулюванням визначення нечіткої множини необхідно задати так звану область (або універсум).

Визначення. Нехай X – довільна непуста множина. Нечіткою підмножиною A (позначається $A \subseteq X$), називається множина пар

$$A = \{(x, \mu_A(x)), x \in X\},$$

де

$$\mu_A: X \to [0,1] –$$

функція належності нечіткої множини A. Ця функція приписує кожному елементу $x \in X$ міру його належності до нечіткої множини A [13].

При цьому:

1) $\mu_A(x) = 1$ означає повну належність елемента x до нечіткої множини A, тобто $x \in A$;

2) $\mu_A(x) = 0$ означає відсутність належності елемента x до нечіткої множини A, тобто $x \notin A$;

3) $0 < \mu_A(x) < 1$ означає часткову належність елемента x до нечіткої множини A.

В літературі використовується символьне подання нечітких множин. Якщо X – це простір зі скінченною кількістю елементів, тобто $X = \{x_1, x_2, \dots, x_n\}$, то нечітка множина $A \subseteq X$ може бути записана у вигляді:

$$A = \mu_A(x_1)/x_1 + \dots + \mu_A(x_n)/x_n.$$

Знак "/" в цьому записі означає приписування конкретним елементам $x_1, \dots, x_n$ мір належності $\mu_A(x_1), \dots, \mu_A(x_n)$. Іншими словами, запис $\mu_A(x_i)/x_i$, $i = 1, \dots, n$ означає пару $(x_i, \mu_A(x_i))$, $i = 1, \dots, n$.

Точно так, знак "+" означає не операцію додавання, а інтерпретується як множинна сума елементів.

Якщо X – це простір з нескінченною кількістю елементів, то нечітка множина $A \subseteq X$ символічно записується у вигляді:

$$\int_X \mu_A(x) / x dx.$$

2.2 Операції на нечітких множинах

Визначення. Перетином двох нечітких множин $A, B \subseteq X$ називається нечітка множина $A \cap B$ з функцією належності $\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$ для кожного $x \in \boldsymbol{X}$.

Сума нечітких множин A і B – нечітка множина $C = A \cup B$, що визначається функцією належності $\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$ для кожного $x \in X$.

Доповненням нечіткої множини $A \subseteq X$ називається нечітка множина з функцією належності $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$ для кожного $x \in X$.

Визначення. Декартовий добуток нечітких множин $A \subseteq X$ і $B \subseteq Y$ позначається $A \times B$ і визначається як

$$\mu_{A\times B}(x,y) = \mu_A(x) \wedge \mu_B(y) = \min(\mu_A(x), \mu_B(y))$$

або

$$\mu_{A\times B}(x,y) = \mu_A(x)\ \mu_B(y)$$

для кожного $x \in X$ і $y \in Y$.

Приклад. Припустимо, що $X = \{1,2,3,4,5,6,7\}$ і $A = 0{,}9/3+1/4+0{,}6/6$, $B = 0{,}7/3+1/5+0{,}4/6$. У відповідності з визначеннями одержуємо

$$A \cap B = 0{,}7/3+0{,}4/6.$$

$$A \cup B = 0{,}9/3+1/4+1/5+0{,}6/6.$$

Припустимо, $X = \{1,2,3,4,5,6\}$, а також

$$A = 0{,}3/2+1/3+0{,}7/5+0{,}9/6\ .$$

У відповідності з визначенням доповненням множини A є множина

$$\overline{A} = 1/1+0{,}7/2+1/4+0{,}3/5+0{,}1/6\ .$$

Звернемо увагу на те, що

$$A\cap \overline{A} = 0{,}3/2+0{,}3/5+0{,}1/6 \neq \varnothing, \text{ а також}$$

$$A\cup \overline{A} = 1/1+0{,}7/2+1/3+1/4+0{,}7/5+0{,}9/6 \neq X\ .$$

Можна показати, що визначені вище операції на нечітких множинах задовольняють властивостям комутативності, асоціативності і дистрибутивності, крім того, правилам де Моргана і поглинання. Однак, у випадку нечітких множин не виконується умова доповнювальності, тобто

$$A\cap \overline{A} \quad \neq \varnothing$$

$$A\cup \overline{A} \quad \neq X$$

Приклад. Припустимо, що $X = \{2,4\}$, $Y = \{2,4,6\}$ і

$$A = 0{,}5/2+0{,}9/4,\ B = 0{,}3/2+0{,}7/4+0{,}1/6.$$

Тоді

$$A\times B = 0{,}3/(2,2)+0{,}5/(2,4)+0{,}1/(2,6)+0{,}3/(4,2)\ +0{,}7/(4,4)+0{,}1/(4,6).$$

2.3 Нечіткі відношення

Одним із основних понять теорії нечітких множин вважається поняття нечіткого відношення. Ці відношення дозволяють формалізувати нечіткі твердження типу

"*x* майже рівне *y*" або "*x* значно більше за *y*".

Визначення. Нечітким відношенням *R* між двома непустими множинами (чіткими) *X* та *Y* будемо називати нечітку множину, визначену на декартовому добутку $X\times Y$, тобто

$$R \subseteq X\times Y = \{(x,y): x\in X, y\in Y\}.$$

Іншими словами, нечітке відношення – множина пар

$$R = \{((x,y), \mu_R(x,y))\}, \text{ де } \mu_R: X\times Y\to[0,1]$$

– це функція належності, яка кожній парі (x,y) приписує її міру належності $\mu_R(x,y)$, яка інтерпретується як сила зв'язку між елементами $x \in X$ і $y\in Y$. У відповідності з прийнятими домовленостями, нечітке відношення можна подати у вигляді

$$R = \sum_{X\times Y} \frac{\mu_R(x,y)}{(x,y)} \quad \text{або} \quad R = \int_{X\times Y} \frac{\mu_R(x,y)}{(x,y)} \, dxdy.$$

Визначення. Композицією типу sup-*T* нечітких відношень $R\subseteq X\times Y$ і $S\subseteq Y\times Z$ називається нечітке відношення $R\circ S \subseteq X\times Z$ з функцією належності

$$\mu_{R\circ S}(x,z) = \sup_{y\in Y}\left\{[\mu_R(x,y) \overset{T}{*} \mu_s(y,z)]\right\}.$$

Як бачимо, конкретна форма функції належності $\mu_{R\circ S}(x,z)$ композиції $R\circ S$ залежить від *T*-норми. Якщо *T*-норма визначається як $T(a,b)=\min(a,b)$, то вище приведену рівність можна переписати у вигляді

$$\mu_{R\circ S}(x,z) = \sup_{y\in Y}\left\{\min[\mu_R(x,y), \mu_s(y,z)]\right\}.$$

Ця формула відома в літературі під назвою “композиція типу sup-min”. Якщо множина ***Y*** має скінчену кількість елементів, то композиція типу sup-min зводиться до композиції типу max-min в формі

$$\mu_{R\circ S}(x,z)=\max_{y\in Y}\left\{\min[\mu_R(x,y),\mu_s(y,z)]\right\}.$$

Приклад. Нехай відношення *R* і *S* задані матрицями

$$R=\begin{bmatrix}0,2 & 0,5\\ 0,6 & 1\end{bmatrix},S=\begin{bmatrix}0,3 & 0,6 & 0,8\\ 0,7 & 0,9 & 0,4\end{bmatrix}$$

Причому $X=\{x_1,x_2\}$, $Y=\{y_1,y_2\}$, $Z=\{z_1,z_2,z_3\}$.

Композиція типу max-min відношень *R* і *S* має вигляд

$$Q=R\circ S\begin{bmatrix}0,2 & 0,5\\ 0,6 & 1\end{bmatrix}\circ\begin{bmatrix}0,3 & 0,6 & 0,8\\ 0,7 & 0.9 & 0,4\end{bmatrix}=\begin{bmatrix}q_{11} & q_{12} & q_{13}\\ q_{21} & q_{22} & q_{23}\end{bmatrix}$$

де

q_{11}=max[min(0,2;0,3),min(0,5;0,7)]=0,5,

q_{12}=max[min(0,2;0,6),min(0,5;0,9)]=0,5,

q_{13}=max[min(0,2;0,8),min(0,5;0,4)]=0,4,

q_{21}=max[min(0,6;0,3),min(1;0,7)]=0,7,

q_{22}=max[min(0,6;0,6),min(1;0,9)]=0,9,

q_{23}=max[min(0,6;0,8),min(1;0,4)]=0,6,

тому

$$Q=\begin{bmatrix}0,5 & 0,5 & 0,4\\ 0,7 & 0,9 & 0,6\end{bmatrix}.$$

Для практичних застосувань дуже важлива композиція нечіткої множини з нечітким відношенням. Розглянемо нечітку множину $A \subseteq X$ і нечітке відношення $R \subseteq \boldsymbol{X} \times \boldsymbol{Y}$ з функціями належності $\mu_A(x)$, $\mu_R(x,y)$.

Визначення. Композиція нечіткої множини $A \subseteq X$ і нечіткого відношення $R \subseteq X \times Y$ позначається $A \circ R$ і визначається як нечітка множина $B \subseteq Y$ з функцією належності

$$\mu_B(y) = \sup_{x \in X} \{[\mu_A(x) \overset{T}{*} \mu_R(x,y)]\}.$$

Конкретна форма цієї композиції залежить від T-норми і від властивостей множини $\boldsymbol{X}$. Виділяють 4 випадки:

1) якщо $T(a,b) = \min(a,b)$, то одержуємо композицію типу sup-min

$$\mu_B(y) = \sup_{x \in X} \{\min[\mu_A(x), \mu_R(x,y)]\};$$

2) якщо $T(a,b) = \min(a,b)$ і X – множина зі скінченою кількістю елементів, то одержуємо композицію типу max-min

$$\mu_B(y) = \max_{x \in X} \{\min[\mu_A(x), \mu_R(x,y)]\};$$

3) якщо $T(a,b) = a \cdot b$, то одержуємо композицію типу sup-добуток

$$\mu_B(y) = \sup_{x \in X} \{\mu_A(x) \mu_R(x,y)\};$$

4) якщо $T(a,b) = a \cdot b$ і X – множина зі скінченою кількістю елементів, одержуємо композицію типу max-добуток.

Приклад 25. Нехай $X=\{x_1,x_2,x_3\}$, $Y=\{y_1,y_2\}$ і A має вигляд

$$A = 0{,}4/x_1 + 1/x_2 + 0{,}6/x_3,$$

тоді як відношення R задане матрицею

$$R = \begin{bmatrix} 0{,}5 & 0{,}7 \\ 0{,}2 & 1 \\ 0{,}9 & 0{,}3 \end{bmatrix}$$

Композиція $A \circ R$ типу max-min тоді – це нечітка множина B

$$B = \mu_B(y_1)/y_1 + \mu_B(y_2)/y_2 .$$

Причому

$$\mu_B(y_1) = \max\{\min(0{,}4;0{,}5), \min(1;0{,}2), \min(0{,}6;0{,}9)\} = 0{,}6,$$

$$\mu_B(y_2) = \max\{\min(0{,}4;0{,}7), \min(1;1), \min(0{,}6;0{,}3)\} = 1.$$

Тому $B = 0{,}6/y_1 + 1/y_2$.

2.4 Системи нечіткого логічного виведення

У бінарній логіці істинність висловлювання визначається на основі істинності його складових. Процес виведення задається за допомогою схеми: вище горизонтальної лінії зазначені усі допоміжні вхідні висловлювання, а під нею – висловлювання-висновок. Ця схема базується на принципі: якщо всі висловлювання вище лінії є істинними, то висловлювання під лінією також вважається істинним, оскільки з істинних висловлювань виводиться лише істинний результат.

Далі великими буквами A і B будемо записувати твердження, а не нечіткі множини.

Основне правило виведення в бінарній логіці це правило *modus ponens.*

Визначення. Правило *modus ponens* визначається наступною схемою:

якщо A та $A \to B$ то B.

У контексті нечіткої логіки узагальнене правило виведення *modus ponens* визначається наступною схемою [14]:

якщо x це A' і з того, що x це A випливає, що y це B, то y це B'

де $A, A' \subseteq X$ і $B, B' \subseteq Y$ – нечіткі множини, x, y – так звані *лінгвістичні змінні*.

Лінгвістичними називаються змінні, значеннями яких є слова або висловлювання на природній мові. Як приклад можна привести вирази типу “мала швидкість”, “помірна температура” або “молода людина”. Подібні вирази можна формалізувати приписуванням їм деяких нечітких множин.

Нехай x та y позначають вхідну та вихідну лінгвістичну змінну, а A і B позначають нечіткі множини, які визначають елементи терм-множин для змінних x та y відповідно.

Розглянемо кілька способів задання систем нечіткого логічного виведення [15-16].

Найбільш спрощеною нечіткою системою логічного виведення є наступна:

вхід (x);

якщо $x \in A$, *то* $y \in B$;

вихід (y).

Правило «*якщо x належить A*, *то y належить B*» розглядається як нечітке висловлювання типу $A \rightarrow B$. Таке висловлювання ототожнюється з нечітким відношенням на декартовому добутку просторів X (для вхідної змінної) та Y (для вихідної змінної). При подачі на вхід нечіткої множини A' на виході системи з одного правила формується нечітка множина, функція належності якої обчислюється за допомогою наступної формули:

$$B'(y) = \max_{x \in X} \min\left(A'(x), \min\{A(x), B(y)\}\right), y \in Y.$$

Більш складну систему логічного виведення можна задати наступною схемою:

вхід (x);

якщо $x \in A_1$, *то* $y \in B_1$;

якщо $x \in A_2$, *то* $y \in B_2$;

. . .

якщо $x \in A_m$, *то* $y \in B_m$;

вихід (y),

де A_i і B_i – деякі нечіткі множини, $i = \overline{1, m}$.

Для складних нечітких експертних систем одного вхідного параметра недостатньо. Отже, раніше описані схеми можна адаптувати таким чином, щоб система працювала з декількома вхідними параметрами:

вхід ($x_1, x_2, \ldots, x_n$);

якщо $x_1 \in A_{11} \wedge x_2 \in A_{12} \wedge \ldots \wedge x_n \in A_{1n}$ *то* $y \in B_1$;

якщо $x_1 \in A_{21} \wedge x_2 \in A_{22} \wedge \ldots \wedge x_n \in A_{2n}$ *то* $y \in B_2$;

. . .

якщо $x_1 \in A_{m1} \wedge x_2 \in A_{m2} \wedge \ldots \wedge x_n \in A_{mn}$ *то* $y \in B_m$;

вихід (y),

де x_j, $j = \overline{1,n}$ – лінгвістичні змінні на вході, y – вихідна лінгвістична змінна, A_{ij} та B_i – деякі нечіткі множини.

У контексті задання правил з великої кількістю вхідних змінних пряме використання правил виведення у формі відношень стає нездійсненним, на відміну від використання правил з єдиною вхідною змінною. Тому застосовується альтернативний підхід до визначення виходу, що ґрунтується на використанні так званих рівнів істинності для правил певного типу:

якщо $x_1 \in A_{i1} \wedge x_2 \in A_{i2} \wedge \ldots \wedge x_n \in A_{in}$ то $y \in B_i$.

При наявності двох вхідних змінних x_1 і x_2, алгоритм виконуватиметься за наступною послідовністю дій:

1. Для кожного правила R, $i = 1, \ldots, m$ визначаємо його рівень істинності

$$\alpha_i = \min\left[\max_{X_1}(A_1'(x_1) \wedge A_{i1}(x_1)), \max_{X_2}(A_2'(x_2) \wedge A_{i2}(x_2))\right];$$

2. Для кожного правила виведення визначаємо його вихід

$$B_i'(y) = \min(\alpha_i, B_i(y));$$

3. Обчислюємо загальний вихід

$$B'(y) = \max(B_1', B_2', \ldots, B_m').$$

Цей підхід відомий як max-min метод логічного висновку Мамдані, де імплікація трактується як операція отримання мінімуму, а агрегація результатів правил – як операція отримання максимуму. Застосування нечітких методів логічного виведення в комбінації із відповідними методами навчання може демонструвати вищу ефективність порівняно з нейронними мережами. Описані підходи до визначення рівнів істинності нечітких подій не вважаються остаточними, але вони надають можливості для інтеграції додаткових оцінок, роблячи аналіз за допомогою нечітких систем логічного виведення, більш точним та адаптивним.

2.5 Алгоритм знаходження нечіткої моделі хвороби

Для розробки алгоритму визначення нечіткої моделі хвороби пацієнта були враховані наступні особливості предметної області – гомеопатії:

1. Подібне лікується подібним. Це означає, що для лікування хвороби у пацієнта слід застосовувати гомеопатичний препарат (медичний агент), який викликає подібні симптоми.
2. Різне розведення одного й того ж гомеопатичного препарату лікує різні хвороби. Це означає, що при розробці алгоритму потрібно враховувати симптоматику різних розведень препарату.
3. Модальність симптоматики. Це означає, що симптоми захворювання можуть бути яскраво вираженими, можуть бути ледь помітними, можуть бути помірними і т.д. Це означає, що однією з адекватних моделей для опису симптомів може бути саме нечітка модель у вигляді систем нечіткого логічного виведення.

Завдання полягає в створенні системи, яка за заданою симптоматикою пацієнта та з врахуванням всіх особливостей предметної області знаходить нечітку модель хвороби пацієнта у просторі хвороб.

В основі будь-якої системи підтримки прийняття рішень знаходяться три ключові компоненти: база знань, база даних і система логічного виведення. Ці компоненти, доповнені інтерфейсом для спілкування з користувачем, створюють основну структуру системи, яку можна вважати експертною. База знань забезпечує загальні знання, що стосуються визначеної проблемної області (наприклад, симптоми, хвороби, правила їх взаємодії). Функція бази даних полягає в зберіганні інформації про контекст системи прийняття рішень (наприклад, дані про лікарів, пацієнтів, історію захворювань і т. д.). Завданням системи логічного виведення є використання правил для формування нечітких висновків – моделей хвороб пацієнта.

Алгоритм знаходження нечіткої моделі хвороби полягає у виконанні наступної послідовності кроків:

1. Проведення опитування пацієнта.
2. За знайденими симптомами знаходимо всі можливі медичні агенти із подібною симптоматикою.
3. Визначаємо всі не розпізнані симптоми. Якщо таких симптомів не виявлено, то переходимо на крок 4. В іншому випадку переходимо на крок 2.

4. В базі знань знаходимо всі нечіткі правила, які містять задані симптоми і формуємо систему нечіткого логічного виведення.
5. За модифікованим правилом Мамдані знаходимо вихід системи у вигляді нечіткого діагнозу пацієнта (нечітка множина у просторі хвороб).
6. Кінець.

2.6 Методи побудови нечітких правил системи

Розглянемо задачу визначення хвороби пацієнта відповідно до поточних симптомів пацієнта.

Для вирішення цієї задачі в контексті нечіткої логіки потрібно побудувати систему нечіткого логічного виведення, що використовує залежності між хворобами, симптомами та їх рівнями (модальністю).

Для прикладу, нехай кількість хвороб 3 а кількість симптомів 3.

Позначимо множину хвороб D:

$$D = \{\text{Хвороба\_1}, \text{Хвороба\_2}, \text{Хвороба\_3}\}.$$

Також введемо множину симптомів S і терм-множину рівнів симптомів L:

$S = \{\text{Симптом\_1}, \text{Симптом\_2}, \text{Симптом\_3}\}$, $L = \{\text{Низький},\ \text{Середній},\ \text{Високий}\}$.

Тоді належність симптома s з відповідним рівнем l до хвороби d можна описати за допомогою множини пар R_d:

$$R_d = \{\ (s, l) \mid s \in S, l\ \in L\}.$$

Таку залежність можна подати у табличному вигляді, де рядки відповідають хворобам, а колонки симптомам (Таблиця 2.1):

Таблиця 2.1

Відповідності рівнів конкретних симптомів до хвороби

Хвороби	**Симптом_1**	**Симптом_2**	**Симптом_3**
Хвороба_1	Середній	Високий	-
Хвороба_2	Низький	Середній	Середній
Хвороба_3	-	Високий	Низький

Додатково введемо терм-множину рівнів визначеності хвороби M:

$$M = \{\text{Малий},\ \text{Помірний},\ \text{Великий}\}.$$

Використовуючи вищезазначену початкову інформацію про предметну область і залежності в ній, можна безпосередньо приступати до нечіткого моделювання. Перш за все, введемо лінгвістичні змінні та функції належності для поточних рівнів симптомів і рівнів визначеності хвороби:

- {Поточний_Симптом_1, Поточний_Симптом_2, Поточний_Симптом_3} – поточні симптоми.
- {Рівень_Хвороба_1, Рівень_Хвороба_2, Рівень_Хвороба_3} – рівні хвороб.

Для кожного поточного симптома та рівня хвороби побудуємо наступні нечіткі модифіковані трикутні функції належності [17], як зображено на рис. 2.1.

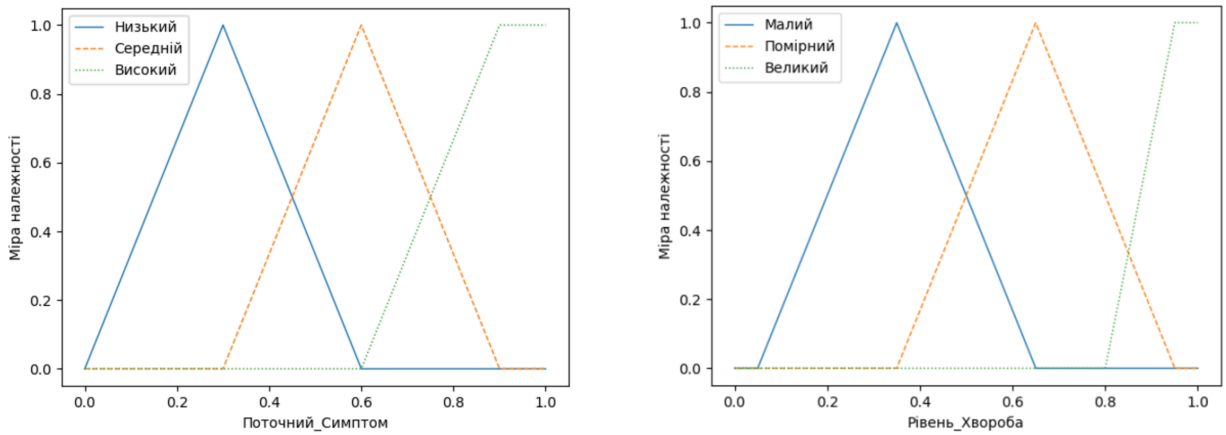

Рис. 2.1. Функції належності поточного симптома та рівня хвороби

Аналогічно такі функції можна задати трійками значень, а саме:

- Поточний_Симптом: Низький – (0, 0.3, 0.6); Середній – (0.3, 0.6, 0.9); Високий – (0. 6, 0.9, 0.9).
- Рівень_Хвороба: Малий – (0.05, 0.35, 0.65); Помірний – (0.35, 0.65, 0.95); Великий – (0.8, 0.95, 0.95).

Наступний крок – це побудова нечітких правил, саме від них залежить якість і точність виведення в системах нечіткого логічного виведення. Ці правила визначають, як система реагує на різноманітні вхідні дані та обставини, тому їх правильне формування є ключовим для отримання бажаних результатів.

Розглянемо декілька способів задання правил:

Метод 1

В контексті нашої проблеми визначення хвороби пацієнта за симптомами і початкової інформації про залежність симптомів і хвороб має сенс використатити інтуїтивно зрозумілі прямі правила наступного формату:

1. Якщо (Поточний_Симптом_1 є R_1(Поточний_Симптом_1)) І (Поточний_Симптом_2 є R_1(Поточний_Симптом_2)) … І (Поточний_Симптом_К є R_1(Поточний_Симптом_К)) ТО Рівень_Хвороба_1 є Великий;

2. Якщо (Поточний_Симптом_1 є R_2(Поточний_Симптом_1))) І (Поточний_Симптом_2 є R_2(Поточний_Симптом_2)) … І (Поточний_Симптом_К є R_2(Поточний_Симптом_К)) ТО Рівень_Хвороба_2 є Великий;

…

N. Якщо (Поточний_Симптом_1 є R_N(Поточний_Симптом_1)) І (Поточний_Симптом_2 є R_N(Поточний_Симптом_2) … І (Поточний_Симптом_К є R_N(Поточний_Симптом_К)) ТО Рівень_Хвороба_N є Великий,

де N – кількість хвороб, K – кількість симптомів, $R_i(x)$ – функція, що визначає рівень симптому $x \in S$ для хвороби $i = 1..\text{N}$.

Отже, система правил нечіткого логічного виведення матиме вигляд в нашому випадку буде такою:

1. Якщо (Поточний_Симптом_1 є Середінй) І (Поточний_Симптом_2 є Високий) ТО Рівень_Хвороба_1 є Великий;

2. Якщо (Поточний_Симптом_1 є Низький)) І (Поточний_Симптом_2 є Середній) І (Поточний_Симптом_3 є Середній) ТО Рівень_Хвороба_2 є Великий;

3. Якщо (Поточний_Симптом_2 є Високий) І (Поточний_Симптом_3 є Низький) ТО Рівень_Хвороба_3 є Великий.

Для такої системи можна використати різні механізми виведення, наприклад, виведення Мамдані [18-19].

Даний підхід простий і зрозумілий, водночас, при такій конструкції системи правил можуть виникати проблеми. Наприклад, нерідко під час опитування пацієнта можна припуститись помилок – пацієнт може надати лише частину симптомів і не надати їх на вхід для обробки програмою.

Оскільки, особливість роботи нечітких правил з оператором "І" під час процесу виведення така, що якщо хоча б одна з вхідних лінгвістичних змінних не відповідає терму в правилі, то результат для цього конкретного правила буде дуже низьким, оскільки мінімальний оператор відображає найнижчий ступінь належності серед змінних. Таку проблему можна спробувати вирішити додатковими, компенсуючими правилами, проте коли кількість вхідних змін зростає, кількість комбінацій правил може швидко збільшуватися, призводячи до проблеми «прокляття розмірності» [20].

Для підвищення ефективності процесу визначення хвороби за нечіткими правилами пропонується розглянути дещо інший спосіб побудови правил системи.

Метод 2

Будемо оперувати нечіткими поняттями, що описують кількість коректно наданих симптомів відносно загальної кількості симптомів хвороби.

Введемо додаткову терм-множину відносної кількості коректно наданих симптомів C:

$$C = \{\text{Низька\_Відносна\_Кількість}, \text{Середня\_Відносна\_Кількість}, \text{Велика\_Відносна\_Кількість}\}.$$

Варто зазначити, що значення лінгвістичних змінних Відносна_Кількість_i, $i = 1..\text{N}$ обчислюється як кількість наявних симптомів на вході, які є у пацієнта для Хвороби_i, поділене на кількість всіх симптомів Хвороби_i.

Введемо лінгвістичні змінні та функції належності для відносної кількості коректно наданих симптомів для згаданих вище хвороб:

- $\{\text{Відносна\_Кількість\_1}, \text{Відносна\_Кількість\_2}, \text{Відносна\_Кількість\_3}\}$ – відносні кількості хвороб.
- Нечіткі модифіковані трикутні функції належності (відповідно до терм-множини, рис. 2.2).

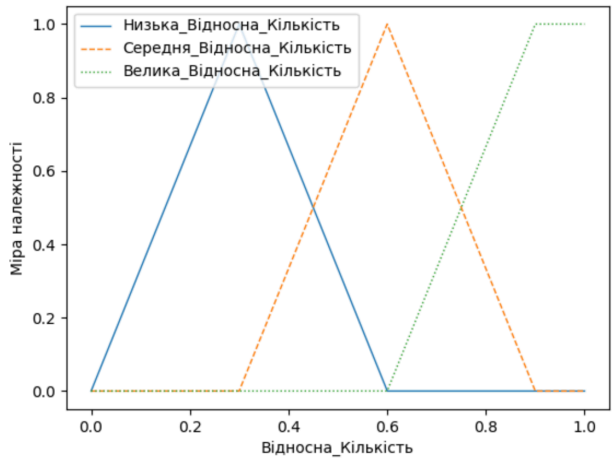

Рис. 2.2. Функція належності лінгвістичних змінних відносної кількості коректно наданих симптомів хвороби

Такі функції змінної Відносна_Кількість можна задати трійками значень, а саме:

- Низька_Відносна_Кількість – (0, 0.3, 0.6);
- Середня_Відносна_Кількість – (0.3, 0.6, 0.9);
- Висока_Відносна_Кількість – (0. 6, 0.9, 0.9).

Тоді нечіткі правила логічного виведення будуть мати такими:

1.1 Якщо (Відносна_Кількість_1 є Висока_Відносна_Кількість) ТО Рівень_Хвороба_1 є Великий;

1.2 Якщо (Відносна_Кількість_1 є Середня_Відносна_Кількість) ТО Рівень_Хвороба_1 є Помірний;

1.3 Якщо (Відносна_Кількість_1 є Низька_Відносна_Кількість) ТО Рівень_Хвороба_1 є Малий;

…

N.1 Якщо (Відносна_Кількість_*N* є Висока_Відносна_Кількість ТО Рівень_Хвороба_*N* є Великий;

N.2 Якщо (Відносна_Кількість_*N* є Середня_Відносна_Кількість ТО Рівень_Хвороба_*N* є Помірний;

N.3 Якщо (Відносна_Кількість_N є Низька_Відносна_Кількість ТО Рівень_Хвороба_N є Малий,

де N – кількість хвороб, Відносна_Кількість_i, $i = 1..\text{N}$ відносні кількості коректних симптомів хвороб

Отже, система правил нечіткого виведення в нашому випадку матиме вигляд:

1.1 Якщо (Відносна_Кількість_1 є Висока_Відносна_Кількість) ТО Рівень_Хвороба_1 є Великий;

1.2 Якщо (Відносна_Кількість_1 є Середня_Відносна_Кількість) ТО Рівень_Хвороба_1 є Помірний;

1.3 Якщо (Відносна_Кількість_1 є Низька_Відносна_Кількість) ТО Рівень_Хвороба_1 є Малий;

2.1 Якщо (Відносна_Кількість_2 є Висока_Відносна_Кількість) ТО Рівень_Хвороба_2 є Великий;

2.2 Якщо (Відносна_Кількість_2 є Середня_Відносна_Кількість) ТО Рівень_Хвороба_2 є Помірний;

2.3 Якщо (Відносна_Кількість_2 є Низька_Відносна_Кількість) ТО Рівень_Хвороба_2 є Малий;

3.1 Якщо (Відносна_Кількість_3 є Висока_Відносна_Кількість) ТО Рівень_Хвороба_3 є Великий;

3.2 Якщо (Відносна_Кількість_3 є Середня_Відносна_Кількість) ТО Рівень_Хвороба_3 є Помірний;

3.3 Якщо (Відносна_Кількість_3 є Низька_Відносна_Кількість) ТО Рівень_Хвороба_3 є Малий.

Порівняння описаних систем логічного виведення

Порівняємо два описаних вище типи систем з точки зору правильності визначення хвороби в умовах, коли картина симптоматики неповна (при опитуванні пацієнта визначені не всі симптоми).

Для тестування скористаємось набором даних [21]. Використовуючи ці дані, підготуємо наступні спотворені 11 наборів, кожен з яких містить $P = 2650$ прикладів

хвороб і 370 потенційно спотворених значень симптомів для кожного з прикладів хвороби (Таблиця 2.2).

Оскільки вихід системи нечіткого логічного виведення для кожного прикладу хвороби є набір пар (міра, Рівень_Хвороба_i), $i = 1..N$ з потенційно однаковими значеннями міри, введемо функцію γ_j, $j = 1..P$ для обчислення дійсного значення правильно визначеного j прикладу хвороби.

$$\gamma_j = \begin{cases} 0.0, & \text{якщо хвороби } j \text{ немає серед пар з максимальною мірою} \\ \dfrac{1.0}{T}, & \text{де } T \text{ кількість пар з однаковою максимальною мірою} \end{cases}$$

Тоді коректність роботи системи σ на тому чи іншому наборі даних будемо визначати як суму правильно визначених прикладів хвороб γ_j відносно загальної кількості прикладів хвороб у наборі, враховуючи можливість отримання однакової міри під час роботи програми:

$$\sigma = \frac{\sum_{j=1}^{P} \gamma_j}{P}.$$

Таблиця 2.2

Опис наборів тестових даних і результатів визначення хвороби кожним методом

Набір	**% неспотверених симптомів**	**σ 1 метод, %**	**σ 2 метод, %**	**$\Delta\sigma$, %**
data_50	50	9.06	22.84	13.78
data_55	55	11.24	23.62	12.38
data_60	60	12.53	24.76	12.23
data_65	65	14.31	24.87	10.56
data_70	70	15.76	25.84	10.08
data_75	75	19.53	27.95	8.42
data_80	80	21.76	29.44	7.68
data_85	85	26.13	34.24	8.11
data_90	90	31.96	40.70	8.74
data_95	95	39.48	48.78	9.3
data_100	100	54.36	54.90	0.54

Додатково побудуємо графіки відповідно до даних з таблиці (рис. 2.3).

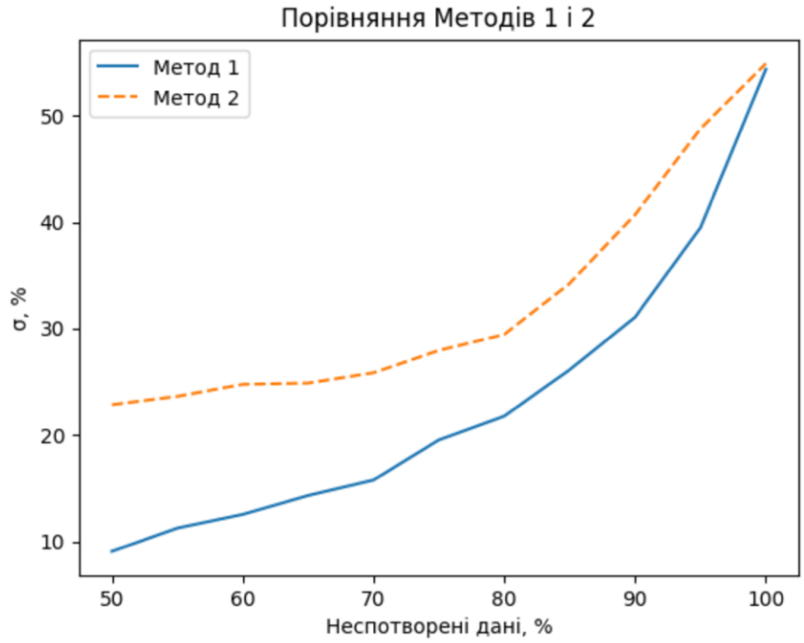

Рис. 2.3. Порівняння графіків σ для Метод 1 і Метод 2

Обидві розглянуті системи нечіткого виведення гнучко реагували на нечітку інформацію, що подавалась на вхід, та адекватно знаходили діагнози на основі наявних даних. Вдосконалена система, що включає нові лінгвістичні змінні та правила, показала себе дещо ефективніше, ніж традиційна, особливо при наявності обмеженої чи спотвореної інформації.

РОЗДІЛ 3: ЗНАХОДЖЕННЯ ОЦІНОК ДОСТОВІРНОСТІ ПРИ НЕЧІТКОМУ МОДЕЛЮВАННІ

Як уже зазначалося, підхід до побудови нечітких моделей математичних систем з входами та виходами широко використовується в експертних діагностичних системах, при проектуванні контролерів, при розробці керуючих пристроїв для побутової техніки тощо.

Виникає закономірне питання про достовірність виходу таких нечітких моделей. Це питання є досить актуальним, особливо в діагностичних системах. Для того, щоб відповісти на ці питання, запропоновано підхід, що дозволяє отримати числову характеристику виходу системи нечіткого висновку. Цю числову характеристику будемо називати достовірністю виходу нечіткої системи логічного виведення або просто достовірністю виходу системи нечітких правил.

Тому в даному розділі пропонується підхід до знаходження чисельних оцінок достовірності нечітких знань у системах нечіткого висновку. Оригінальність підходу полягає в тому, що довільна математична система типу «вхід-вихід» моделюється засобами нечіткої логіки, за правилом Мамдані знаходиться вихід такої системи і обчислюється його достовірність. Приводиться приклад обчислення достовірності для системи зі скінченними просторами для моделювання нечіткості [22].

3.1 Побудова нечітких правил

Один із підходів до побудови нечітких правил запропонований у [23]. Він полягає в наступному. Нехай створюється система нечітких правил з трьома входами та одним виходом на основі навчаючих даних, які представляються множиною

$$(x_1(i), x_2(i), x_3(i); d(i)), i = 1, 2, ..., m,$$

де $x_1(i), x_2(i), x_3(i)$ – входи, $d(i)$ – виходи правил нечіткої системи логічного виведення.

Алгоритм розв’язання цієї задачі полягає у виконанні наступної послідовності кроків:

1. Для входів та виходу визначаємо області допустимих значень – простори $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3$ та $\boldsymbol{Y}$ відповідно. Розбиваємо простори входів і виходу на три області і для кожної з них задаємо функції належності нечітких величин A_{ij}, B_i. Приклад такого розбиття приведений на рис. 3.1.

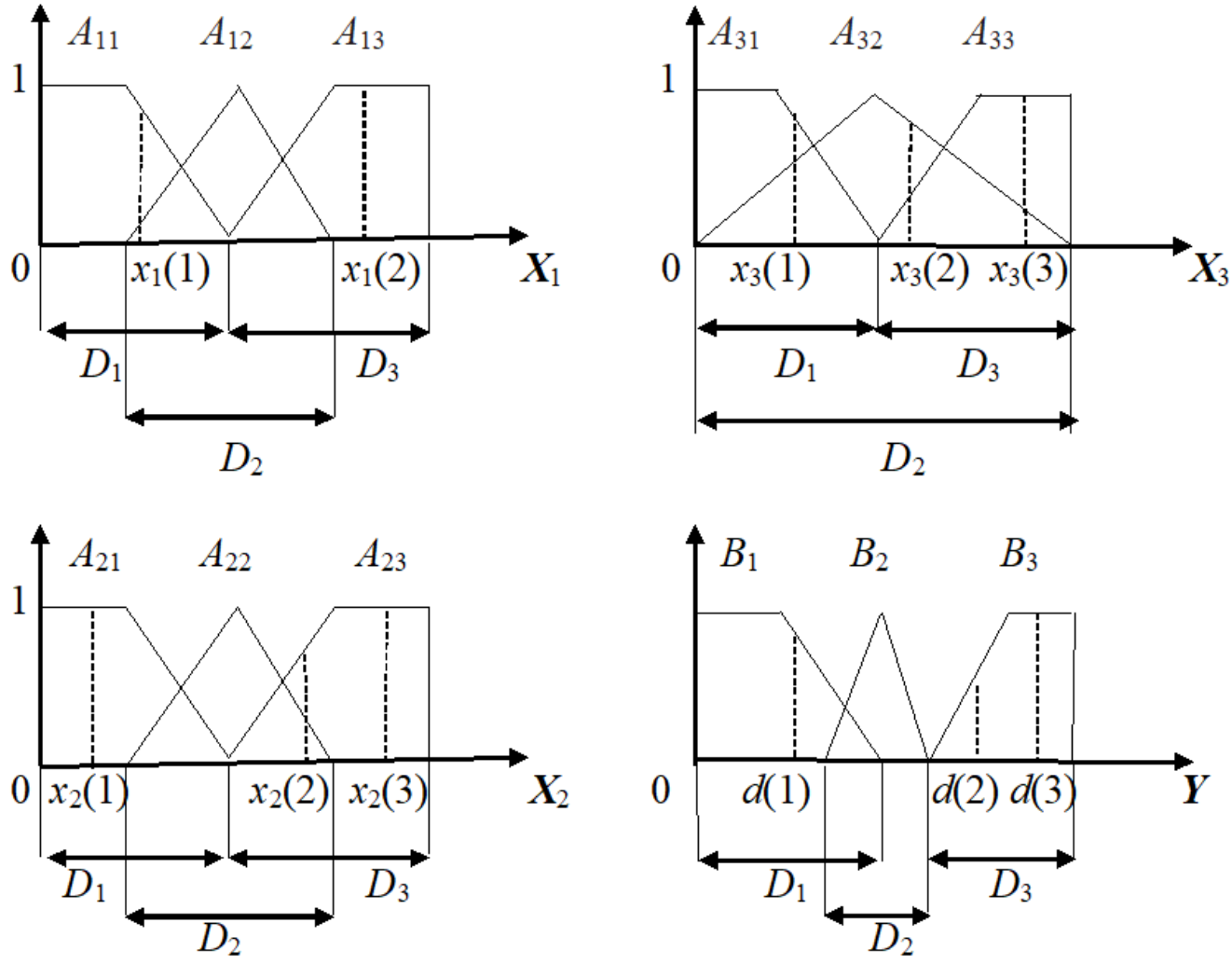

Рис. 3.1. Приклад розбиття просторів з функціями належності

Зрозуміло, що приведене на рисунку розбиття просторів є одним з прикладів такого розбиття. Те саме стосується і функцій належності. Тобто, можна запропонувати інші способи розбиття просторів та інші функції належності. Крім того, наведене розбиття визначає так звані терм-простори $\boldsymbol{T}_1$, $\boldsymbol{T}_2$, $\boldsymbol{T}_3$, $\boldsymbol{T}_4$, елементами яких є нечіткі поняття в просторах $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3$ та $\boldsymbol{Y}$. Зокрема,

$\boldsymbol{T}_1 = \{A_{11}, A_{21}, A_{31}\}$, $\boldsymbol{T}_2 = \{A_{21}, A_{22}, A_{23}\}$, $\boldsymbol{T}_3 = \{A_{31}, A_{32}, A_{33}\}$, $\boldsymbol{T}_4 = \{B_1, B_2, B_3\}$.

2. Будуємо нечіткі правила за навчаючими даними (рис. 3.1):

$$(1)\ (x_1(1), x_2(1), x_3(1); d(1)),$$
$$(2)\ (x_1(2), x_2(2), x_3(2); d(2)),$$
$$(3)\ (x_1(3), x_2(3), x_3(3); d(3)).$$

Ці правила наступні:

$$R_1 : if\ x_1\ is\ A_{11}\ and\ x_2\ is\ A_{21}\ and\ x_3\ is\ A_{31}\ then\ y\ is\ B_1$$
$$R_2 : if\ x_1\ is\ A_{13}\ and\ x_2\ is\ A_{23}\ and\ x_3\ is\ A_{32}\ then\ y\ is\ B_3$$
$$R_3 : if\ x_1\ is\ A_{13}\ and\ x_2\ is\ A_{23}\ and\ x_3\ is\ A_{33}\ then\ y\ is\ B_3.$$

Зазначимо, що при побудові правил враховано, що $x_1(2) = x_1(3)$. Крім того, якщо значення $x_i(j)$ належить кільком областям, то такому значенню ставиться у відповідність нечітка множина, функція належності якої приймає на цьому елементі простору максимальне значення.

3. Вилучаємо суперечності. Потреба в цьому кроці виникає, коли серед побудованих правил є такі, що мають однакові посилання (умови), але різні наслідки. У цьому разі для кожного правила задають міру його істинності. Наприклад, для першого правила міра істинності обчислюється за формулою

$$\mu(R_1) = \mu_{A_{11}}(x_1) \cdot \mu_{A_{21}}(x_2) \cdot \mu_{A_{31}}(x_3).$$

У системі нечітких правил залишаємо правило з найбільшою мірою істинності.

3.2 Вихід системи нечітких правил

Для знаходження виходу системи нечітких правил із заданими входами A_1', A_2', A_3'використовується алгоритм логічного виведення Мамдані [23], який полягає у виконанні наступних кроків:

1. Для кожного правила R_i обчислюємо його рівні істинності. Зокрема,

$$\alpha_1 = \min[\max_{X_1}(A_1'(x_1) \wedge A_{11}(x_1)), \max_{X_2}(A_2'(x_2) \wedge A_{21}(x_2)), \max_{X_3}(A_3'(x_3) \wedge A_{31}(x_3))],$$

$$\alpha_2 = \min[\max_{X_1}(A_1'(x_1) \wedge A_{13}(x_1)), \max_{X_2}(A_2'(x_2) \wedge A_{23}(x_2)), \max_{X_3}(A_3'(x_3) \wedge A_{32}(x_3))],$$

$$\alpha_3 = \min[\max_{X_1}(A_1'(x_1) \wedge A_{13}(x_1)), \max_{X_2}(A_2'(x_2) \wedge A_{23}(x_2)), \max_{X_3}(A_3'(x_3) \wedge A_{33}(x_3))].$$

2. Обчислюємо індивідуальні виходи кожного правила. Зокрема,

$$B_1'(y) = \min(\alpha_1, B_1(y)),$$
$$B_2'(y) = \min(\alpha_2, B_2(y)),$$
$$B_3'(y) = \min(\alpha_3, B_3(y)).$$

3. Обчислюємо агрегатний вихід

$$B'(y) = \max(B_1'(y),\ B_2'(y),\ B_3'(y)).$$

3.3 Елементи теорії ймовірностей нечітких подій

Числові оцінки виходу системи нечітких правил обчислюються за допомогою імовірнісних характеристик нечітких подій. Нечіткою подією A в просторі $\boldsymbol{X}$ будемо називати нечітку множину

$$A = \{x, \mu_A(x), x \in \boldsymbol{X}\},$$

де $\mu : \boldsymbol{X} \to [0,1]$ є функцією належності нечіткої множини A. Формулу для обчислення ймовірності події A можна знайти в [24]. За цією формулою обчислюють ймовірності будь-яких нечітких подій для заданої міри ймовірності.

Умовна ймовірність нечіткої події A за умови виконання нечіткої події B визначається за Демпстером [24-26]. Інакше кажучи, функція розподілу $P_{(A/B)}$

умовної ймовірності нечіткої події A, за умови, що виконується нечітка подія B, визначається за допомогою функції розподілу $P_{(A,B)}$ бінарної ймовірності декартового добутку $A \times B$ і функції розподілу P_B ймовірності нечіткої події B за умови, що вона не дорівнює нулю, Отже,

$$Q_{(A\times B)}(x,y) = \begin{cases} \dfrac{P_{(A,B)}(x,y)}{P_B(y)}, & P_B(y) \neq 0 \\ 1, & P_B(y) = 0. \end{cases}$$

$$P_{(A|B)}(x,y) = \left\{ \frac{Q_{A\times B}(x,y)}{\sum\limits_{x,y} Q_{A\times B}(x,y)} \right.$$

Водночас функція розподілу бінарної ймовірності $P_{(A,B)}$ декартового добутку $A \times B$ обчислюється за формулою

$$P_{(A,B)}(x,y) = \min(P_A(x), P_B(y)),$$

яку можна знайти в [24].

Формула обчислення умовної ймовірності узагальнюється на випадок функцій з будь-якою скінченною кількістю аргументів. Для цього пропонується функцію $Q_{(A\times B\times C\times D)}(x,y,z,v)$ обчислювати за формулою

$$Q_{(A\times B\times C\times D)}(x,y,z,v) = Q_{(A\times B)}(x,y) \cdot Q_{(A\times C)}(x,z) \cdot Q_{(A\times D)}(x,v).$$

Отже, умовна ймовірність нечіткої події A за умови виконання нечітких подій B, C, D визначається у такий спосіб:

$$Q_{(A\times B\times C\times D)}(x,y,z,v)=\begin{cases}\dfrac{P_{(A,B)}(x,y)}{P_B(y)}\cdot\dfrac{P_{(A,C)}(x,z)}{P_C(z)}\cdot\dfrac{P_{(A,D)}(x,v)}{P_D(v)}, & P_B(y)\cdot P_C(z)\cdot P_D(v)\neq 0\\ 1, & P_B(y)\cdot P_C(z)\cdot P_D(v)=0.\end{cases}$$

$$P_{(A|B,C,D)}(x,y,z,v)=\left\{\frac{Q_{A\times B\times C\times D}(x,y,z,v)}{\sum\limits_{x,y,z,v}Q_{A\times B\times C\times D}(x,y,z,v)}.\right.$$

Далі, використовуючи аналог формули повної ймовірності

$$P(B')=\sum_{i=1}^{n}P(B_i/A_{i1},A_{i2},...,A_{im})P(B'/A'_1,A'_2,...A'_m),$$

можна обчислити ймовірність виходу B' системи нечітких правил при входах $A'_1, A'_2, \dots, A'_m$.

3.4 Алгоритм знаходження достовірності виходу

Нехай маємо систему нечіткого логічного виведення, яка містить правила R_1-R_3. Для спрощення розрахунків припустимо, що простори $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3$ однакові, а входи A'_1, A'_2, A'_3 задаються нечіткими множинами

$$A'_1=A_{12},\ A'_2=A_{22},A'_3=A_{12}.$$

Припустимо також, що $A_{11}=A_{21}=A_{31}=B_1$, $A_{12}=A_{22}$, $A_{13}=A_{23}=A_{33}=B_3$

Знаходимо вихід системи нечітких правил за алгоритмом, приведеним вище. Отже, для рівнів істинності правил маємо:

$$\max_{X_1}(A'_1(x_1)\wedge A_{11}(x_1))=\max\mu_{A'_1\cap A_{11}}(x_1)=0.5,$$

$$\max_{X_2}(A_2'(x_2) \wedge A_{21}(x_2)) = \max \mu_{A_2' \cap A_{21}}(x_2) = 0.5,$$

$$\max_{X_3}(A_3'(x_3) \wedge A_{31}(x_3)) = \max \mu_{A_3' \cap A_{31}}(x_3) = 0.5.$$

тому $\alpha_1 = 0.5$.

$$\max_{X_1}(A_1'(x_1) \wedge A_{13}(x_1)) = \max \mu_{A_1' \cap A_{13}}(x_1) = 0.5,$$

$$\max_{X_2}(A_2'(x_2) \wedge A_{23}(x_2)) = \max \mu_{A_2' \cap A_{23}}(x_2) = 0.5,$$

$$\max_{X_3}(A_3'(x_3) \wedge A_{32}(x_3)) = \max \mu_{A_3' \cap A_{32}}(x_3) = 1,$$

тому $\alpha_2 = 0.5$.

$$\max_{X_1}(A_1'(x_1) \wedge A_{13}(x_1)) = \max \mu_{A_1' \cap A_{13}}(x_1) = 0.5,$$

$$\max_{X_2}(A_2'(x_2) \wedge A_{23}(x_2)) = \max \mu_{A_2' \cap A_{23}}(x_2) = 0.5,$$

$$\max_{X_3}(A_3'(x_3) \wedge A_{33}(x_3)) = \max \mu_{A_3' \cap A_{33}}(x_3) = 0.5.$$

тому $\alpha_3 = 0.5$.

Обчислюємо індивідуальні виходи кожного правила. Для цього вводимо нечіткі множини $A_{\alpha_1}, A_{\alpha_2}, A_{\alpha_3}$ з функціями належності рівними константі 0.5. Тоді індивідуальні виходи правил – це нечіткі множини:

$$B_1' = A_{\alpha_1} \cap B_1,$$

$$B_2' = A_{\alpha_2} \cap B_2,$$

$$B_3' = A_{\alpha_3} \cap B_3,$$

з діаграмами Заде, зображеними на рис. 3.2.

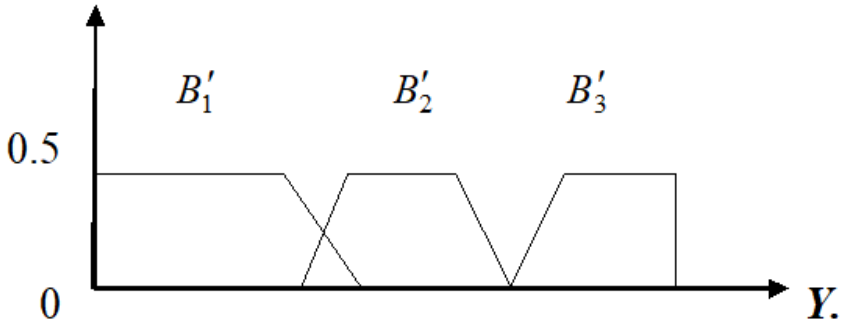

Рис. 3.2. Діаграми Заде для трьох нечітких множин

Агрегатний вихід (вихід системи правил) – це нечітка множина

$$B' = B_1' \cup B_2' \cup B_3'$$

з діаграмою Заде, зображеною на рис. 3.3.

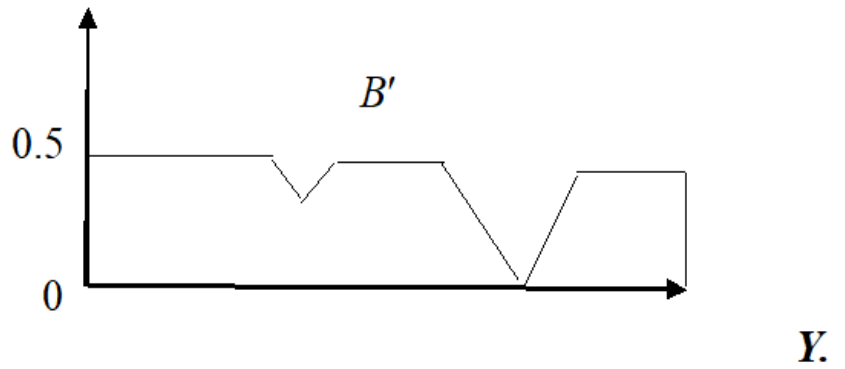

Рис. 3.3. Діаграма Заде для об'єднання трьох нечітких множин

Отже, отримали ще одне правило системи (обчислене правило), яке позначатимемо R_4.

Нам потрібно знайти ймовірність нечіткої події B' при входах A_1', A_2', A_3'. Нехай $P_{X_1}(x), P_{X_2}(x), P_{X_3}(x), P_Y(x)$ – функції розподілу ймовірностей у просторах $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3$, $\boldsymbol{Y}$ відповідно. Обчислимо ймовірності нечітких правил системи та ймовірність виходу системи при входах A_1', A_2', A_3'. Для цього спочатку обчислюємо ймовірності нечітких подій у відповідних правилах. Отже маємо:

$$P(A_{11}) = \int_{X_1} A_{11}(x) \cdot P_{X_1}(x),\ P(A_{21}) = \int_{X_2} A_{21}(x) \cdot P_{X_2}(x),\ P(A_{31}) = \int_{X_3} A_{31}(x) \cdot P_{X_3}(x),$$

$$P(A_{13}) = \int_{X_1} A_{13}(x) \cdot P_{X_1}(x),\ P(A_{23}) = \int_{X_2} A_{23}(x) \cdot P_{X_2}(x),\ P(A_{32}) = \int_{X_3} A_{32}(x) \cdot P_{X_3}(x),$$

$$P(A_{13}) = \int_{X_1} A_{13}(x) \cdot P_{X_1}(x),\ P(A_{23}) = \int_{X_2} A_{23}(x) \cdot P_{X_2}(x),\ P(A_{33}) = \int_{X_3} A_{33}(x) \cdot P_{X_3}(x),$$

$$P(A_1') = \int_{X_1} A_1'(x) \cdot P_{X_1}(x),\ P(A_2') = \int_{X_2} A_2'(x) \cdot P_{X_2}(x),\ P(A_3') = \int_{X_3} A_3'(x) \cdot P_{X_3}(x),$$

$$P(B_1) = \int_{Y} B_1(x) \cdot P_Y(x),\ P(B_2) = \int_{Y} B_2(x) \cdot P_Y(x),\ P(B_3) = \int_{Y} B_3(x) \cdot P_Y(x),$$

тоді ймовірність правил R_1-R_4 обчислюємо як умовні ймовірності. Зокрема, ймовірність правила R_1 обчислюємо як ймовірність нечіткої події B_1 при умові, що відбулися нечіткі події A_{11}, A_{21}, A_{31}. Для цього обчислюємо функції розподілу бінарних ймовірностей декартових добутків $\boldsymbol{T}_1 \times \boldsymbol{T}_4$, $\boldsymbol{T}_2 \times \boldsymbol{T}_4$, $\boldsymbol{T}_3 \times \boldsymbol{T}_4$. Зокрема, для правила R_1:

$$P_{(T_1,T_4)}(A_{11}, B_1) = \min(P(A_{11}), P(B_1)),\ P_{(T_2,T_4)}(A_{21}, B_1) = \min(P(A_{21}), P(B_2)),$$
$$P_{(T_3,T_4)}(A_{31}, B_1) = \min(P(A_{31}), P(B_1)).$$

Для правила R_2:

$$P_{(T_1,T_4)}(A_{13}, B_2) = \min(P(A_{13}), P(B_2)),\ P_{(T_2,T_4)}(A_{23}, B_2) = \min(P(A_{23}), P(B_2)),$$
$$P_{(T_3,T_4)}(A_{32}, B_2) = \min(P(A_{32}), P(B_2)).$$

Для правила R_3:

$$P_{(T_3,T_4)}(A_{13},B_3)=\min(P(A_{13}),P(B_3)),\ P_{(T_2,T_4)}(A_{23},B_3)=\min(P(A_{23}),P(B_3)),$$
$$P_{(T_3,T_4)}(A_{33},B_3)=\min(P(A_{33}),P(B_3)).$$

Для правила R_4:

$$P_{(T_1,T_4)}(A_1',B')=\min(P(A_1'),P(B')),\ P_{(T_2,T_4)}(A_2',B')=\min(P(A_2'),P(B')),$$
$$P_{(T_3,T_4)}(A_3',B')=\min(P(A'),P(B')).$$

Для кожного із правил R_1-R_4 знаходимо значення функцій

$$Q_{R_1}(x,y,z,v),\ Q_{R_2}(x,y,z,v),Q_{R_3}(x,y,z,v),\ Q_{R_4}(x,y,z,v)$$

на елементах простору $\boldsymbol{T}_1\times\boldsymbol{T}_2\times\boldsymbol{T}_3\times\boldsymbol{T}_4$, які відповідають нечітким правилам.

Зокрема,

$$Q_{R_1}(A_{11},A_{21},A_{31},B_1)=\begin{cases}\dfrac{P_{(T_1,T_4)}(A_{11},B_1)}{P(A_{11})}\cdot\dfrac{P_{(T_2,T_4)}(A_{21},B_1)}{P(A_{21})}\cdot\dfrac{P_{(T_3,T_4)}(A_{31},B_1)}{P(A_{31})},\\ 1\end{cases}$$

$$Q_{R_2}(A_{13},A_{23},A_{32},B_2)=\begin{cases}\dfrac{P_{(T_1,T_4)}(A_{13},B_2)}{P(A_{13})}\cdot\dfrac{P_{(T_2,T_4)}(A_{23},B_2)}{P(A_{23})}\cdot\dfrac{P_{(T_3,T_4)}(A_{32},B_2)}{P(A_{32})},\\ 1,\end{cases}$$

$$Q_{R_3}(A_{13},A_{23},A_{33},B_3)=\begin{cases}\dfrac{P_{(T_1,T_4)}(A_{13},B_3)}{P(A_{13})}\cdot\dfrac{P_{(T_2,T_4)}(A_{23},B_3)}{P(A_{23})}\cdot\dfrac{P_{(T_3,T_4)}(A_{33},B_3)}{P(A_{33})},\\ 1,\end{cases}$$

$$Q_{R_4}(A_1', A_2', A_3', B') = \begin{cases} \dfrac{P_{(T_1,T_4)}(A_1', B')}{P(A_1')} \cdot \dfrac{P_{(T_2,T_4)}(A', B')}{P(A')} \cdot \dfrac{P_{(T_3,T_4)}(A_3', B')}{P(A_3')}, \\ 1, \end{cases}$$

зауважимо, що значення функції Q_{R_i} дорівнює одиниці у випадку рівності нулю хоча б одного із знаменників.

На наступному кроці знаходимо умовні ймовірності нечітких правил.

$$P_{R_1}(A_{11}, A_{21}, A_{31} | B_1) = \begin{cases} \dfrac{Q_{R_1}(A_{11}, A_{21}, A_{31}, B_1)}{\sum\limits_i Q_{R_i}(x, y, z, v)}, \end{cases}$$

$$P_{R_2}(A_{13}, A_{23}, A_{32} | B_2) = \begin{cases} \dfrac{Q_{R_2}(A_{13}, A_{23}, A_{32}, B_2)}{\sum\limits_i Q_{R_i}(x, y, z, v)}, \end{cases}$$

$$P_{R_3}(A_{13}, A_{23}, A_{33} | B_3) = \begin{cases} \dfrac{Q_{R_3}(A_{13}, A_{23}, A_{33}, B_3)}{\sum\limits_i Q_{R_i}(x, y, z, v)}, \end{cases}$$

$$P_{R_4}(A_1', A_2', A_3' | B') = \begin{cases} \dfrac{Q_{R_4}(A_1', A_2', A', B')}{\sum\limits_i Q_{R_i}(x, y, z, v)}. \end{cases}$$

За узагальненою формулою повної ймовірності знаходимо ймовірність виходу системи нечітких правил при входах A_{11}, A_{21}, A_{31}. Отже, маємо:

$$\begin{aligned} P(B') = {} & P_{R_1}(A_{11}, A_{21}, A_{31} | B_1) \cdot P_{R_4}(A_1', A_2', A_3' | B') + \\ & P_{R_2}(A_{13}, A_{23}, A_{32} | B_2) \cdot P_{R_4}(A_1', A_2', A_3' | B') + \\ & P_{R_3}(A_{13}, A_{23}, A_{33} | B_3) \cdot P_{R_4}(A_1', A_2', A_3' | B'). \end{aligned}$$

3.5 Приклад (скінченні простори)

Нехай $\boldsymbol{X}_1 = \boldsymbol{X}_2 = \boldsymbol{X}_3 = \{K, V, S, E, G, I\}$, $Y = \{a, b, c\}$. Як і в неперервному випадку, розбиваємо кожен з просторів на три частини і задаємо відповідні функції належності. Зокрема, нечіткі множини A_{ij} та B_k задамо у вигляді:

$$A_{11} = 1/K + 0.5/V,\ A_{12} = 0.5/S + 0.5/E,\ A_{13} = 1/G + 0.5/I,$$
$$A_{21} = 1/K + 0.5/V,\ A_{22} = 0.5/S + 0.5/E,\ A_{23} = 1/G + 0.5/I,$$
$$A_{31} = 1/G + 0.5/I,\ A_{32} = 0.8/S + 0.8/E,\ A_{33} = 1/G + 0.5/I,$$
$$B_1 = 1/a,\ B_2 = 0.5/b,\ B_3 = 1/c.$$

Нехай на вхід системи нечітких правил

$$R_1 : if\ x_1\ is\ A_{11}\ and\ x_2\ is\ A_{21}\ and\ x_3\ is\ A_{31}\ then\ y\ is\ B_1$$
$$R_2 : if\ x_1\ is\ A_{13}\ and\ x_2\ is\ A_{23}\ and\ x_3\ is\ A_{32}\ then\ y\ is\ B_3$$
$$R_3 : if\ x_1\ is\ A_{13}\ and\ x_2\ is\ A_{23}\ and\ x_3\ is\ A_{33}\ then\ y\ is\ B_3.$$

подаються значення:

$$A_1' = 0.8 / K + \ 0.5 / V,\ A_2' = 0.5 / S + \ 0.5 / E, A_3' = 0.6 / G + \ 0.5 / I.$$

Знайдемо вихід B' системи нечітких правил за алгоритмом Мамдані. Отже, для рівнів істинності правил маємо:

$$\max_{X_1}(A_1'(x_1) \wedge A_{11}(x_1)) = \max(0.8, 0.5) = 0.8,$$
$$\max_{X_2}(A_2'(x_2) \wedge A_{12}(x_2)) = \max(0.5, 0.5) = 0.5,$$
$$\max_{X_3}(A_3'(x_3) \wedge A_{13}(x_3)) = \max(0.6, 0.5) = 0.6.$$

тому $\alpha_1 = 0.5$.

$$\max_{X_1}(A_1'(x_1) \wedge A_{21}(x_1)) = \max(0.8, 0.5) = 0.8,$$

$$\max_{X_2}(A_2'(x_2) \wedge A_{22}(x_2)) = \max(0.5, 0.5) = 0.5,$$

$$\max_{X_3}(A_3'(x_3) \wedge A_{23}(x_3)) = \max(0.6, 0.5) = 0.6,$$

тому $\alpha_2 = 0.5$.

$$\max_{X_1}(A_1'(x_1) \wedge A_{31}(x_1)) = \max(0.8, 0.5) = 0.8,$$

$$\max_{X_2}(A_2'(x_2) \wedge A_{32}(x_2)) = \max(0.5, 0.5) = 0.5,$$

$$\max_{X_3}(A_3'(x_3) \wedge A_{33}(x_3)) = \max \mu_{A_3' \cap A_{33}}(x_3) = 0.5.$$

тому $\alpha_3 = 0.5$.

Обчислюємо індивідуальні виходи кожного правила.

$$B_1'(a) = \min(0.5, B_1(1)) = 0.5,$$

$$B_2'(b) = \min(0.5, B_2(b)) = 0.5,$$

$$B_3'(c) = \min(0.5, B_3(c)) = 0.5.$$

Агрегатний вихід (вихід системи правил) – це нечітка множина

$$B' = 0.5 / a + 0.5 / b + 0.5 / c.$$

Отже, маємо ще одне нечітке правило R_4:

$$R_4 : if\ x_1\ is\ A_1'\ and\ x_2\ is\ A_2'\ and\ x_3\ is\ A_{33}'\ then\ y\ is\ B'.$$

Нехай $P_1(x)$ – функція розподілу ймовірностей у просторах $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3$ відповідно, а $P_2(x)$ – функція розподілу ймовірностей у просторі $\boldsymbol{Y}$. До того ж, $P_1(K) = 0.1$, $P_1(V) = 0.1$, $P_1(S) = 0.2$, $P_1(E) = 0.2$, $P_1(G) = 0.2$, $P_1(I) = 0.2$, $P_2(a) = 0.4$, $P_2(b) = 0.4$, $P_2(\text{c}) = 0.2$.

Обчислимо ймовірності нечітких подій у відповідних правилах.

$$P(A_{11}) = \sum_x A_{11}(x) \cdot P_1(x) = 0.15,\ P(A_{12}) = \sum_x A_{12}(x) \cdot P_1(x) = 0.2,$$

$$P(A_{13}) = \sum_x A_{13}(x) \cdot P_1(x) = 0.3,\ P(A_{21}) = \sum_x A_{21}(x) \cdot P_1(x) = 0.15,$$

$$P(A_{22}) = \sum_x A_{22}(x) \cdot P_1(x) = 0.2,\ P(A_{23}) = \sum_x A_{23}(x) \cdot P_1(x) = 0.3,$$

$$P(A_{31}) = \sum_x A_{31}(x) \cdot P_1(x) = 0.15,\ P(A_{32}) = \sum_x A_{32}(x) \cdot P_1(x) = 0.32,$$

$$P(A_{33}) = \sum_x A_{33}(x) \cdot P_1(x) = 0.3,\ P(A_1') = \sum_x A_1'(x) \cdot P_1(x) = 0.13,$$

$$P(A_2') = \sum_x A_2'(x) \cdot P_1(x) = 0.2,\ P(A_3') = \sum_x A_3'(x) \cdot P_1(x) = 0.22,$$

$$P(B_1) = \sum_x B_1(x) \cdot P_2(x) = 0.4,\ P(B_2) = \sum_x B_2(x) \cdot P_2(x) = 0.2,$$

$$P(B_3) = \sum_x B_3(x) \cdot P_2(x) = 0.2,\ P(B') = \sum_x B'(x) \cdot P_2(x) = 0.5.$$

Утворюємо простори терм-множин $\boldsymbol{T}_1 = \{A_{11}, A_{12}, A_{13}\}$, $\boldsymbol{T}_2 = \{A_{21}, A_{22}, A_{23}\}$, $\boldsymbol{T}_3 = \{A_{31}, A_{32}, A_{33}\}$, $\boldsymbol{T}_4 = \{B_1, B_2, B_3\}$ і обчислюємо функції розподілу бінарних ймовірностей декартових добутків $\boldsymbol{T}_1 \times \boldsymbol{T}_4$, $\boldsymbol{T}_2 \times \boldsymbol{T}_4$, $\boldsymbol{T}_3 \times \boldsymbol{T}_4$.

Зокрема, для правила R_1:

$$P_{(T_1,T_4)}(A_{11}, B_1) = \min(P(A_{11}), P(B_1)) = 0.15,$$

$$P_{(T_2,T_4)}(A_{12}, B_1) = \min(P(A_{12}), P(B_1)) = 0.2,$$

$$P_{(T_3,T_4)}(A_{13}, B_1) = \min(P(A_{13}), P(B_1)) = 0.3.$$

Для правила R_2:

$$P_{(T_1,T_4)}(A_{21},B_2)=\min(P(A_{21}),P(B_2))=0.15,\ P_{(T_2,T_4)}(A_{22},B_2)=\min(P(A_{22}),P(B_2))=0.2,$$
$$P_{(T_3,T_4)}(A_{23},B_2)=\min(P(A_{23}),P(B_2))=0.2.$$

Для правила R_3:

$$P_{(T_3,T_4)}(A_{31},B_3)=\min(P(A_{31}),P(B_3))=0.15,\ P_{(T_2,T_4)}(A_{32},B_3)=\min(P(A_{32}),P(B_3))=0.2,$$
$$P_{(T_3,T_4)}(A_{33},B_3)=\min(P(A_{33}),P(B_3))=0.2.$$

Для правила R_4:

$$P_{(T_1,T_4)}(A_1',B')=\min(P(A_1'),P(B'))=0.13,\ P_{(T_2,T_4)}(A_2',B')=\min(P(A_2'),P(B'))=0.2,$$
$$P_{(T_3,T_4)}(A_3',B')=\min(P(A_3'),P(B'))=0.22.$$

Для кожного із правил R_1-R_4 знаходимо значення функцій

$$Q_{R_1}(x,y,z,v),\ Q_{R_2}(x,y,z,v),Q_{R_3}(x,y,z,v),\ Q_{R_4}(x,y,z,v)$$

на елементах простору $\boldsymbol{T}_1\times\boldsymbol{T}_2\times\boldsymbol{T}_3\times\boldsymbol{T}_4$, які відповідають нечітким правилам.

Зокрема,

$$Q_{R_1}(A_{11},A_{12},A_{13},B_1)=\begin{cases}\dfrac{P_{(T_1,T_4)}(A_{11},B_1)}{P(A_{11})}\cdot\dfrac{P_{(T_2,T_4)}(A_{12},B_1)}{P(A_{12})}\cdot\dfrac{P_{(T_3,T_4)}(A_{13},B_1)}{P(A_{13})}=1,\\ 1\end{cases}$$

$$Q_{R_2}(A_{21},A_{22},A_{23},B_2)=\begin{cases}\dfrac{P_{(T_1,T_4)}(A_{21},B_2)}{P(A_{21})}\cdot\dfrac{P_{(T_2,T_4)}(A_{22},B_2)}{P(A_{22})}\cdot\dfrac{P_{(T_3,T_4)}(A_{23},B_2)}{P(A_{23})}=\dfrac{2}{3},\\ 1,\end{cases}$$

$$Q_{R_3}(A_{31},A_{32},A_{33},B_3)=\begin{cases}\dfrac{P_{(T_1,T_4)}(A_{31},B_3)}{P(A_{31})}\cdot\dfrac{P_{(T_2,T_4)}(A_{32},B_3)}{P(A_{32})}\cdot\dfrac{P_{(T_3,T_4)}(A_{33},B_3)}{P(A_{33})}=\dfrac{1}{16},\\ 1,\end{cases}$$

$$Q_{R_4}(A_1',A_2',A_3',B')=\begin{cases}\dfrac{P_{(T_1,T_4)}(A_1',B')}{P(A_1')}\cdot\dfrac{P_{(T_2,T_4)}(A_2',B')}{P(A_2')}\cdot\dfrac{P_{(T_3,T_4)}(A_3',B')}{P(A_3')}=1,\\ 1,\end{cases}$$

на наступному кроці знаходимо умовні ймовірності нечітких правил.

$$P_{R_1}(A_{11},A_{12},A_{13}\,|\,B_1)=\begin{cases}\dfrac{Q_{R_1}(A_{11},A_{12},A_{13},B_1)}{\sum\limits_i Q_{R_i}(x,y,z,v)}=\dfrac{48}{131},\end{cases}$$

$$P_{R_2}(A_{21},A_{22},A_{23}\,|\,B_2)=\begin{cases}\dfrac{Q_{R_2}(A_{21},A_{22},A_{23},B_2)}{\sum\limits_i Q_{R_i}(x,y,z,v)}=\dfrac{96}{393},\end{cases}$$

$$P_{R_3}(A_{31},A_{32},A_{33}\,|\,B_3)=\begin{cases}\dfrac{Q_{R_3}(A_{31},A_{32},A_{33},B_3)}{\sum\limits_i Q_{R_i}(x,y,z,v)}=\dfrac{131}{768},\end{cases}$$

$$P_{R_4}(A_1',A_2',A_3'\,|\,B')=\begin{cases}\dfrac{Q_{R_4}(A_1',A_2',A',B')}{\sum\limits_i Q_{R_i}(x,y,z,v)}=\dfrac{48}{131}.\end{cases}$$

За узагальненою формулою повної ймовірності знаходимо ймовірність виходу B' системи нечітких правил при входах A_{11},A_{21},A_{31}. Отже, маємо:

$$P(B') = P_{R_1}(A_{11}, A_{21}, A_{31} | B_1) \cdot P_{R_4}(A'_1, A'_2, A'_3 | B') + P_{R_2}(A_{13}, A_{23}, A_{32} | B_2) \cdot P_{R_4}(A'_1, A'_2, A'_3 | B') + P_{R_3}(A_{13}, A_{23}, A_{33} | B_3) \cdot P_{R_4}(A'_1, A'_2, A'_3 | B') \approx 0.06.$$

3.6 Висновки

Запропонований підхід на основі нечітких моделей дозволяє обчислювати достовірність виходу системи нечітких правил. Зрозуміло, що ці результати потребують подальших досліджень. Зокрема, необхідно відповісти на питання про межі оцінок достовірності, повноту системи правил, залежність достовірності результатів від кількості правил. Відповіді на деякі з цих питань досить прості, але відповіді на інші потребують досить складних обчислень.

Цей підхід до моделювання нечітких знань використовувався авторами при побудові системи прийняття рішень в нечітких умовах. Є сподівання, що одержані в статті результати (можливо з певними уточненнями в запропонованих формулах) можуть бути використані в подальшому для навчання таких систем з метою отримання найбільш достовірних результатів.

РОЗДІЛ 4: ДЕТАЛІЗАЦІЯ ПРОГРАМНИХ РІШЕНЬ РОЗРОБКИ СИСТЕМИ

4.1 Критерії архітектури програмного забезпечення

4.1.1 Основні класи сучасного програмного забезпечення

До основних класів сучасного програмного забезпечення можна віднести десктопні та веборієнтовані програми. Розглянемо кожен з цих класів програм окремо.

Десктопні програми – це програми, які розробляються для встановлення та використання на локальному комп'ютері або ноутбуці, що працюють під операційною системою, такою як Windows, MacOS чи Linux. Класичний приклад десктопної програми – текстовий редактор, який дозволяє створювати та редагувати документи, що зберігаються на локальному диску. Інші приклади десктопних програм включають графічні редактори, програми для обробки відео та звуку, браузери, клієнти електронної пошти, різноманітні ігри, програми для налаштування системи тощо.

Основна перевага десктопних програм полягає в тому, що вони працюють безпосередньо на комп'ютері, що дає можливість отримати більшу продуктивність та функціональність, ніж в вебпрограмах. Крім того, десктопні програми можуть працювати в автономному режимі, не потребуючи постійного з'єднання з Інтернетом, що робить їх більш зручними для користувачів, які працюють у зоні з обмеженим доступом до Інтернету або у віддалених місцях.

Однак десктопні програми мають свої недоліки, зокрема вони потребують встановлення та налаштування на кожному комп'ютері, на якому вони повинні працювати.

Вебзастосунки або вебсистеми – це клас програмного забезпечення, які використовують вебтехнології для забезпечення доступу до різних функцій через браузер. Вони можуть бути доступні як локально на сервері, так і в мережі Інтернет.

Основні компоненти вебсистем включають вебсервер, базу даних та фронтенд. Вебсервер відповідає за обробку запитів від клієнтів та відправку відповіді. База даних використовується для зберігання даних, які використовуються в системі.

Фронтенд – це один із можливих інтерфейсів користувача, який відображається в браузері та дозволяє користувачам взаємодіяти з системою. Крім інтерфейсу в браузері, універсальні протоколи передачі даних дозволяють реалізовувати клієнтську частину на широкому колі пристроїв – від звичайних комп'ютерів та смартфонів до вбудованих систем або суперкомп'ютерів.

Однією з головних переваг створення вебсистем є їх доступність з будь-якого місця, де є Інтернет-з'єднання. Користувач може отримати доступ до вебсистеми зі свого персонального пристрою. В цьому випадку не потрібно встановлювати окремий додаток на кожен пристрій, що значно спрощує процес розгортання та підтримки системи.

Іншою важливою перевагою є можливість оновлення системи з використанням централізованих серверів. Користувачі не потрібно завантажувати нову версію системи на свій комп'ютер, оскільки оновлення відбуваються на сервері. Це зменшує ризик помилок та вразливостей, що можуть виникнути при локальному оновленні десктопної програми.

Також вебсистеми зазвичай мають більш простий інтерфейс, який дозволяє користувачам легко розуміти та використовувати систему без необхідності вивчення складних інструкцій. Крім того, вебсистеми можуть бути більш гнучкими та модульними, що дозволяє змінювати функціональність та додавати нові можливості в вебсистему без необхідності встановлення нових версій на локальні комп'ютери.

Нарешті, створення вебсистем може бути більш ефективним з точки зору вартості, оскільки вони можуть бути створені з використанням відкритих технологій та безкоштовних програмних засобів. Це може знизити вартість розробки та підтримки системи, зокрема у порівнянні з десктопними додатками, які часто потребують дорогих ліцензій на програмне забезпечення та обладнання.

Отже, створення вебсистем має багато переваг порівняно з десктопними додатками. Вебсистеми є більш доступними, оскільки вони можуть бути запущені на будь-якому пристрої з доступом до Інтернету. Більш того, вебсистеми можуть бути оновлені централізовано на сервері, тим самим зменшуючи зусилля з оновлення на кожному пристрої окремо.

Загалом, вебсистеми є більш зручним і ефективним способом для створення програмного забезпечення, оскільки вони дозволяють знизити витрати на розробку, зберігання та підтримку програмного забезпечення. Відтак, вебсистеми стають все більш популярними у сучасному світі, де швидкість і доступність є ключовими факторами успіху.

Таким чином, найкращим класом для реалізації сучасної програмної системи діагностики пацієнтів є вебсистема.

4.1.2 Вимоги до сучасних вебсистем

Сучасні вебсистеми мають високі апаратні та програмні вимоги, щоб відповідати вимогам користувачів. Для успішної роботи вебсистеми необхідне забезпечення достатньої швидкодії, безпеки, надійності та масштабованості.

Перш за все, необхідність масштабованості приводить до вимог до серверного обладнання, яке повинно забезпечувати високу продуктивність та надійність. Вебсистема повинна бути здатна працювати з великою кількістю користувачів та обробляти великі обсяги даних. Тому серверне обладнання повинно мати достатньо ресурсів для забезпечення продуктивності та можливості масштабування.

Друге, вимога до безпеки системи є дуже важливою, оскільки вебсистеми зберігають велику кількість конфіденційної інформації, такої як особисті дані користувачів, банківські реквізити, інформацію про платежі тощо. Для забезпечення безпеки вебсистема повинна мати вбудовану систему безпеки, яка забезпечує шифрування та автентифікацію користувачів, перевірку прав доступу, захист від атак та вірусів.

Третє, вимога до швидкодії вебсистеми є важливою, оскільки користувачі очікують, що система буде працювати швидко та ефективно. Швидкодія вебсистеми залежить від багатьох чинників, таких як швидкість серверного обладнання, кількість запитів до бази даних, оптимізація коду програмного забезпечення тощо. Щоб забезпечити високу швидкість роботи вебсистем існує безліч високорівневих архітектур для вебзастосунків. До найбільш відомого типу можна віднести трирівневу архітектуру [27].

В рамках роботи розглянемо розширену деталізовану архітектуру програмного забезпечення, яка враховує вимоги масштабованості, швидкодії та безпеки [28]. Вона може бути розглянута в контексті наступних основних рівнів:

- Рівень інфраструктури
- Рівень зберігання даних
- Презентаційний рівень
- Рівень бізнес-логіки
- Рівень моніторингу

Схематично всі рівні можна представити схемою, як зображено на рис. 4.1.

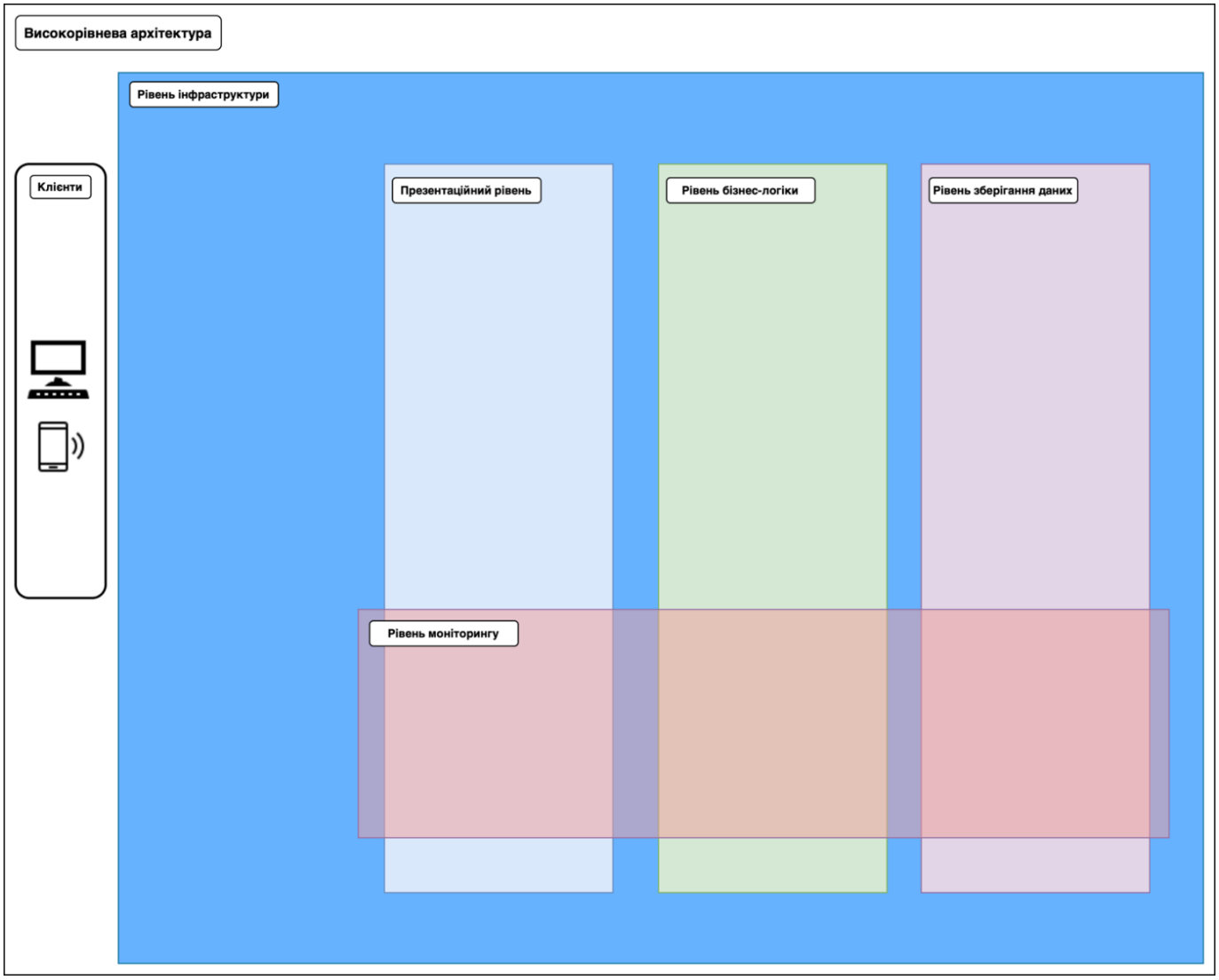

Рис. 4.1. Високорівнева архітектура вебзастосунка для системи підтримки прийняття рішень

Оскільки вибір реалізації кожного з рівня є нетривіальною задачею, то опишемо кожен з рівнів більш детально.

4.1.3 Інфраструктурний рівень

Рівень інфраструктури – це основа, на якій працює вся вебсистема. Він включає всі аспекти фізичної та віртуальної інфраструктури, необхідні для підтримки роботи вебсистеми.

Фізична інфраструктура. Ця інфраструктура включає в себе всі сервери, мережеві пристрої (маршрутизатори, комутатори, брандмауери) та інші апаратні компоненти, необхідні для роботи вебсистеми. Фізична інфраструктура може бути розміщена у власному дата-центрі (внутрішня інфраструктура) або може бути орендована від постачальників хмарних сервісів (зовнішня інфраструктура). У цьому сенсі також потрібно чітко відрізняти **внутрішню інфраструктуру**, яка знаходиться у зоні відповідальності команди розробників, і **зовнішню інфраструктуру**.

Віртуальна інфраструктура. Ця інфраструктура включає в себе всі віртуальні машини, контейнери та інші віртуальні ресурси, що використовуються для розгортання та роботи вебсистеми. Віртуальна інфраструктура може бути розміщена на фізичній інфраструктурі, або може бути надана як служба (Infrastructure as a Service, IaaS) постачальниками хмарних сервісів.

Мережева інфраструктура. Ця інфраструктура включає в себе всі аспекти мережі, необхідні для підключення вебсистеми до Інтернету та забезпечення її доступності для користувачів. Мережева інфраструктура включає такі компоненти, як DNS-сервери, балансувальники навантаження, брандмауери та VPN-тунелі.

Управління інфраструктурою. Управління включає в себе всі процеси та інструменти, необхідні для розгортання, моніторингу, обслуговування та масштабування інфраструктури. Це може включати інструменти автоматизації, такі як Terraform або Ansible, для автоматичного розгортання та управління інфраструктурою.

Безпека інфраструктури. Безпека включає в себе всі аспекти захисту інфраструктури від загроз та забезпечення її роботи в разі відмов. Це може включати використання брандмауерів та системи виявлення вторгнень для захисту від

зовнішніх загроз, застосування політик безпеки для забезпечення контролю доступу до інфраструктури, та розробку планів відновлення після аварій для забезпечення роботи системи в разі відмов.

Отже, рівень інфраструктури має критичне значення для стабільності, масштабованості та безпеки вебсистеми. Він вимагає ретельного планування, проектування та управління, щоб забезпечити ефективну роботу всіх інших рівнів архітектури вебсистеми.

Для відповідності всім аспектам рівня інфраструктури було обрано наступні технології та сервіси для реалізації системи діагностики пацієнтів:

- Hetzner Cloud
- Kubernetes
- CloudFlare

Hetzner Cloud – це постачальник хмарних ресурсів, який надає доступ до потужних серверів за доступними цінами [29]. Основні переваги використання Hetzner Cloud включають:

Вартість. Hetzner Cloud відомий своїми конкурентоспроможними цінами. Вони зазвичай надають більше ресурсів за меншу вартість порівняно з іншими провайдерами хмарних послуг.

Простота використання. Hetzner Cloud має простий та інтуїтивно зрозумілий інтерфейс, який полегшує налаштування та управління серверами.

Якість обслуговування. Hetzner Cloud відомий своєю високою якістю обслуговування та відмінною підтримкою. Сервіс має гарну репутацію в індустрії за свою здатність швидко реагувати на проблеми та надавати допомогу.

Місцезнаходження центрів обробки даних. Hetzner Cloud має центри обробки даних в Німеччині, Фінляндії та США, що може бути перевагою для користувачів з цих або сусідніх країн.

Опції масштабування. Hetzner Cloud пропонує гнучкі опції масштабування, які дозволяють легко збільшувати або зменшувати ресурси в залежності від потреб користувача.

Безпека та надійність. Hetzner Cloud пропонує ряд рішень для захисту даних та безперебійної роботи, включаючи резервне копіювання та захист від DDoS-атак.

API та автоматизація. Hetzner Cloud надає високоякісний API, який дозволяє автоматизувати ряд процесів, таких як створення, змінення, масштабування та видалення серверів.

Мережеві можливості. Hetzner Cloud надає високу пропускну спроможність мережі, що може покращити швидкість передачі даних.

Сервери з високою продуктивністю. Hetzner Cloud надає потужні сервери з високоякісним обладнанням. Зокрема, вони пропонують сервери з SSD-дисками для підвищення швидкості I/O.

Треба зазначити, що, незважаючи на всі ці переваги, Hetzner Cloud має і свої недоліки. Наприклад, вони не мають центрів обробки даних поза Європою та США, що може обмежувати їх привабливість для користувачів в інших частинах світу. Також вони не надають вбудованих сервісів у сфері штучного інтелекту, які надають деякі інші провайдери хмарних послуг. Однак, враховуючи наші вимоги, можна сказати, що Hetzner Cloud відповідає потребам системи, що розробляється.

Kubernetes – це відкрита платформа-оркестратор контейнерів, яка автоматизує розгортання, масштабування та управління контейнерізованими застосунками. Він був розроблений компанією Google і переданий в опіку Cloud Native Computing Foundation. Kubernetes використовується для управління сервісами та розгортання застосунків віртуальних машин та контейнерів [30-31]. Основні переваги Kubernetes включають:

Самовідновлення. Kubernetes забезпечує стійкість контейнерів, перезапускаючи збойні контейнери, замінюючи їх або перерозподіляючи їх на випадок виходу з ладу вузла.

Масштабування. Kubernetes дозволяє горизонтальне масштабування контейнерів, додаючи або видаляючи контейнери залежно від потреб. Масштабування може виконуватися автоматично або за ручним запитом.

Балансування навантаження. Kubernetes може автоматично розподіляти навантаження між контейнерами, забезпечуючи рівномірне використання ресурсів та підтримуючи високу доступність.

Декларативна конфігурація. Kubernetes дозволяє вказати бажаний стан розгорнутого застосунку, а система автоматично приводить інфраструктуру до вказаного стану.

Безшовне оновлення. Kubernetes спрощує процес оновлення застосунків, дозволяючи поетапне розгортання нових версій без зупинки роботи старих версій.

Сегрегація ресурсів. Kubernetes дозволяє ізолювати ресурси від інших застосунків та контейнерів, забезпечуючи безпеку та оптимальне використання ресурсів. Він дозволяє задавати обмеження на використання ресурсів на рівні контейнера, що допомагає уникнути ситуацій, коли один контейнер може використовувати більше ресурсів, ніж йому потрібно, тим самим знижуючи роботу інших контейнерів.

Управління секретами та конфіденційними даними. Kubernetes дозволяє управляти секретними та конфіденційними даними, такими як паролі, OAuth токени та SSH ключі, без відкриття їх в конфігурації застосунку.

Отже, Kubernetes – це міцна платформа, що спрощує процеси розгортання, масштабування, оновлення та відновлення застосунків, забезпечуючи при цьому високий рівень безпеки та ефективності ресурсів, саме тому її було обрано для реалізації системи.

Cloudflare є одним з провідних провайдерів послуг **мережі розповсюдження контенту** CDN (Content Delivery Network) [33-34], що надає цілий ряд переваг для своїх користувачів:

Безкоштовний план. Cloudflare пропонує безкоштовний план, який включає багато основних функцій CDN, що робить його доступним для малих сайтів та початкових проектів.

Високий рівень продуктивності та швидкості. Cloudflare забезпечує швидку доставку контенту завдяки своєму глобальному мережевому присутності. Це може

значно покращити час завантаження сторінки та загалом покращити користувацький досвід.

Безпека. Cloudflare пропонує ряд рішень для захисту від різноманітних загроз, включаючи DDoS-атаки, SQL-ін’єкції та XSS.

Широкий спектр послуг. Крім послуг CDN, Cloudflare також пропонує ряд інших послуг, включаючи DNS, WAF (Web Application Firewall), оптимізацію SSL/TLS і багато іншого.

Простота використання та налаштування. Cloudflare відомий своїм інтуїтивним, користувацьки-орієнтованим інтерфейсом, який полегшує налаштування та управління послугами.

Масштабованість. Cloudflare здатний масштабуватися, щоб відповідати зростаючим потребам вашого вебсайту або додатку без додаткових зусиль з вашого боку.

Зменшення навантаження на сервер. Cloudflare виконує кешування статичного контенту на своїх крайніх серверах, зменшуючи тим самим кількість запитів до вашого оригінального сервера. Це може підвищити продуктивність та відповідність сервера.

Основна ідея, що стоїть за Cloudflare, полягає в тому, щоб зробити інтернет більш безпечним, приватним і швидким місцем для всіх. Незважаючи на те, що є інші варіанти CDN, Cloudflare продовжує бути одним з найбільш відомих та широко використовуваних рішень на ринку. Враховуючи вищезазначене, саме Cloudflare було обрано для реалізації системи.

4.1.4 Рівень зберігання даних

Рівень зберігання даних – це суттєво важливий компонент архітектури будь-якої вебсистеми. Цей рівень відповідає за зберігання, відновлення та управління даними, які використовуються системою. Давайте більш детально розглянемо ключові аспекти цього рівня.

Типи баз даних. В залежності від вимог до системи можуть використовуватися різні типи баз даних. Це можуть бути реляційні бази даних (наприклад, PostgreSQL, MySQL), NoSQL бази даних (наприклад, MongoDB, Cassandra), або навіть бази даних

з підтримкою часових рядів (InfluxDB), якщо система обробляє дані сильно залежні від часу.

Шардування та реплікація. Для підтримки великих об’ємів даних та високої доступності, рівень зберігання даних може включати шардування (розподіл даних між кількома базами даних) та реплікацію (створення копій даних для підвищення доступності та відмовостійкості).

Безпека. Рівень зберігання даних відіграє важливу роль у забезпеченні безпеки системи. Це включає захист від несанкціонованого доступу, використання шифрування для захисту даних та регулярне створення резервних копій для захисту від втрати даних.

Процедури резервного копіювання та відновлення. Для забезпечення надійності та відмовостійкості системи необхідно регулярно створювати резервні копії даних та мати процедури для їх швидкого відновлення у разі втрати або пошкодження даних.

Масштабованість. Рівень зберігання даних повинен бути здатний масштабуватися, щоб впоратися зі збільшенням обсягу даних і кількості запитів. Це може включати горизонтальне масштабування (додавання нових вузлів до бази даних), вертикальне масштабування (збільшення потужності окремого вузла) та оптимізацію запитів до бази даних.

Оптимізація запитів. Для забезпечення високої продуктивності та ефективного використання ресурсів, запити до бази даних повинні бути оптимізовані. Це може включати використання індексів для прискорення пошуку, оптимізацію структури даних та використання кешування для зменшення навантаження на базу даних.

Транзакційна обробка. В залежності від вимог до системи, рівень зберігання даних може підтримувати транзакційну обробку, що забезпечує консистентність даних, навіть у разі помилок або збоїв.

Отже, рівень зберігання даних відповідає за безпечне, надійне та ефективне зберігання та управління даними, що використовуються вебсистемою, включаючи

забезпечення масштабованості та високої продуктивності, транзакційної обробки та захисту від несанкціонованого доступу та втрати даних.

Для відповідності всім аспектам рівня зберігання даних було обрано наступні технології та підходи для реалізації системи діагностики пацієнтів:

- PostgreSQL
- Реплікація бази даних для зчитування
- Об’єктно-реляційне відображення

PostgreSQL є однією з найпотужніших систем керування базами даних (СКБД) з відкритим вихідним кодом, яка має декілька унікальних переваг у порівнянні з іншими SQL та NoSQL базами даних [35-37].

Повнота SQL. PostgreSQL повністю підтримує стандарт SQL, що включає такі можливості, як підзапити, транзакції, виклик зовнішніх функцій, а також підтримку майже всіх типів даних.

Розширюваність. PostgreSQL є надзвичайно гнучкою СКБД, яка дозволяє користувачам створювати та використовувати свої власні типи даних, оператори та функції. Крім того, вона підтримує розширення, що дозволяють додавати нові функції або типи даних без зміни ядра бази даних.

Багатоверсійність транзакцій. PostgreSQL використовує модель MVCC, яка дозволяє кільком користувачам одночасно читати і записувати дані без блокування. Це забезпечує високу продуктивність та консистентність даних при виконанні паралельних запитів.

Підтримка об’єктно-реляційного відображення. PostgreSQL надає можливості об’єктно-орієнтованого програмування, що дозволяє користувачам використовувати об’єкти, класи та успадкування в базі даних.

Висока надійність та доступність. PostgreSQL надає розширені можливості реплікації даних та автоматичного відновлення після збоїв, що гарантує високу доступність та надійність даних.

Порівняно з NoSQL базами даних, PostgreSQL має більш розвинуті можливості щодо структурування та забезпечення цілісності даних. Зокрема, PostgreSQL має можливість виконувати складні JOIN операції, які дозволяють об’єднувати дані з

різних таблиць. PostgreSQL також підтримує ACID-транзакції (Atomicity, Consistency, Isolation, Durability), що забезпечує надійність операцій з даними.

NoSQL бази даних, на відміну, мають тенденцію бути більш гнучкими щодо структури даних, але це може призвести до проблем з цілісністю даних. Крім того, NoSQL бази даних зазвичай не підтримують складні JOIN операції або транзакції ACID, які є ключовими для багатьох застосунків.

Враховуючи вищезазначені фактори, саме PostgreSQL є відмінним вибором для застосуванні у розробці системи для діагностики захворювань.

Реплікація бази даних для зчитування – створення реплікації бази даних для зчитування має ряд значущих переваг та особливостей, про які слід пам’ятати в процесі її впровадження. Основні переваги реплікації для зчитування включають:

Підвищення продуктивності. Реплікація бази даних для зчитування дозволяє розподіляти навантаження між основною та реплікованою базою даних, забезпечуючи краще використання ресурсів сервера та зменшуючи час очікування на відповідь при виконанні запитів.

Масштабованість. Використання реплікації бази даних для зчитування сприяє вертикальному та горизонтальному масштабуванню системи, оскільки дозволяє під’єднувати додаткові репліки для розподілу навантаження та підтримки зростаючого обсягу користувачів та запитів.

Забезпечення доступності. Реплікація для зчитування забезпечує вищий рівень доступності, оскільки навіть у разі відмови основного сервера бази даних, репліковані сервери можуть продовжувати обслуговувати запити на зчитування.

Аналітика та звітність. Реплікація для зчитування дозволяє відокремити аналітичні та звітні операції від основної бази даних, зменшуючи навантаження на основний сервер та покращуючи продуктивність.

Основні моменти, про які слід пам’ятати в процесі створення реплікації бази даних для зчитування, включають:

Синхронізація даних. Важливо ретельно вибрати метод синхронізації між основною та реплікованою базою даних, враховуючи потреби системи та прийнятний

рівень затримки у синхронізації даних. Синхронізація може відбуватись в режимі реального часу, або ж дані можуть оновлюватися протягом певних проміжків часу.

Відмовостійкість. Система реплікації бази даних повинна бути прозорою та відмовостійкою, так щоб у разі збою основної бази даних, виконання запитів продовжувалося безперебійно на реплікованих серверах.

Обробка записів. Хоча репліковані бази даних використовуються в основному для операцій зчитування, важливо коректно обробляти операції запису, якщо вони допускаються. Зазвичай операції запису виконуються на основній базі даних, а потім реплікуються на всі репліки.

Балансування навантаження. Важливо правильно налаштувати балансування навантаження між основною та реплікованими базами даних, щоб оптимально використовувати ресурси і забезпечити високу продуктивність системи.

Безпека. Безпека даних повинна бути високим пріоритетом при реплікації бази даних. Потрібно вжити всіх необхідних заходів для захисту даних під час передачі, зберігання та обробки.

Реплікація бази даних для зчитування вимагає високого рівня планування, налаштування та управління. Однак при правильному впровадженні вона може значно покращити продуктивність, масштабованість та доступність системи.

Об’єктно-реляційне відображення (ORM – Object-Relational Mapping) є потужним інструментом у розробці програмного забезпечення, який пропонує ряд переваг:

Абстракція від SQL. ORM надає можливість відділити бізнес-логіку від взаємодії з базою даних, розробники можуть працювати з об’єктами, не думаючи про SQL. Це дозволяє зменшити кількість помилок та полегшити розробку.

Портативність коду. Різні СКБД використовують різні версії SQL. ORM автоматично генерує відповідний SQL для вибраної СКБД, що робить код більш портативним.

Швидкість розробки. Використання ORM дозволяє зменшити кількість коду, що потрібно написати, оскільки ORM надає багато зручних функцій "з коробки".

Об’єктно-орієнтоване програмування. ORM дозволяє використовувати всі переваги об’єктно-орієнтованого програмування (наслідування, поліморфізм, інкапсуляція), коли працюєте з базами даних.

Кешування. ORM може надати кешування на рівні сесії, що допомагає збільшити продуктивність застосунку.

Інтеграція з іншими технологіями. ORM може бути легко інтегрований з іншими технологіями та інструментами, такими як фреймворки розробки вебзастосунків.

Всі ці переваги забезпечують ефективність ORM як інструмента при роботі з реляційними базами даних у контексті об’єктно-орієнтованого програмування.

4.1.5 Презентаційний рівень

Презентаційний рівень – це перший рівень архітектури, з яким безпосередньо взаємодіє користувач. Його головною метою є надання користувачам зрозумілого, ефективного та приємного інтерфейсу для взаємодії з системою. Презентаційний рівень включає в себе елементи дизайну інтерфейсу, технології фронтенду та процеси, що забезпечують відображення даних та збір вводу від користувача.

Дизайн інтерфейсу. Це включає в себе елементи візуального дизайну, архітектури інформації, взаємодії користувача та досвіду користувача, що допомагають створити інтерфейс, який відповідає потребам користувача та контексту використання.

Технології фронтенду. Це включає в себе мови програмування та фреймворки, які використовуються для створення інтерфейсу користувача. На 2023 рік популярними технологіями є HTML, CSS, JavaScript, а також такі фреймворки та бібліотеки, як React.js, Vue.js, Angular та Svelte.

Відображення даних та ввід користувача. Презентаційний рівень відповідає за отримання введення від користувача, його валідацію та перенаправлення до відповідного рівня бізнес-логіки або зберігання даних. Він також відповідає за отримання даних від цих рівнів та їх відображення користувачу у зрозумілому форматі.

Швидкодія. Оскільки презентаційний рівень – це те, з чим безпосередньо взаємодіє користувач, він повинен бути оптимізованим для швидкості та реактивності. Вебсторінка, яка довго завантажується або не реагує на дії користувача, може негативно вплинути на досвід користувача та загальну продуктивність системи.

Безпека. Презентаційний рівень також є важливим для забезпечення безпеки системи. Він повинен забезпечувати валідацію даних на стороні клієнта, безпечну передачу даних до сервера та захист від таких загроз, як міжсайтовий скриптинг (XSS) та підробка міжсайтових запитів (CSRF).

Варто відзначити, що хоча презентаційний рівень відповідає за відображення даних, він не повинен містити бізнес-логіку або безпосередньо взаємодіяти з базою даних. Замість цього він повинен використовувати API для взаємодії з іншими рівнями системи, забезпечуючи тим самим модульність, масштабованість та безпеку архітектури.

До основних модулів презентаційного рівня відносяться **шлюз API**, що зв'язує кінцевих клієнтів з набором внутрішніх сервісів, а також **фронтенд сервіс**, який виконує логіку інтерфейсної частини вебзастосунку у браузері.

З метою побудови презентаційного рівня для системи діагностики стану здоров'я пацієнтів пропонується використати такі технології:

- SvelteKit
- Spring Cloud Gateway

SvelteKit – це інноваційний фреймворк для розробки вебдодатків, який має низку суттєвих переваг [38-39], що роблять його привабливим для сучасних розробників.

Ефективність. SvelteKit, так само як і Svelte, компілюється на етапі збірки, замість виконання в браузері. Це робить ваш додаток швидшим і менш витратним за ресурсами.

Спрощене управління станом. В SvelteKit вбудований механізм управління станами, який дозволяє зберігати та маніпулювати станом додатку без необхідності використання сторонніх бібліотек.

Вбудована підтримка Server Side Rendering (SSR) та Static Site Generation (SSG). SvelteKit дає можливість використовувати як SSR, так і SSG, що дозволяє збільшити продуктивність додатків, покращити Search Engine Optimization (SEO) та забезпечити кращу підтримку для користувачів зі слабким інтернет-з'єднанням.

Модульність. SvelteKit використовує модульну структуру, яка дозволяє легко додавати та видаляти складові вашого додатку, зберігаючи код організованим і зрозумілим.

Простота розгортання. За допомогою адаптерів SvelteKit дозволяє легко розгортати ваш додаток на багатьох популярних платформах, таких як Vercel, Netlify, Begin та інших.

Повна інтеграція з Svelte. SvelteKit має повну сумісність з Svelte, що робить його максимально зручним для тих, хто вже використовує Svelte у своїх проектах.

Підтримка TypeScript. SvelteKit підтримує TypeScript "з коробки", що дозволяє використовувати всі переваги цього мовного розширення.

Усі ці переваги роблять SvelteKit ефективним інструментом для сучасної розробки вебдодатків.

Spring Cloud Gateway, що є частиною екосистеми Spring Cloud, пропонує ряд значущих переваг [40-41] для створення шлюзу API у вебзастосунках.

Висока продуктивність. Технологія Spring Cloud Gateway заснована на Project Reactor і Spring WebFlux, що дозволяє їй обробляти великі потоки даних та паралельні запити ефективно та без блокування.

Інтеграція з Spring Cloud Discovery. Spring Cloud Gateway може використовувати служби виявлення з Spring Cloud Discovery для динамічної маршрутизації до кінцевих сервісів.

Гнучкість налаштування маршрутів. Spring Cloud Gateway дозволяє конфігурувати маршрути за допомогою анотацій в коді, або ж за допомогою зовнішнього файлу конфігурації, надаючи значну гнучкість для адаптації до потреб конкретної архітектури.

Фільтри і перехоплювачі. Spring Cloud Gateway пропонує вбудовані фільтри та перехоплювачі для маніпулювання вхідними та вихідними запитами, що забезпечує можливість попередньої та постобробки запитів.

Безпека. Інтеграція Spring Cloud Gateway з Spring Security дозволяє забезпечити високий рівень безпеки API за допомогою різних механізмів аутентифікації та авторизації.

Підтримка Circuit Breaker. Spring Cloud Gateway підтримує патерн Circuit Breaker для попередження відмов системи при відмові окремих мікросервісів.

Ці характеристики забезпечують Spring Cloud Gateway потужним, гнучким та надійним інструментом для управління вебсервісами у сучасних розподілених системах.

4.1.6 Рівень бізнес-логіки

Рівень бізнес-логіки – це серце будь-якої вебсистеми. Цей рівень містить в собі код, який виконує ключові функції системи, обробляє вхідні дані від презентаційного рівня та взаємодіє з рівнем зберігання даних для зберігання та отримання інформації. Розглянемо детальніше деякі ключові аспекти рівня бізнес-логіки.

Обробка бізнес-правил. Це ядро рівня бізнес-логіки. Бізнес-правила – це код, який визначає, як система повинна виконувати ключові операції, такі як обробка транзакцій, валідація даних, виконання алгоритмів, проведення обчислень та інше.

Взаємодія з рівнем зберігання даних. Рівень бізнес-логіки відповідає за забезпечення правильної взаємодії з рівнем зберігання даних. Це включає в себе відправку запитів до бази даних для зберігання та отримання інформації, а також обробку цієї інформації для подальшого використання у системі.

Безпека. Рівень бізнес-логіки відіграє важливу роль у забезпеченні безпеки системи. Це включає в себе валідацію даних, що надходять від презентаційного рівня, для запобігання таким загрозам, як SQL-ін’єкції, а також забезпечення авторизації та аутентифікації.

Масштабованість. Оскільки рівень бізнес-логіки відповідає за обробку ключових операцій системи, важливо забезпечити його масштабованість. При збільшенні навантаження на систему рівень бізнес-логіки повинен мати змогу

ефективно обробляти збільшення об’єму даних та запитів. Це можна досягти за допомогою таких технологій, як вебсервіси та контейнеризація, що дозволяють горизонтальне масштабування.

Надійність та відмовостійкість. Рівень бізнес-логіки також повинен бути надійним та здатним відновитися після помилок. Це можна досягти за допомогою технік, таких як використання перевірених бібліотек та фреймворків, реалізація відмовостійких алгоритмів, створення резервних копій та їх відновлення, та використання автоматичних тестів для перевірки надійності коду.

Таким чином, рівень бізнес-логіки є ключовим елементом архітектури вебсистеми, відповідальним за обробку бізнес-правил, взаємодію з рівнем зберігання даних, забезпечення безпеки, масштабованості та надійності системи.

Основними модулями рівня бізнес-логіки є **сервіс предметної області**, що оперує сутностями та функціоналом деякої предметної області (в нашому випадку системою діагностики захворювань), а також **сервіс підтримки прийняття рішень**, який здійснює обчислення результату відповідно до деякої математичної моделі (у нашому випадку це нечітка логіка).

Для гарантування і виконання основних вимог системи діагностики пацієнтів пропонується застосувати технології та супутні бібілотеки мов програмування:

- Java
- Python

Використання **Java** як сервіса предметної області для системи підтримки прийняття рішень пропонує безліч переваг [42-44], що підвищують ефективність та гнучкість в процесі розробки. Деякі з цих переваг включають:

Широка підтримка і зрілість технології. Java – це мова програмування, що є великою частиною розробки програмного забезпечення вже багато років. Це означає, що є багато інструментів, бібліотек, і фреймворків, що спрощують розробку і забезпечують широкий спектр функцій.

Масштабованість та продуктивність. Java відома своєю здатністю до масштабування, що робить її ідеальним вибором для великих систем підтримки

прийняття рішень. Вона може обробляти великі обсяги даних і є ефективною при виконанні складних обчислень.

Безпека. Java пропонує сильну підтримку безпеки на рівні мови і платформи, включаючи підтримку різних механізмів авторизації і аутентифікації, а також криптографічних бібліотек.

Крос-платформність. Програми, написані на Java, можуть працювати на будь-якій операційній системі, що підтримує Java Runtime Environment (JRE). Це забезпечує гнучкість розгортання і спрощує розподілену роботу.

Підтримка об'єктно-орієнтованого програмування (ООП). Java – це об'єктно-орієнтована мова програмування, що сприяє створенню масштабованого, гнучкого та легко розширюваного коду. Це є особливо важливим у контексті сложних систем підтримки прийняття рішень.

Таким чином, використання Java як сервіса предметної області для системи підтримки прийняття рішень пропонує багато переваг, враховуючи зрілість технології, масштабованість, безпеку, крос-платформність, а також підтримку ООП.

Використання **Python** як сервіса обчислення і аналізу даних для системи підтримки прийняття рішень пропонує численні переваги [45-47], що забезпечують його високий рівень придатності та ефективності для цих завдань. Ось деякі з них:

Можливості аналізу даних. Python пропонує різноманітну екосистему бібліотек та інструментів для аналізу даних, включаючи Pandas, NumPy, SciPy та Matplotlib. Ці бібліотеки пропонують функціональність для обробки даних, статистичного аналізу, машинного навчання, візуалізації даних та багато іншого.

Зрозумілість коду. Python славиться своєю простотою та зрозумілістю, що забезпечує швидку розробку та легкість підтримки коду. Це дозволяє дослідникам та аналітикам зосередитися на розумінні та інтерпретації даних, а не на деталях реалізації коду.

Спільнота і підтримка. Python має одну з найбільших і найактивніших програмістських спільнот, що означає, що ви отримаєте широку підтримку в процесі розробки. Це також означає, що майже на будь-яку проблему, з якою ви зіткнетесь, вже існує відповідь або рішення.

Масштабованість та інтеграція. Python може легко інтегруватися з іншими мовами програмування та інструментами, що робить його добрим вибором для складних систем.

Отже, Python пропонує вражаючий набір переваг, які відрізняють його як прекрасний інструмент для обчислень і аналізу даних в контексті системи підтримки прийняття рішень.

4.1.7 Рівень моніторингу

Рівень моніторингу – це критичний компонент архітектури будь-якої вебсистеми, який дозволяє командам відстежувати роботу системи в реальному часі та виявляти проблеми ще до того, як вони вплинуть на користувачів. Він також допомагає командам визначати тенденції, аналізувати ефективність системи та планувати майбутнє масштабування. Рівень моніторингу складається з трьох основних компонентів: моніторингу метрик, логування та трасування.

Моніторинг метрик включає збір та аналіз числових значень, які характеризують роботу системи. До цих метрик можуть відноситися навантаження на CPU, використання пам’яті, кількість запитів до бази даних, час відгуку сервера, кількість активних користувачів, кількість помилок і т.д. Системи моніторингу, як правило, вміють візуалізувати ці дані, що полегшує процес аналізу та виявлення аномалій.

Логування є процесом збору, зберігання та аналізу текстових повідомлень, які генеруються системою під час її роботи. Логи можуть містити інформацію про помилки, попередження, інформаційні повідомлення та діагностичну інформацію. Вони є критичними для виявлення та вирішення проблем.

Також варто враховувати важливість сповіщення в рамках рівня моніторингу. Система сповіщення повинна бути налаштована таким чином, щоб сповіщати відповідних людей або системи про важливі події або проблеми, які виникли в системі.

Використання рівня моніторингу дозволяє підтримувати стабільність системи, швидко виявляти та вирішувати проблеми, а також робити обґрунтовані рішення щодо оптимізації та масштабування системи.

Рівень моніторингу складається з трьох основних модулів: **сервіс логування**, який дозволяє агрегувати та здійснювати пошук логів; **збірник метрик**, що допомагає відстежувати основні показники та стан процесів; **візуалізатор метрик**, який спрощує обробку та сприйняття ключових метрик системи.

Для впровадження ефективного рівня моніторингу нашої системи пропонується використати наступні технології:

- ELK Stack
- Prometheus
- Grafana

ELK Stack, що складається з Elasticsearch, Logstash і Kibana, стала важливим інструментом для аналізу даних в реальному часі, особливо коли мова йде про обробку великих обсягів лог-файлів або потокових даних. Використання ELK Stack надає декілька значущих переваг [48-50]:

Швидкість пошуку та аналізу. Elasticsearch використовує Lucene для обробки індексації та пошуку, що забезпечує високу швидкість і гнучкість при роботі з великими обсягами даних.

Гнучкість. ELK Stack дозволяє обробляти та аналізувати дані різного формату і з різних джерел. Це дозволяє використовувати дану технологію для широкого спектру застосувань, включаючи аналіз журналів, моніторинг процесів, аналіз безпеки та ін.

Масштабованість. Технологія Elasticsearch була розроблена з урахуванням горизонтального масштабування, що означає, що вона може ефективно обробляти великі обсяги даних.

Інтерактивна візуалізація. Kibana пропонує багатий набір можливостей для візуалізації даних, що спрощує аналіз і допомагає швидко виявляти закономірності або аномалії в даних.

Надійність і відновлення. Elasticsearch має вбудовані механізми для керування відмовами та відновлення, що забезпечують стійкість до відмов у випадку проблем з апаратним забезпеченням або мережею.

Отже, ELK Stack являє собою потужний інструмент для аналізу великих обсягів даних, забезпечуючи швидкий пошук, гнучкість, масштабованість, візуалізацію та надійність.

Prometheus, продукт для системного моніторингу та сповіщення, який початково був розроблений в SoundCloud, став одним із ключових інструментів для нагляду за метриками в екосистемі хмарних обчислень. Цей інструмент має кілька значних переваг [51-53]:

Мультивимірність. Prometheus використовує мультивимірні дані у вигляді часових рядів, що дозволяє користувачам виконувати складний аналіз на основі різних вимірів.

Гнучкість запитів. Prometheus надає мову запитів PromQL, що дозволяє користувачам вибирати та агрегувати часові ряди даних у реальному часі.

Вбудовані механізми сповіщення. Prometheus має вбудований Alertmanager, що дозволяє визначати гнучкі правила сповіщення та обробляти сповіщення про події.

Масштабованість і надійність. Prometheus включає в себе аспекти, які дозволяють йому масштабуватися і витримувати відмови, включаючи декомпозицію на сервіси, автономність вузлів та високий рівень надійності під час мережевих перебоїв.

Підтримка багатьох екосистем. Prometheus має широку підтримку від множини мов програмування та служб, що робить його легко інтегрованим у різні системи.

Отже, Prometheus є потужним інструментом для моніторингу систем, що надає гнучкість, масштабованість, і глибокий аналіз даних.

Grafana є високопродуктивною відкритою платформою для аналітики і моніторингу, яка вже довгий час використовується у сфері DevOps для візуалізації даних та оцінки метрик в реальному часі. Ось декілька переваг використання Grafana [54-56]:

Велика сумісність з джерелами даних. Grafana підтримує інтеграцію з широкий спектром технологій, що зберігають часові дані, наприклад, Prometheus,

InfluxDB, Elasticsearch, Graphite, а також багато інших. Це дозволяє збирати дані з різних джерел та представляти їх у єдиному візуальному інтерфейсі.

Гнучкі можливості візуалізації. Grafana пропонує велику кількість варіантів візуалізації: від простих графіків і гістограм до складних теплових карт і діаграм. Це дозволяє адаптувати панелі дашбордів під конкретні потреби користувача.

Сповіщення. Grafana має вбудовану систему сповіщень, що дозволяє визначати умови сповіщення на основі даних метрик. Крім того, вона підтримує багато каналів сповіщень, таких як Email, Slack, PagerDuty та інші.

Підтримка шаблонів. Grafana дозволяє створювати шаблони дашбордів, що спрощує повторне використання і ділиться наборами моніторингових панелей.

Використання Grafana дозволяє створювати ефективні рішення для моніторингу та аналітики, адаптовані під конкретні потреби вашої організації.

Схематично всі рівні вебзастосунку з основними модулями та обраними до них технологіями можна представити діаграмою (рис. 4.2):

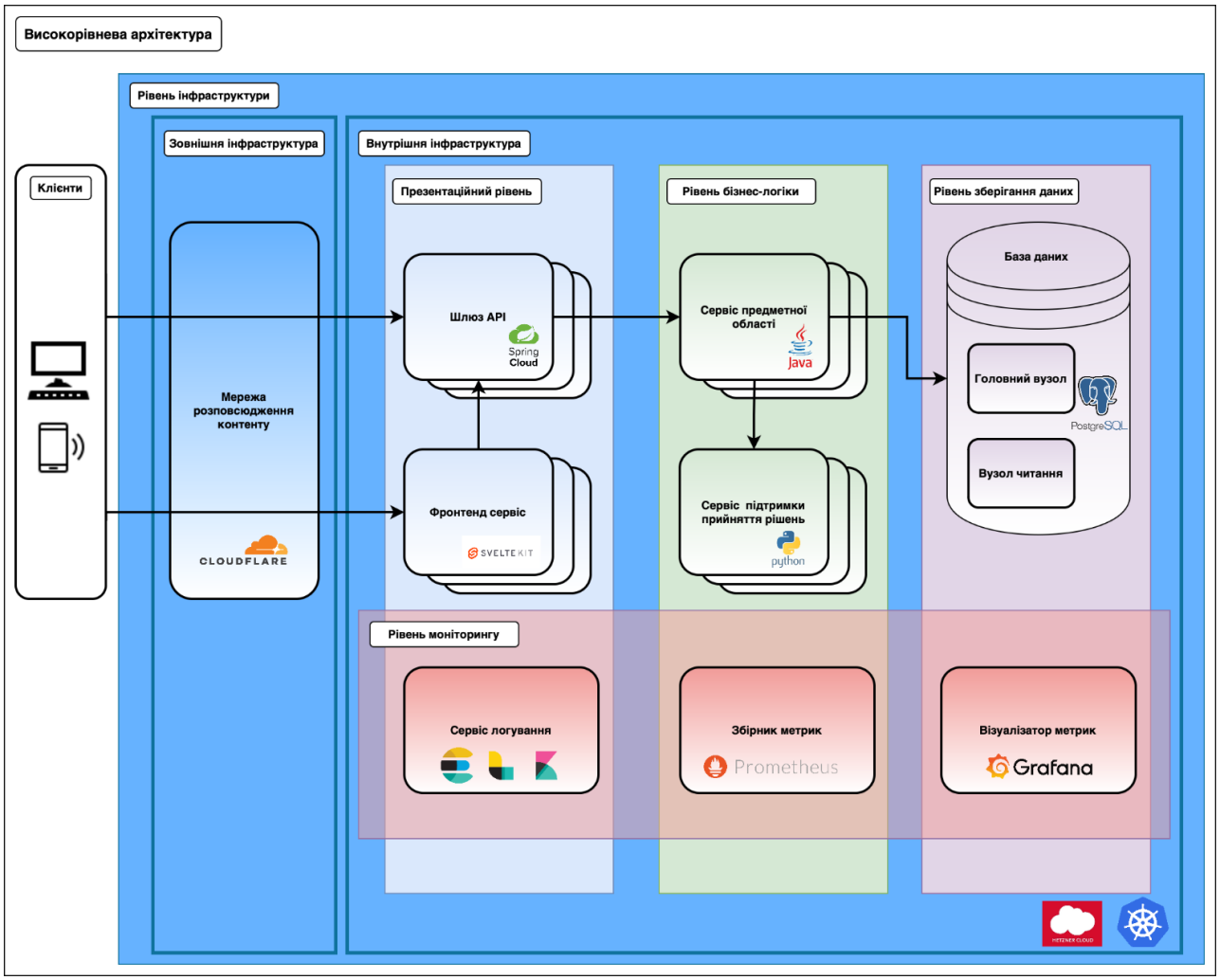

Рис. 4.2. Деталізована високорівнева архітектура вебзастосунка для системи підтримки прийняття рішень

4.2 Моделювання рівня зберігання даних відповідно до предметної області

Необхідно зазначити, що моделювання даних має суттєво предметно-орієнтований характер. Концептуальні сутності, які представляють структуру даних, які обробляються системою, суттєво залежать від специфіки предметної області та цілей самої системи.

Важливо підкреслити, що кількість, типи та взаємозв'язки сутностей предметної області можуть значно відрізнятися, коли ми переходимо від одного проекту до іншого. Це робить моделювання даних складним адаптивним процесом, який вимагає глибокого розуміння предметної області та потреб користувачів.

Беручи до уваги конкретні вимоги до системи діагностики медичних станів пацієнтів, пропонується наступна схема даних (рис. 4.3):

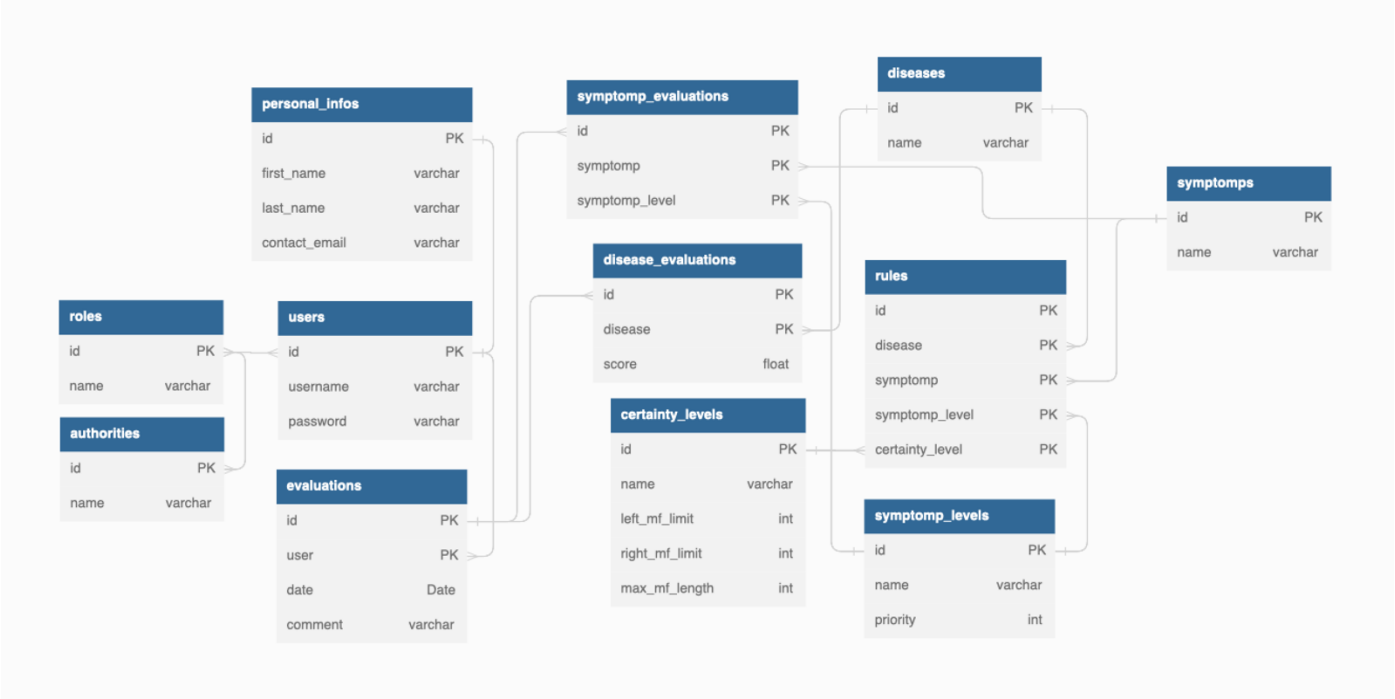

Рис. 4.3. Схема бази даних для системи діагностики захворювань пацієнтів

Ця схема була розроблена з урахуванням унікальних потреб системи підтримки прийняття рішень для діагностики стану здоров'я пацієнтів, зокрема потреби в ефективній організації та обробці великих обсягів медичних даних. Вона відображає

ключові сутності та їх взаємозв'язки в контексті цієї системи, забезпечуючи таким чином спрощення процесу обробки даних та підвищення загальної ефективності системи. Дана схема використовує та розширює початкові сутності надані у 1.4.1.

4.3 Деталізація життєвого циклу сучасного програмного рішення

4.3.1 Неперервна інтеграція

Неперервна інтеграція (Continuous Integration, CI) [57-58] є сучасним підходом в області розробки програмного забезпечення, який спрямований на автоматизацію та постійне вдосконалення процесу розробки.

Приступаючи до роботи над зміною програмного коду, розробник створює локальну копію поточної спільної кодової бази. Оскільки інші розробники паралельно можуть оновлювати спільну кодову базу, ця локальна копія поступово перестає відображати актуальний стан коду програмного рішення. До таких оновлень можна віднести зміни не тільки у існуючих даних спільної бази коду, але також додавання нового коду, нових програмних бібліотек та інших ресурсів, які створюють залежності та потенційні конфлікти.

Чим довше розробник буде працювати над своєю локальною копію і відкладати злиття (merge) змін в спільну кодову базу, тим більше конфліктів буде виникати під час такого злиття, призводячи до так званого «інтеграційного пекла» [59-60]. У найгіршому випадку різниця між локальною копією та спільною кодовою базою буде настільки великою, що злиття змін може стати неможливим і потребуватиме повної переробки завдання розробника, яка вимагатиме додаткового часу і вплине на кінцевий час виконання завдання.

Центральна ідея неперервної інтеграції полягає у зворотньому – прискорити постійне злиття змін коду, внесених розробниками, в одну спільну кодову базу кілька разів на день. Це дозволяє на ранньому етапі виявляти та усувати інтеграційні конфлікти між змінами, що прискорює процес розробки.

До основних етапі неперервної інтеграції відносяться:

Інтеграція коду. Розробники регулярно (кілька разів на день) відправляють свої зміни в спільну кодову базу.

Автоматична компіляція. Після внесення змін у спільній кодовій базі система неперервної інтеграції автоматично компілює код, щоб переконатися, що він не має синтаксичних чи інших помилок, що унеможливлюють збірку та запуск програми.

Автоматичне тестування. Після компіляції система неперервної інтеграції запускає набір автоматизованих тестів на поточній версії програми. Зазвичай таке тестування включає модульні тести, інтеграційні тести, функціональні тести та інші.

Додатково розглядають і використовують розширення неперервної інтеграції через такі підходи, як неперервна доставка (Continuous Delivery) [61] або неперервне розгортання (Continuous Deployment) [62]. Ці обидва підходи спрямовані на автоматизацію процесу доставки готового продукту до кінцевого користувача.

Неперервна доставка. Після проходження всіх етапів неперервної інтеграції зміни готові до ручного розгортання. Неперервна доставка надає методи та засоби для ручного розгортання будь-якої версії програмного забезпечення на необхідне середовище у будь-який час.

Неперервне розгортання. На відміну від неперервної доставки неперервне розгортання не потребує ручного запуску, нові зміни у програмному рішенні розгортаються на необхідне середовище повністю автоматично. Це забезпечує максимальну автоматизацію та зменшує вплив людського фактора на процес розгортання.

Використання підходів неперервної інтеграції, доставки та розгортання надає ряд значущих переваг:

Підвищення ефективності роботи. Автоматизація рутинних процесів, таких як компіляція, тестування та розгортання, дозволяє командам розробників зосередитися на основних завданнях щодо створення та оптимізації програмного коду.

Раннє виявлення та усунення помилок. Завдяки швидкій інтеграції та постійному тестуванню коду в процесі розробки можна виявляти та усувати помилки на ранніх стадіях, що сприяє зниженню витрат на їх виправлення.

Скорочення часу впровадження продукту на ринок. Автоматизовані процеси доставки та розгортання дозволяють швидше реагувати на зміни ринкових умов, а також оперативно впроваджувати нові версії програмного рішення.

Підвищення якості програмного рішення. Постійна перевірка та автоматизація гарантують, що кожна версія продукту відповідає високим стандартам якості перед її розгортанням у кінцевому середовищі.

Забезпечення послідовності та прозорості. Автоматизовані процеси забезпечують однорідність та послідовність дій на всіх етапах розробки, а також гарантують прозорість робочих процесів для всіх учасників команди.

Мінімізація ризиків. Неперервне розгортання на різних середовищах зменшує ризики, пов’язані з впровадженням нових версій продукту, завдяки поетапному тестуванню та верифікації.

Оптимізація взаємодії команд. Постійна інтеграція сприяє покращенню комунікації між учасниками команди, а також оптимізує процеси взаємодії розробників, тестувальників та інших спеціалістів.

Неперервна інтеграція є важливим елементом сучасних методологій розробки, яка забезпечує швидкість, гнучкість та якість програмного забезпечення. Завдяки автоматизації ключових процесів, команди можуть ефективніше реагувати на зміни, вносити їх та постійно вдосконалювати свої програмні рішення. Реалізація конкретного підходу неперервної інтеграції залежить від обраних технологій та предметної області, але можна виділити наступні основні компоненти, які так чи інакше притаманні всім підходам:

- Система контролю версій
- Середовища розгортання різних рівнів
- Конвеєри

Нижче приведена деталізація кожного з компонентів, а також опис їх спільного застосування при розробці системи підтримки прийняття рішень при діагностуванні стану здоров’я пацієнтів.

4.3.2 Система контроля версій

Ручне управління спільною кодовою базою є складним та затратним процесом. Для оптимізації цього процесу сучасні підходи розробки використовують спеціальні компоненти та технології, які називаються системами контроля версій.

Система контроля версій (Version Control System) є важливим інструментом в області розробки програмного забезпечення, що дозволяє вести облік, стежити за змінами та управляти версіями файлів або наборів файлів протягом тривалого часу. Це забезпечує можливість відновлення попередніх версій, порівняння та аналіз змін, а також співпрацю між різними розробниками [63].

До основних типів систем контролю версій відносяться:

Локальні системи контролю версій. Ці системи працюють на одному комп'ютері та зберігають змінені версії файлів локально. Вони можуть використовувати просту базу даних для ведення обліку всіх змін.

Централізовані системи контролю версій. Ці системи використовують єдиний сервер, на якому зберігаються всі версії файлів, та декілька клієнтів, які отримують копії файлів з центрального сервера. Зразками таких систем є, наприклад, Subversion [64] та Perforce [65].

Розподілені системи контролю версій. У таких системах кожен клієнт або робоча станція має свою повну копію репозиторію, що дозволяє працювати локально та синхронізуватися з іншими репозиторіями. Зразками таких систем є, наприклад, Git [66], Mercurial [67] та Bazaar [68].

Переваги використання систем контролю версій:

Відновлення версій. Система контролю версій дозволяє легко відновлювати попередні версії файлів або проекту в цілому, що є корисним у випадках помилок або необхідності повернення до попереднього стану.

Співпраця. Завдяки системам контролю версій, декілька розробників можуть ефективно співпрацювати над одним проектом, вносячи зміни паралельно.

Моніторинг змін. Всі зміни зберігаються з відомостями про автора, датою та коментарями, що полегшує розуміння історії змін.

Безпека. Система контролю версій забезпечує захист від втрати даних завдяки резервному копіюванню та можливості відновлення.

Гнучкість. Розгалуження та злиття дозволяють розробникам працювати над новими функціями або виправленнями без впливу на основний код.

У порівнянні з локальними та централізованими розподілені системи контролю версій представляють собою еволюційний крок у розробці даного класу компонентів. До основних значущих переваг розподілених систем контролю версій належать:

Локальна робота. Розподілені системи дозволяють розробникам здійснювати більшість операцій (таких як фіксація, історія, злиття) локально, без необхідності постійного з’єднання з центральним сервером. Це прискорює робочі процеси та дозволяє працювати в умовах відсутності інтернету.

Підвищена стійкість до відмов. Оскільки кожен учасник має повну копію репозиторію, втрата даних або відмова одного сервера не призводить до втрати всієї інформації.

Гнучкість розгалуження та злиття. Розподілені системи, такі як Git, надають розширені можливості для розгалуження та злиття, що полегшує паралельну роботу над різними функціями або виправленнями.

Розподілена відповідальність. Відсутність єдиної "центральної" копії коду забезпечує розподілену відповідальність серед учасників, стимулюючи культуру співпраці.

Більш ефективне використання ресурсів. Розподілені системи, як правило, використовують ресурси (такі як мережевий трафік, місце на диску) більш ефективно через оптимізовані алгоритми зберігання та передачі даних.

Враховуючи вищезазначену інформацію саме Git було обрано, як технологію для компонента контролю версій для системи підтримки прийняття рішень при діагностуванні стану здоров’я пацієнтів.

У процесі розробки програмного забезпечення правильне управління гілками та релізами в системах контролю версій відіграє критично важливу роль. Використання попередньо визначеної схеми створення та розгалуження гілок, а також релізних копій, має наступні переваги:

Організованість та послідовність. Стандартні схеми гілок та створення релізів забезпечують однорідність у процесі розробки, дозволяючи команді мати чітке розуміння структури кодової бази та стану розробки.

Підтримка паралельної розробки. За допомогою розгалужень можна ефективно управляти різними версіями коду, що розробляються паралельно – наприклад, функціональними змінами, виправленнями помилок або експериментальними функціями.

Мінімізація ризиків. Розгалуження дозволяє командам тестувати нові функції або виправлення в ізольованому середовищі перед їх злиттям з основною гілкою.

Прозорість версіонування. Релізні копії допомагають визначити конкретні версії продукту, що спрощує процес розгортання та підтримки.

Ефективна комунікація: За допомогою стандартизованих назв гілок та релізних копій команда може швидко та ефективно комунікувати стосовно стану розробки, планів релізів та інших ключових аспектів проекту.

Спрощення моніторингу. Стандартні схеми гілок та тегів полегшують процес відслідковування змін, аналізу причинно-наслідкових зв’язків та перевірки відповідності стандартам проекту.

Існують різні схеми управління гілками, до найбільш відомих можна віднести просту GitHub flow [69] схему або більш складну Git flow [70]. В рамках роботи над системою підтримки прийняття рішень було використано схему, зображену на рис. 4.4. Дана схема сформована за допомогою обох вищезазначених підходів і є максимально простою, але при цьому надає широкі можливості для роботи і внесення паралельних змін у всі можливі версії програмного рішення.

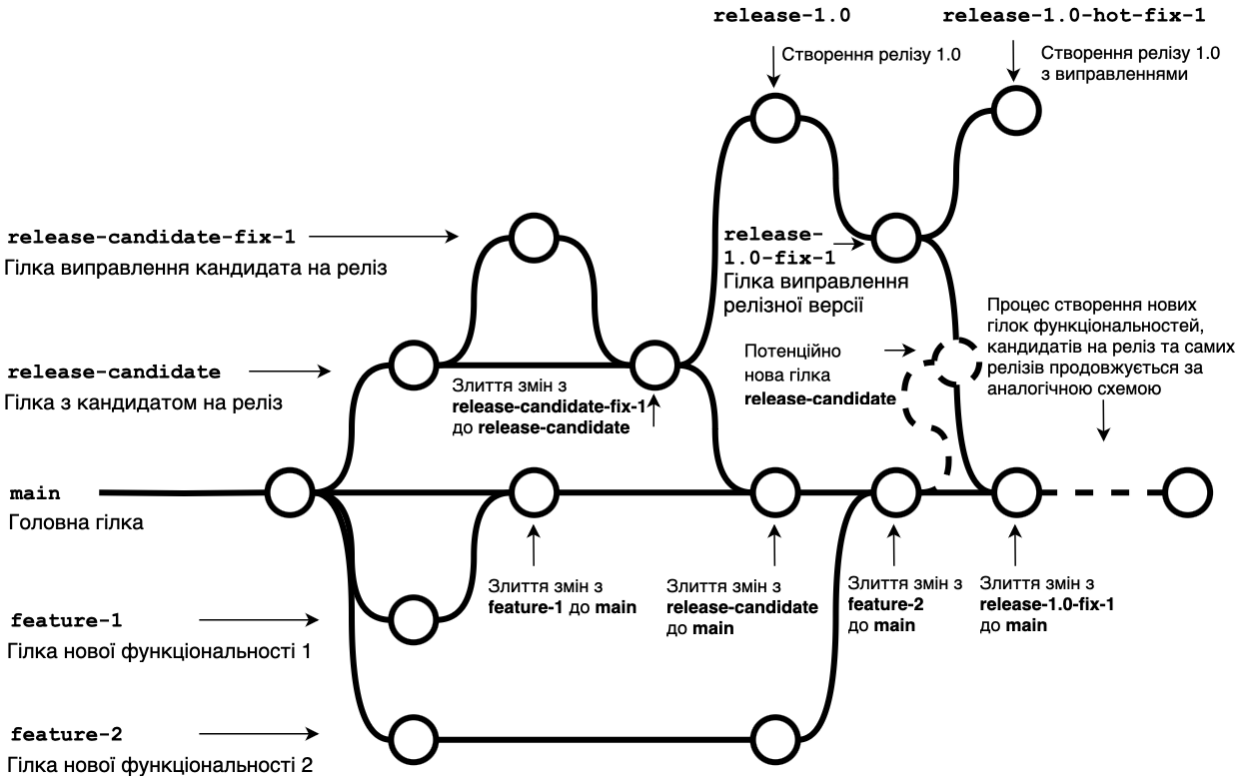

Рис. 4.4. Схема управління гілками

Процес розробки за такою схемою відбувається наступним чином:

1. Головною гілкою, яка зберігає найостанніші зміни, є гілка **main**. Усі зміни, які відбуваються в програмному рішенні рано чи пізно повинні потрапити у **main**, інакше є ризик не мати цих змін у майбутніх релізних версіях програми.
2. Як тільки розробник отримує завдання на зміну коду, то під це завдання створюється окрема гілка, наприклад, **feature-1** для нової функціональності. Ця гілка утворює розгалуження від **main**. Як тільки завдання виконано, то розробник повинен зробити злиття своїх змін назад до **main**. Варто зазначити, що інший розробник може паралельно виконувати інше завдання у рамках своєї окремої гілки **feature-2**. За таким підходом розробники не повинні блокувати роботу один одного.
3. Крім паралельної розробки і злиття змін назад до **main,** інша частина команди може працювати над стабілізацією поточної версії програмного рішення і готувати його до релізу. Для цього від гілки **main** утворюється гілка **release-candidate**. Над цією версією активно працюють тестувальники програмного

забезпечення та перевіряють якість програмного рішення, його готовність до релізу. Якщо під час такого тестування знаходиться дефект, то створюється гілка, наприклад, **release-candidate-fix-1** від гілки **release-candidate**. Варто зазначити важливість створення такої гілки саме від **release-candidate**, а не від **main**, інакше була б потреба завчасно знову створювати нову гілку **release-candidate** від гілки **main** і проводити нове повне тестування тому, що до такої версії могли потрапити вже нові зміни з гілки **main**, а не тільки виправлення дефекта. Під кінець життєвого циклу гілки r**elease-candidate**, всі зміни, які були додані до неї в рамках виправлення дефектів, повинні пройти злиття назад до гілки **main**, щоб не мати поточних виявлених дефектів у наступних версіях програмного рішення.

4. Як тільки програмне рішення з гілки **release-candidate** повністю протестоване**,** перед злиттям з гілкою **main**, створюється релізна версія, наприклад, версія **release-1.0**. Дана версія є кінцевою для користувачів. Якщо під час експлуатації релізної версії знаходиться додатковий дефект, який не був виявлений під час попередньої стадії тестування, то від версії **release-1.0** створюється гілка, наприклад, **release-1.0-fix-1**, в рамках якої цей дефект виправляється. На основі такої гілки з виправленим дефектом створюється нова релізна версія, наприклад, **release-1.0-hot-fix-1**. Виправлення дефекту з гілки **release-1.0-fix-1** обов’язково повинне бути злите назад до **main** для уникнення виникнення цього дефекту у наступних версіях програмного рішення. Варто зазначити, що якщо на момент злиття до гілки **main** існує нова гілка **release-candidate**, то злиття з гілки **release-1.0-fix-1** повинно відбутися до гілки **release-candidate**, у такому випадку гарантується, що навіть версія, яка готується до релізу, не буде мати цього дефекту.
5. Як тільки кількість змін і нового функціоналу в гілці **main** буде достатньо для створення нової версії програмного рішення, то процес повторюється і відгалужується нова гілка **release-candidate** для підготовки цієї версії.

4.3.3 Середовища розгортання різних рівнів

Розгортання програмних рішень в сучасних інформаційних системах є критичним процесом, який об’єднує розробку, тестування та впровадження програмного забезпечення до кінцевих користувачів. Вивчення специфікацій кожного з основних середовищ розгортання є важливим для забезпечення якості та безпеки програмних рішень. Рівні середовища розгортання утворюють ієрархічну структуру, кожен рівень ізольований один від одного [71]. На найвищому рівні знаходиться кінцеве середовище користувачів, яке повністю відображає реальний стан даних та процесів програмного рішення. На найнижчому рівні знаходиться розробницьке середовище, якому притаманне найбільше спрощення даних та процесів у порівнянні з кінцевим середовищем. В рамках роботи над системою підтримки прийняття рішень було обрано схему з використанням трьох рівнів середовищ – розробницьке, тестувальне та кінцеве. Детально опишемо кожне з них:

Розробницьке (Development). Первинне середовище, призначене для створення, модифікації коду та проведення експериментальних змін.

Переваги:

- Максимальна гнучкість для внесення змін та експериментів.
- Швидке виявлення та виправлення базових помилок.
- Здатність інтегруватися з іншими системами налагодження програм для виправлення помилок.

Недоліки:

- Потенційна нестабільність через часті оновлення.
- Відсутність гарантії сумісності з реальними умовами кінцевого середовища.

Застосування:

Часто використовується для неперервної інтеграції, автоматичного збирання та тестування для отримання швидкого аналізу поточного стану програмного рішення.

Тестувальне (Testing). Середовище, що імітує реальні умови використання програми, але ізолюється від кінцевого користувача. Призначене для проведення різних типів тестів (функціональних, навантажувальних тощо).

Переваги:

- Імітація реальних умов для забезпечення адекватності тестових сценаріїв.
- Можливість застосування автоматизованих тестових сценаріїв.
- Відокремленість від кінцевого середовища, що забезпечує безпеку даних.

Недоліки:

- Можливість виявлення помилок, які не проявляться в реальному кінцевому середовищі користувача.
- Витрати часу та ресурсів на підтримку тестового середовища.

Застосування:

Часто застосовується в неперервній доставці, неперервному розгортанні та автоматичному тестуванні.

Кінцеве (Production). Фінальне середовище, де програмне рішення використовується кінцевими користувачами для виконання реальних бізнес-операцій. Жодні зміни не можуть бути внесені без попереднього тестування.

Переваги:

- Надає конкретну бізнес-цінність для користувачів.
- Стабільна, оптимізована швидкодія та масштабованість.

Недоліки:

- Високий ризик для бізнесу при впровадженні дефектних змін.
- Високі вимоги до відмовостійкості, безпеки та відновлюваності.

Застосування:

Всі реальні бізнес-операції, взаємодія з користувачами, обробка даних та інтеграція з іншими кінцевими системами.

Кожне з вищезазначених середовищ відіграє свою унікальну роль у життєвому циклі програмного рішення. Вони впорядковані між собою, створюючи сприятливі умови для сталого розвитку програмного рішення від нової ідеї до фінальної реалізації.

Схематично всі три запропоновані рівні та ієрархічний зв’язок між ними можна показати діаграмою (рис. 4.5). Саме таке розбиття по типу середовищ було використано при розробці системи підтримки прийняття рішень.

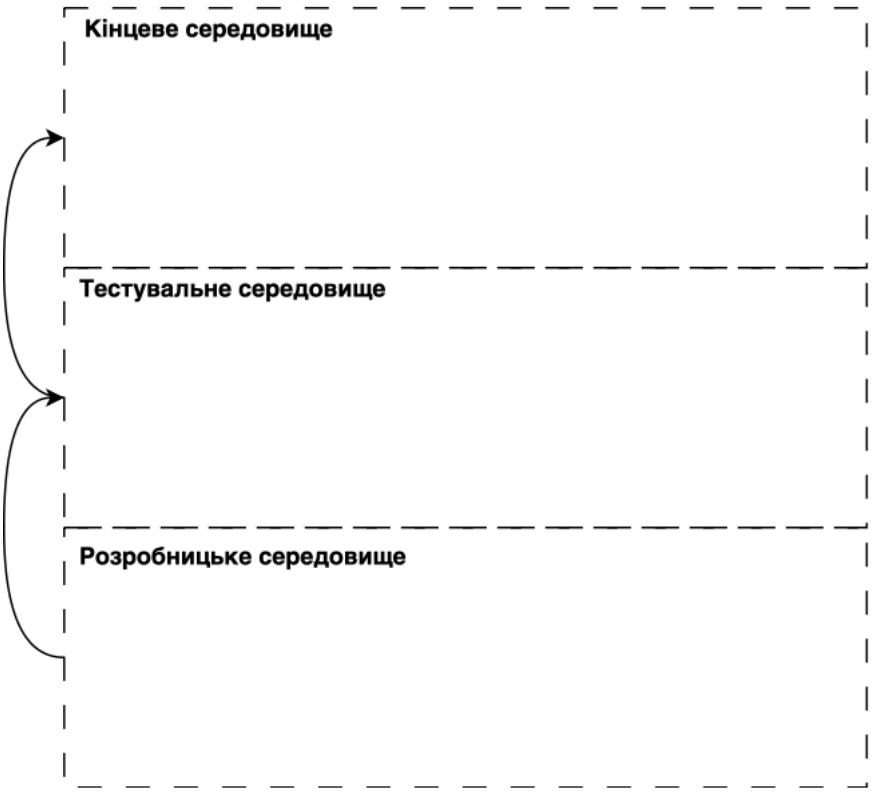

Рис. 4.5. Ієрархчіний зв'язок між середовищами

4.3.4 Конвеєри

Концепція **конвеєрів** (Pipelines) втілює принципи неперервної інтеграції. Це структуровані процеси, що дозволяють автоматизувати етапи розробки, тестування та релізу програмного рішення [72].

Конвеєри представляють собою послідовність етапів, кожен з яких виконує певну відокремлену дію, крім того, деякі етапи (відповідно до налаштування) можуть виконуватись паралельно. За допомогою різних комбінацій і послідовностей етапів можна утворювати різні конвеєри, які будуть вирішувати ту чи іншу задачу.

Зазвичай для запуску конвеєру використовуються системи контролю версій. Як тільки система контролю версій відслідкує нові зміни у спільній кодовій базі, то вона запускає відповідний конвеєр для виконання на версії програмного рішення з цими новими змінами.

Розглянемо загальну модель основних етапів конвеєра, який можна використовувати у сучасній розробці:

Компіляція. Даний етап автоматично створює виконувану версію програмного рішення з наданого коду, це забезпечує сумісність кодової бази та зменшує вірогідність синтаксичних чи інших помилок.

Модульне тестування. Цей етап включає автоматизоване виконання модульних тестів для перевірки правильності виконання основних можливостей програмного рішення.

Перевірка якості коду. Аналізує якість коду та інші метрики, такі як покриття коду тестами, виявлення потенційних вразливостей і технічного боргу.

Розгортання на розробницькому середовищі. Розгортання виконуваної версії програмного рішення на розробницьке середовище для початкового тестування та перевірки відповідності.

Розгортання на тестувальному середовищі. Розгортання протестованої виконуваної версії програмного рішення на тестове середовище для подальшого тестування та перевірки відповідності.

Розгортання на кінцевому середовищі. Відправка протестованого програмного рішення кінцевим користувачам, розгортання програмного рішення на кінцевому середовищі.

Автоматизоване функціональне тестування. Цей етап зазвичай включає автоматизоване виконання кінцевих функціональних тестів на розгорнутому середовищі, що перевіряють відповідність програмного рішення до початкової специфікації.

Ручне функціональне тестування. На практиці не всі тести можна автоматизувати, існують сценарії, які можливо протестувати лише в ручному режимі. Цей етап дозволяє призупинити виконання конвеєра та виконати таке тестування. Як тільки таке тестування буде проведено, то даний етап потрібно закінчити втручанням людини для подальшого розблокування наступних дій конвеєра.

На практиці конвеєри реалізуються за допомогою різних інструментів, таких як Jenkins [73], GitLab CI [74] та інші. Ці інструменти надають користувацькі інтерфейси та предметно-орієнтовану мову (Domain-Specific-Language) [75] для опису етапів конвеєрів, а також інтерфейси для інтеграції з різними сервісами і платформами.

Переваги конвеєрів:

Автоматизація процесів. Конвеєри дозволяють автоматизувати ряд процесів – від компіляції та тестування до розгортання та моніторингу. Це значно зменшує людський фактор та ризики, пов’язані з ручним втручанням.

Підвищення продуктивності. Автоматизація сприяє швидшому виконанню рутинних завдань, дозволяючи розробникам зосередитися на більш складних та креативних аспектах проекту.

Неперервна інтеграція та доставка. Конвеєри є ключовим елементом у неперервній інтеграції та доставці, що дозволяє розробникам частіше та більш ефективно вносити зміни у продукт.

Якість та надійність. Регулярне та автоматизоване тестування забезпечує високу якість продукту, виявляючи помилки на ранніх стадіях розробки.

Масштабованість. Конвеєри можуть бути легко масштабовані та налаштовані відповідно до потреб проекту.

Недоліки конвеєрів:

Складність налаштування. Налаштування та підтримка конвеєрів може бути складним та часовитратним процесом, особливо в великих та складних проектах.

Залежність від інструментів та платформ. Вибір платформ та інструментів для конвеєрів може вплинути на гнучкість та сумісність з іншими системами.

Ризик перевантаження ресурсів. Надмірна автоматизація може призвести до використання великої кількості ресурсів, особливо у великих та масштабних розробках.

Необхідність постійного моніторингу. Конвеєри потребують постійного моніторингу та оновлення, щоб забезпечити їх ефективну роботу.

Виклики з безпекою. Автоматизовані процеси можуть створити безпекові вразливості для інфраструктури розробки, якщо вони не належним чином налаштовані та захищені.

В рамках роботи над системою підтримки прийняття рішень було обрано GitLab CI як основний інструмент з налаштування конвеєрів. Додатково, з огляду на описану

вище загальну модель основних етапів конвеєрів для сучасних програмних рішень, було використано наступні конкретні конвеєри, зображенні на рис. 4.6.

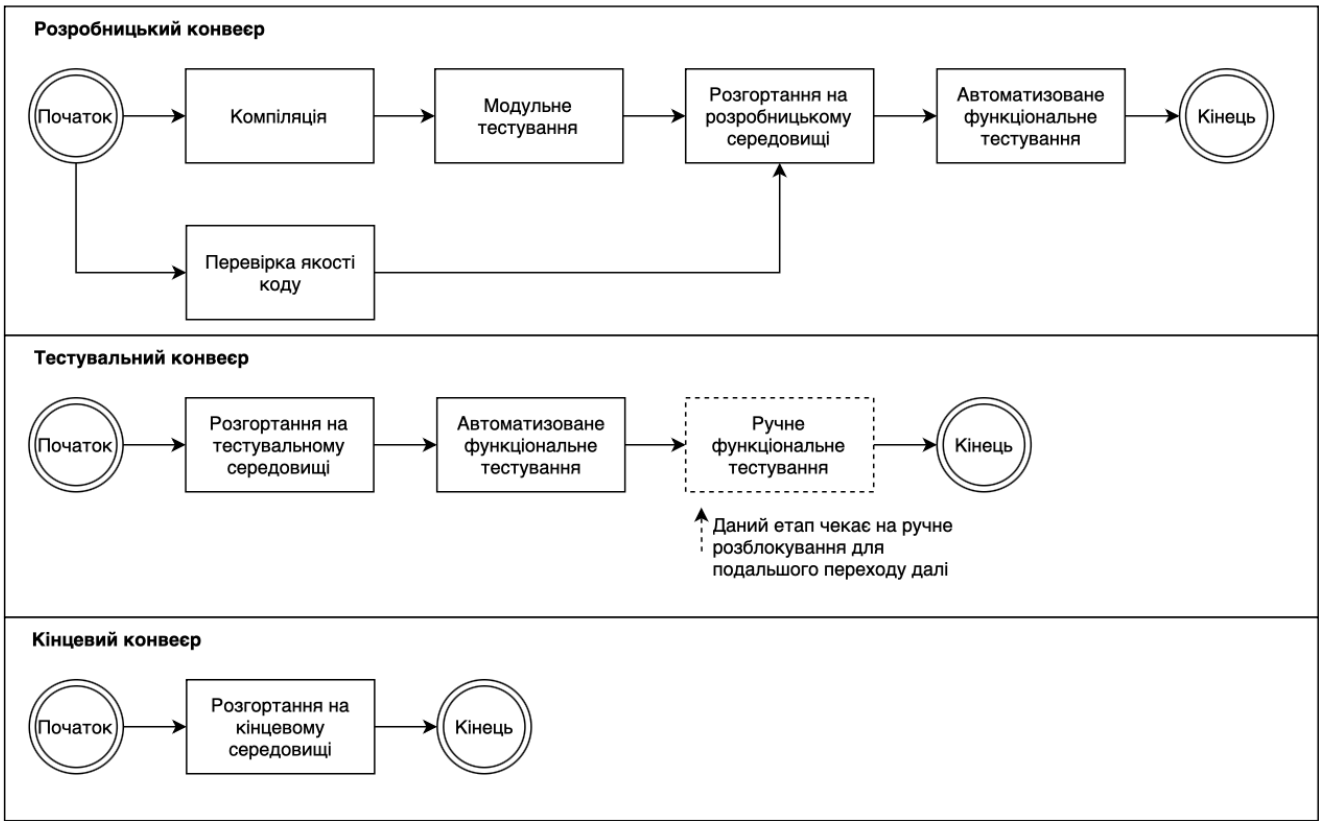

Рис. 4.6. Деталізація конвеєрів і їх основних етапів

Розробницький конвеєр. Даний конвеєр виконує більшість етапів. Він компілює програмне рішення, проводить перевірку якості його коду, розгортає виконуване програмне рішення на розробницькому середовищі та проводить початкове автоматизоване функціональне тестування. Додатковою особливістю даного конвеєру є виконання двох етапів одночасно – компіляції та перевірки якості коду (відповідно до схеми).

Тестувальний конвеєр. Цей конвеєр використовується для подальшого тестування програмного рішення на тестувальному середовищі. Особливістю такого конвеєра є наявність ручного етапу для ручного тестування. Зазвичай компіляція не входить до цього конвеєра, оскільки скомпільоване виконуване програмне рішення було вже створене розробницьким конвеєром.

Кінцевий конвеєр. Даний конвеєр найпростіший за кількістю етапів, але він не менш важливий. Він використовується для відправки та розгортання протестованого і готового програмного рішення на кінцевому середовищі.

4.3.5 Взаємодія основних компонентів неперервної інтеграції

Використовуючи наданий вище опис основних компонентів неперервної інтеграції, можна побудувати наступну загальну діаграму їх взаємодії, зображену на рис. 4.7.

Розглянута на діаграмі модель неперервної інтеграції дозволяє встановити чіткий ланцюг залежностей між версіями коду та їх станами в різних середовищах розгортання. Така структура сприяє підвищенню прозорості процесу розробки і полегшує виявлення та усунення помилок.

Система контролю версій є фундаментом для моніторингу змін і забезпечує відслідковування внесення змін до коду у різні гілки. Це дозволяє зберігати стабільність програмного рішення, одночасно впроваджуючи вдосконалення.

Середовища розгортання різних рівнів, представлені на діаграмі як ізольовані контури, гарантують, що зміни можуть бути протестовані в незалежних умовах, які імітують реальні сценарії. Такий підхід знижує ризик непередбачених помилок під час впровадження нових релізів на кінцеве середовище.

Конвеєри є механізмами, які автоматизують процес переходу від одного етапу розробки до іншого. Як видно з діаграми, конвеєри активізуються залежно від подій у системі контролю версій, забезпечуючи неперервність доставки змін.

Враховуючи вищенаведене, можна зазначити, що неперервна інтеграція (представлена діаграмою), є фундаментально важливою для ефективного управління життєвим циклом програмного рішення. Це особливо актуально при розробці систем підтримки прийняття рішень, де потрібна висока надійність та точність.

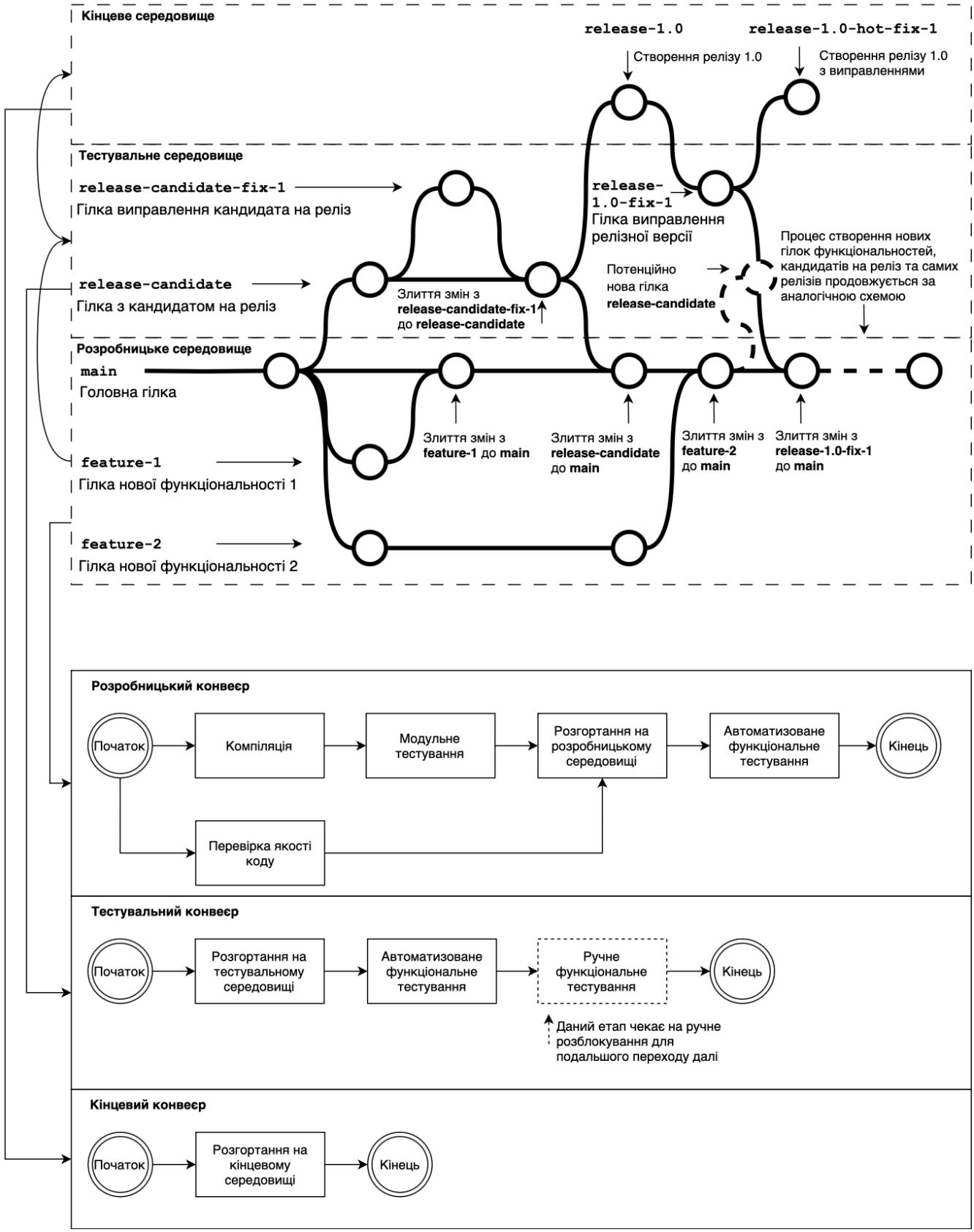

Рис. 4.7. Взаємодія основних компонентів неперервної інтеграції

4.4 Приклад роботи системи підтримки прийняття рішень

У даному розділі надамо кілька зображень використання системи під час роботи.

4.4.1 Сторінка автентифікації

Сторінка автентифікації дозволяє користувачам ввести свої облікові данні і підтвердити вхід до системи (рис. 4.8):

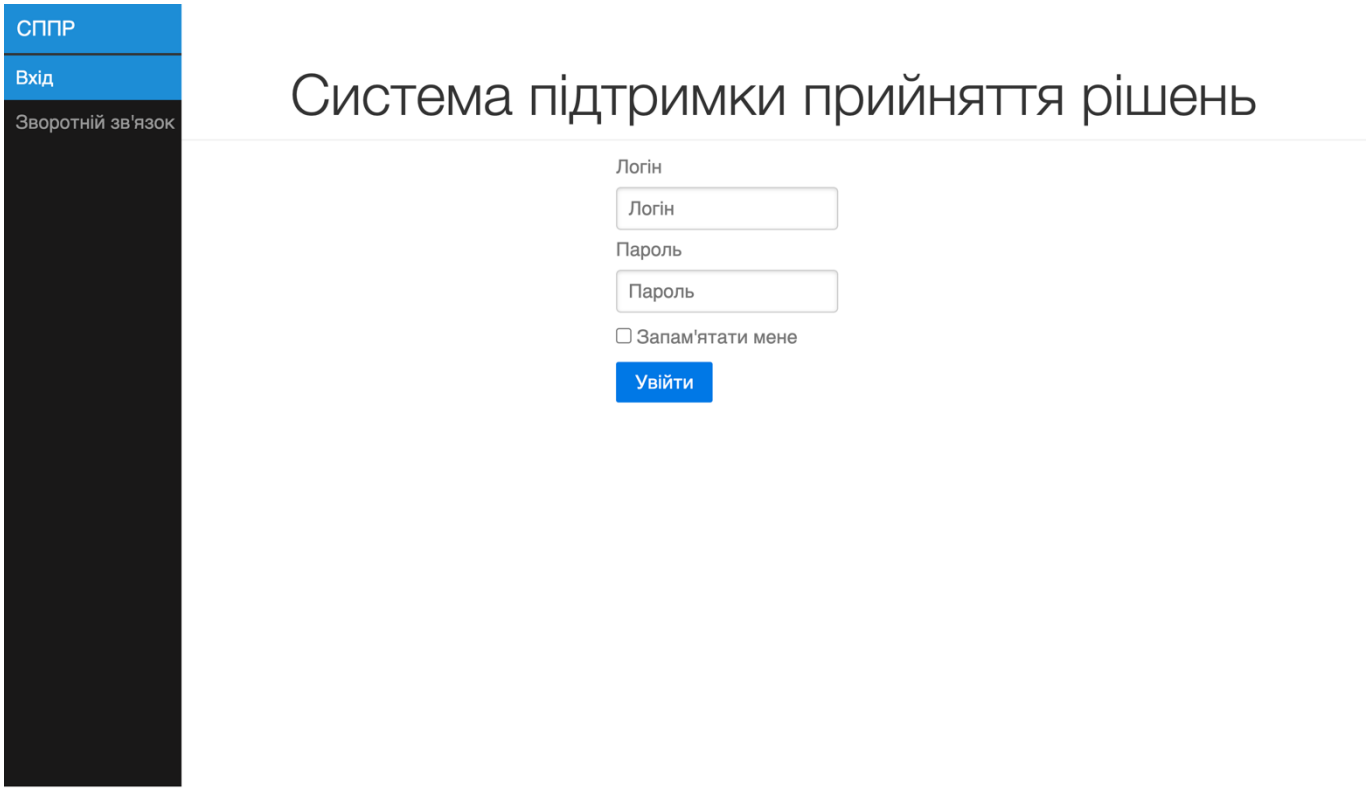

Рис. 4.8. Сторінка автентифікації

4.4.2 Сторінка проходження діагностики

Сторінка діагностики дозволяє користувачам ввести дані стосовно свого поточного самопочуття по списку симптомів.

По завершенню вводу користувач може надіслати дані симптомів для подальшої обробки системою (рис. 4.9):

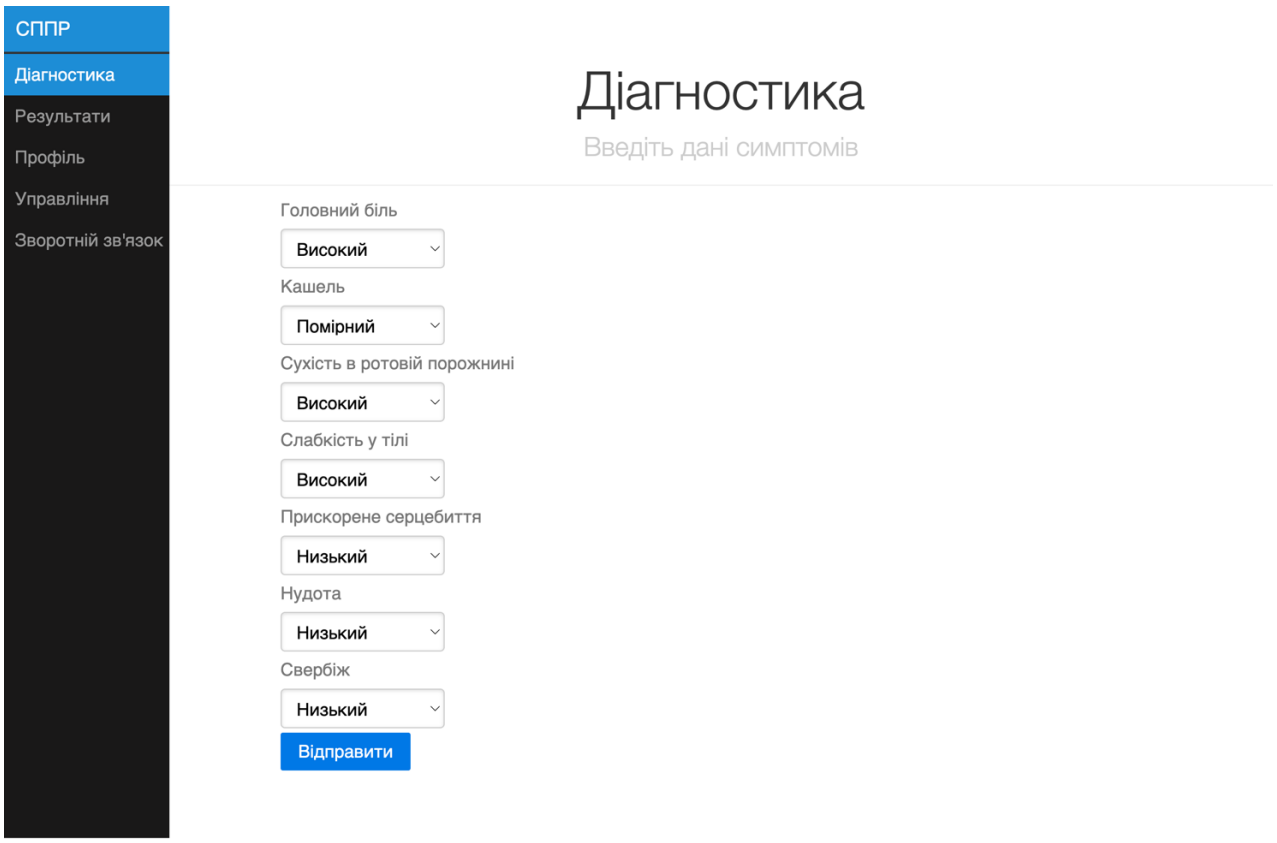

Рис. 4.9. Сторінка проходження діагностики

4.4.3 Сторінка перевірки результатів

Сторінка перевірки результатів дозволяє переглянути автоматичні оцінки системи та минулі відвідування (рис. 4.10):

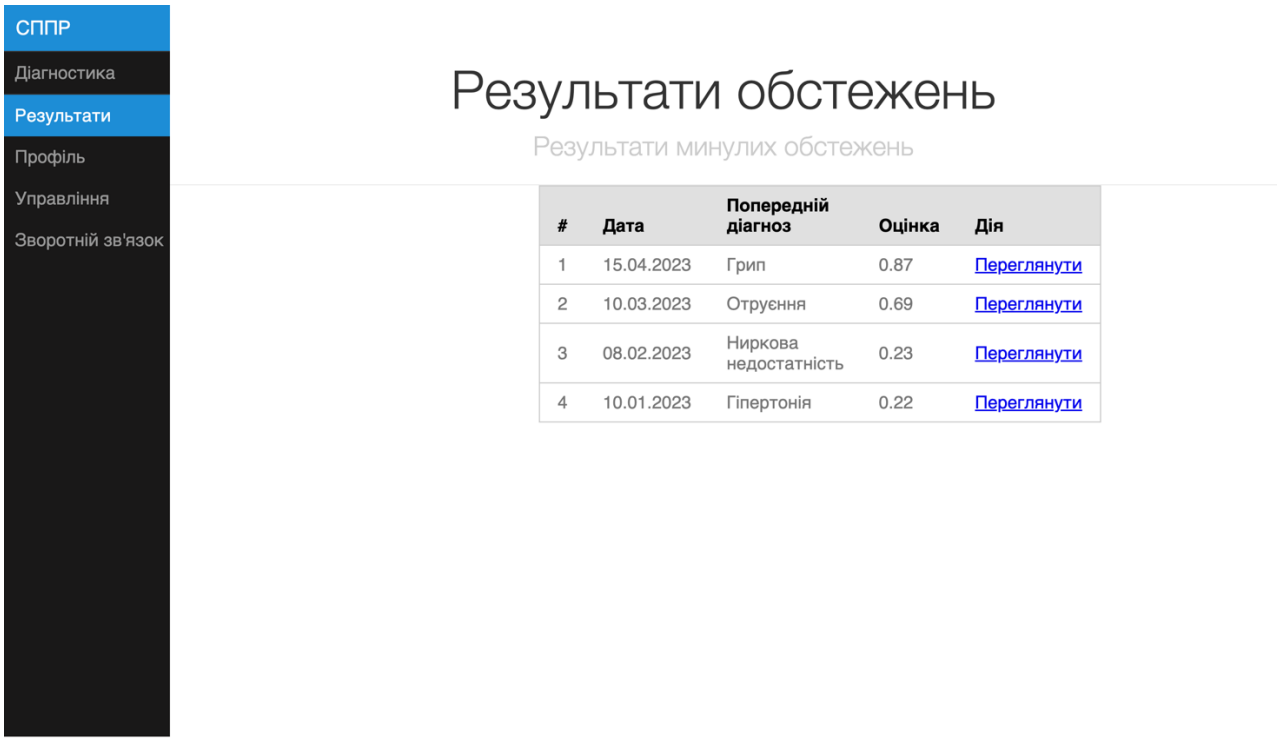

Рис. 4.10. Сторінка перевірки результатів

4.4.4 Сторінка управління системою

Сторінка управління системою дозволяє управляти основними сутностями системи (рис. 4.11):

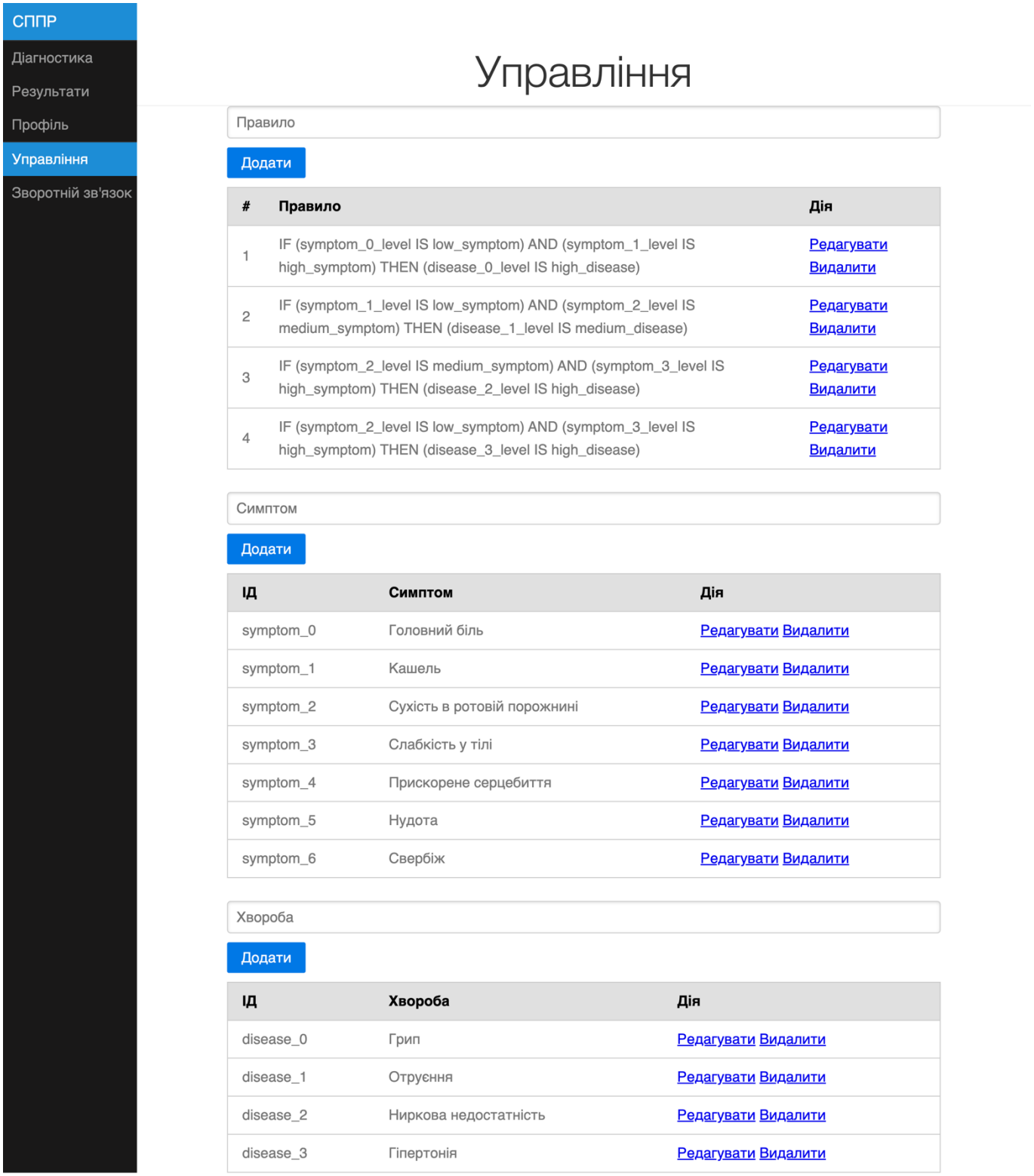

Рис. 4.11. Сторінка управління системою

ВИСНОВКИ

Системи підтримки прийняття рішень відіграють важливу роль в різних областях людської діяльності. Одна з ключових характеристик цих систем полягає у їхній здатності ефективно обробляти великі обсяги інформації за допомогою широкого діапазону технік штучного інтелекту. Ці техніки включають статистичний аналіз, аналіз даних, машинне навчання, оптимізацію, нечіткі множини та багато інших.

У світлі вищенаведеного, проблема розробки високоефективних систем підтримки прийняття рішень, які відповідали б сучасним потребам є особливо актуальною. У цій роботі було розглянуто сучасні методи розробки програмного забезпечення та запропоновано архітектуру для реалізації систем підтримки прийняття рішень, яка може бути адаптована до широкого кола задач.

Було продемонстровано застосування цієї архітектури на прикладі розробки конкретної системи підтримки прийняття рішень, розробленої для діагностики станів пацієнтів. Проведені дослідження не тільки підкреслюють значимість нечіткої логіки в сфері медицини, але й відкривають нові горизонти для подальшого вивчення та розуміння процесів діагностики.

Впровадження розробленої системи відкриває можливість удосконалення сфери обслуговування пацієнтів, зокрема, шляхом зниження рівня залежності від професійної компетентності медичних працівників, а отже, підвищує стандарти медичної допомоги, особливо в місцях, де медичний персонал може мати обмежені ресурси або недостатню підготовку. Ще однією важливою перевагою цієї системи є можливість усунення фактору людської помилки у процесі діагностики. Впровадження автоматизованої системи також дозволить прискорить процес встановлення діагнозу.

Загальними функціональними можливостями запропонованої системи є процедура діагностики потенційних захворювань, яка починається з первинної ідентифікації пацієнта та запису його симптомів. Кінцевим продуктом процедури діагностики є комплексний список можливих захворювань, пов’язаних зі вказаними симптомами. Окрім цього, одним із ключових аспектів розробленої системи є

здатність оператора вносити зміни в базу знань без потреби в маніпуляціях з програмним кодом. Це значно спрощує процес впровадження інновацій та змін. Додатково система надає можливість ведення історії візитів пацієнтів, що може сприяти зростанню ефективності та прозорості медичних служб.

Для створення запропонованої системи була використана детально розроблена архітектура, що складається з п’яти основних рівнів. Кожний з цих рівнів включає в себе набір абстрактних модулів, роль та призначення яких було відповідно описано.

Для практичної імплементації даної системи кожний із цих модулів було спроектовано та розроблено за допомогою вибраного набору технологій та методологій. Цей набір включає такі технології та методи: Hetzner Cloud, що слугує основою для хмарної інфраструктури; Kubernetes, що використовується для автоматизації розгортання, масштабування та управління контейнерізованими додатками; CloudFlare, що надає засоби для підвищення безпеки та прискорення завантаження вебсайтів.

На рівні бази даних було використано PostgreSQL з реплікацією для зчитування, що гарантує високу доступність та забезпечує високу продуктивність. Для взаємодії з базою даних було використано об’єктно-реляційне відображення, що дозволяє легко перетворювати дані між базою даних та програмним кодом.

У розробці фронтенду використовувався SvelteKit, інноваційний фреймворк для створення швидких та гнучких вебдодатків. Для розробки бекенду використовувався потужний маршрутизатор API Spring Cloud Gateway та мови програмування Java та Python для реалізації складних бізнес-логік.

Щодо моніторингу та аналітики, було використано ELK Stack для збору та аналізу логів, Prometheus для моніторингу системи та Grafana для візуалізації даних, які спільно створюють повноцінну систему моніторингу та відстеження стану системи.

Додатково було розглянуто виклики, пов’язані з моделюванням даних у великомасштабних інформаційних системах, зокрема з урахуванням специфіки предметної області. Було акцентовано увагу на тому, що кількість та типи сутностей предметної області можуть значно відрізнятися залежно від специфіки задачі.

Також проведено аналіз використання неперервної інтеграції та її ключових компонентів – системи контролю версій, середовищ розгортання та конвеєрів у процесі життєвого циклу сучасних систем прийняття рішень.

Розвиток та удосконалення систем підтримки прийняття рішень в різних областях продовжується. Значна кількість суб’єктів господарювання все більше залежать від обробки великих об’ємів інформації та від аналітики даних. Це означає, що системи підтримки прийняття рішень є важливими компонентами успішної бізнес діяльності, а отже, їх подальше вивчення є актуальною задачею сьогодення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Filip F.G. Decision Support Systems / Filip F.G., Zamfirescu, CB., Ciurea, C. // Springer, 2017. – P. 31–69.
2. Zadeh L.A. Fuzzy Sets / Zadeh L.A. // Information and Control, Vol.8, 1965.
3. Cohn M. User stories applied: For agile software development / Cohn M. // Addison-Wesley Professional, 2004.
4. Ford F. N. Decision support systems and expert systems: A comparison / Ford F. N. // Information & Management, Volume 8, Issue 1, 1985. – P. 21-26.
5. Klir G. J. Fuzzy Sets and Fuzzy Logic: Theory and Applications / Klir G. J., Yuan B. // Prentice-Hall, 1995.
6. Коцовський В. М. Методи та системи штучного інтелекту / Коцовський В. М. // Коцовський В. М., Ужгород, 2016. – С. 17.
7. Yoon V. Y., Expert Systems Construction / Yoon V. Y., Adya M. // Encyclopedia of Information Systems, Elsevier, 2003, – P. 291-300.
8. Попов Э.В. Экспертные системы / Попов Э.В. // Наука, 1987. – С. 212.
9. Bruce G. B. Rule-based Expert System – The MYCIN Experiments of the Stanford Heuristic Programming Project / Bruce G. B., Edward H. Shortliffe // Addison-Wesley, 1984.
10. Winkel D. J Evaluation of an AI-Based Detection Software for Acute Findings in Abdominal Computed Tomography Scans: Toward an Automated Work List Prioritization of Routine CT Examinations / Winkel D. J. , Heye T., Weikert T. J., Boll D. T., Stieltjes B. // Investigative Radiology 54 (1), 2019. – P. 55-59.
11. NextGen Healthcare EHR Overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.g2.com/products/nextgen-healthcare-ehr/reviews#reviews.
12. CloudMedx [Електронний ресурс] – Режим доступу до ресурсу: https://www.featuredcustomers.com/vendor/cloudmedx/case-studies
13. Глибовець М.М. Штучний інтелект / Глибовець М.М., Олецький О.В. // Видавничий дім "КМ Академія", 2002.

14. Провотар О.І. Використання категорій як абстрактних структур для формалізації моделей обчислень / Провотар О.І., Ількун О.П. // Зв'язок. – 2022. – № 5. С.46-49.
15. Provotar A. I. Fuzzy inference systems and their applications / Provotar A. I., Lapko A. V., Provotar A. A. // Cybernetics and Systems Analysis, Vol. 49, 2013. – P. 517-525.
16. Саввакін В.Д. Діагностична система на основі нечітких знань / Саввакін В.Д., Провотар О.І. // Компьютерная математика, №1 2019. – С. 56-63.
17. A. Matiwos A fuzzy model for chaotic time series prediction / A. Matiwos // International Journal of Innovative Computing, Information and Control 14, 2018. – P. 1767-1786.
18. Mamdani, E. H. Application of fuzzy algorithms for the control of a simple dynamic plant / Mamdani, E. H. // In Proc IEEE, 1974. – P. 121–159.
19. Gupta M.M. Theory of T-norms and fuzzy inference methods / Gupta M.M., Qi J. // Fuzzy Sets and Systems, Volume 40, Issue 3, 1991. – P. 431-450.
20. Keogh E. Curse of Dimensionality / Keogh E., Mueen A. // Encyclopedia of Machine Learning and Data Mining, Springer, Boston, 2017. – P. 314–315.
21. Boger C.M. A Synoptic Key of the Materia Medica / Boger C.M. // B Jain Publishers, 2002.
22. Провотар О.І. Про один підхід до знаходження оцінок достовірності при нечіткому моделюванні / Провотар О.І., Ількун О.П. // Кібернетика та Системний Аналіз. – 2023. – Том 59, № 6. С.30-39.
23. Rutkowska, D. Sieci Neuronowe, Algorytmy Genetyczne i Systemy Rozmyte // Rutkowska, D., Pili´nski, M., Rutkowski, L. // Wydawnictwo Naukove PWN, Warszava 1997.
24. Provotar O.I. Credibility in fuzzy inference systems / Provotar O.I., Provotar O.O. // Cybernetics and Systems Analysis, Vol. 53, №6., 2017. – P. 866–876.
25. Buckley J.J. Fuzzy probabilities: New approach and applications / Buckley J.J. // Studies in Fuzziness and Soft Computing, Vol. 115, Berlin, 2005. – P. 165.
26. Vejnarova J. Conditional independence relations in possibility theory / Vejnarova J. // Int. J. Uncertainty, Fuzziness and Knowledge-Based Systems, Vol. 08, №3, 2000. – P. 253–269.

27. Tie J. Study on application model of three-tiered architecture / J. Tie, J. Jin, X. Wang // 2011 Second International Conference on Mechanic Automation and Control Engineering, Inner Mongolia, China, 2011. – P. 7715-7718.
28. Ількун О.П. Вебзастосунки як один із сучасних способів реалізації систем підтримки прийняття рішень // Вісник Київського національного університету імені Тараса Шевченка. Серія: фізико-математичні науки. – 2023. – Випуск №1. С.94-100.
29. Hetzner Cloud Docs [Електронний ресурс] – Режим доступу до ресурсу: https://docs.hetzner.com/cloud/.
30. Jeffery A. Rearchitecting kubernetes for the edge / Jeffery A. Howard H. Mortier R. // In Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking, 2021. – P. 7-12.
31. Medel V. Modelling performance & resource management in kubernetes / Medel V. Rana O. Bañares J. Á. Arronategui U. // In Proceedings of the 9th International Conference on Utility and Cloud Computing, 2016. – P. 257-262.
32. Kubernetes Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://kubernetes.io/docs/home/.
33. Dewi Estri J. H. Implementation of cloudflare hosting for speeds and protection on the website / Dewi Estri J. H. Umar R. Riadi I. // Universitas Ahmad Dahlan, 2019.
34. Cloudflare Docs [Електронний ресурс] – Режим доступу до ресурсу: https://developers.cloudflare.com.
35. Smith G. PostgreSQL 9.0: High Performance / Smith G. // Packt Publishing Ltd, 2010.
36. Douglas K. PostgreSQL: a comprehensive guide to building, programming, and administering PostgresSQL databases / Douglas K. Douglas S. // SAMS publishing, 2003.
37. PostgreSQL Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/.
38. Wernersson D. Choosing a Rendering Framework: A Comparative Evaluation of Modern JavaScript Rendering Frameworks / Wernersson D. Sjölund, V. // Linnaeus

University, Faculty of Technology, Department of computer science and media technology (CM), 2023.

39. Docs SvelteKit [Електронний ресурс] – Режим доступу до ресурсу: https://kit.svelte.dev/docs/introduction.
40. Xiong Q. Design and implementation of microservices gateway based on spring cloud zuul / Xiong Q. Li W. // In CIBDA 2022, 3rd International Conference on Computer Information and Big Data Applications, 2022. – P. 1-5.
41. Spring Cloud Gateway reference [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-cloud-gateway/docs/current/reference/html/.
42. McLaughlin B. Building Java Enterprise Applications: Architecture (Vol. 1) / McLaughlin B. // O'Reilly Media Inc., 2002.
43. Reimer M. L. Building RESTful Web Services with Java EE 8: Create Modern RESTful Web Services with the Java EE 8 API / Reimer M. L. // Packt Publishing Ltd., 2018.
44. Java Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/en/java/.
45. Millman K. J. Python for scientists and engineers / Millman K. J. Aivazis M. // Computing in science & engineering, 13(2), 2011. – P. 9-12.
46. Raschka S. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence / Raschka S. Patterson J. Nolet C. // Information, 11(4), 2020. – P. 193.
47. Python 3.10 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/.
48. Lahmadi A. Powering monitoring analytics with elk stack / Lahmadi A. Beck F. // 9th international conference on autonomous infrastructure, management and security, 2015.
49. Son S. J. Performance of ELK stack and commercial system in security log analysis / Son S. J. Kwon Y. // 2017 IEEE 13th Malaysia international conference on communications (MICC), 2017. – P. 187-190.
50. ELK Stack documentation [Електронний ресурс] – Режим доступу до ресурсу: https://elastic-stack.readthedocs.io/en/latest/introduction.html.

51. Sukhija N. Towards a framework for monitoring and analyzing high performance computing environments using kubernetes and prometheus / Sukhija N. Bautista E. // 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation, 2019. – P. 257-262.
52. An S. Y. A pre-study on the open source prometheus monitoring system / An S. Y., Cha Y. S., Jeon E. J., Gwon G. Y., Shin B. C., Cha B. R. // Smart Media Journal, 10(2), 2021. – P. 110-118.
53. Prometheus Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://prometheus.io/docs/introduction/overview/.
54. Chakraborty M. Grafana. Monitoring Cloud-Native Applications: Lead Agile Operations Confidently Using Open Source Software / Chakraborty M. Kundan A. P. // Apress, 2021. – P. 187-240.
55. Betke E. Real-time I/O-monitoring of HPC applications with SIOX, elasticsearch, Grafana and FUSE / Betke E. Kunkel J. // High Performance Computing: ISC High Performance 2017 International Workshops, DRBSD, ExaComm, HCPM, HPC-IODC, IWOPH, IXPUG, VHPC, Visualization at Scale, WOPSSS, Springer International Publishing, 2017. – P. 174-186.
56. Grafana Labs Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://grafana.com/docs/.
57. Fowler M. "Continuous Integration" [Електронний ресурс] – Режим доступу до ресурсу: https://martinfowler.com/articles/continuousIntegration.html.
58. Booch G. Object Oriented Design with Applications / Booch G., Maksimchuk R. A., Engle M. W., Young B. J., Conallen J., Houston K. A. // Benjamin Cummings, 1991.
59. Duvall P. M. Continuous Integration: Improving Software Quality and Reducing Risk / Duvall P. M., Matyas S., Glover A. // Addison-Wesley, 2007.
60. Cunningham H. G. "Integration Hell" [Електронний ресурс] – Режим доступу до ресурсу: http://wiki.c2.com/?IntegrationHell
61. Humble J. Continuous delivery: reliable software releases through build, test, and deployment automation / Humble J., Farley D. // Pearson Education, 2010.

62. Rodríguez P. Continuous deployment of software intensive products and services: A systematic mapping study / Rodríguez P., Haghighatkhah A., Lwakatare L.E., Teppola S., Suomalainen T., Eskeli J., Karvonen T., Kuvaja P., Verner J.M. , Oivo M. // Journal of systems and software, 123, 2017. – P. 263-291.
63. Zolkifli N. N. Version Control System: A Review / Zolkifli N. N., Ngah A., Deraman A. // Procedia Computer Science Volume 135, 2018. – P. 408-415.
64. Apache Subversion Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://subversion.apache.org/docs/.
65. Perforce Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.perforce.com/support/self-service-resources/documentation.
66. Loeliger J. Version Control with Git: Powerful tools and techniques for collaborative software development. / Loeliger J. , McCullough M. // O'Reilly Media, Inc., 2012.
67. O'Sullivan B. Mercurial: The Definitive Guide: The Definitive Guide. / O'Sullivan B. // O'Reilly Media, Inc., 2009.
68. Bazaar Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://documentation.help/Bazaar-help/introducing_bazaar.html.
69. GitHub flow [Електронний ресурс] – Режим доступу до ресурсу: https://docs.github.com/en/get-started/quickstart/github-flow.
70. Driessen V. A successful Git branching model [Електронний ресурс] – Режим доступу до ресурсу: https://nvie.com/posts/a-successful-git-branching-model/.
71. Murray P. Traditional development/integration/staging/production practice for software development / Murray P. // Disruptive Library Technology Jester, 2006.
72. Donca I.C. Method for Continuous Integration and Deployment Using a Pipeline Generator for Agile Software Projects / Donca I.C., Stan O.P., Misaros M., Gota D., Miclea L. // Sensors, 22, 4637, 2022.
73. Smart J.F. Jenkins: The Definitive Guide: Continuous Integration for the Masses / Smart J.F. // O'Reilly Media, Inc., 2011.
74. Arefeen M.S. Continuous Integration Using Gitlab / Arefeen M.S., Schiller M. // Undergraduate Research in Natural and Clinical Science and Technology Journal, 3, 2019. – P. 1-6.

75. Kosar T. A preliminary study on various implementation approaches of domain-specific language / Kosar T., Martı P.E., Barrientos P.A., Mernik, M. // Information and software technology, 50(5), 2008. – P. 390-405.

ДОДАТКИ

Додаток А

Список праць здобувача за темою дисертації

Статті у наукових фахових виданнях України:

1. Провотар О.І., Ількун О.П. Використання категорій як абстрактних структур для формалізації моделей обчислень // Зв'язок. – 2022. – № 5. С.46-49.
2. Ількун О.П. Вебзастосунки як один із сучасних способів реалізації систем підтримки прийняття рішень // Вісник Київського національного університету імені Тараса Шевченка. Серія: фізико-математичні науки. – 2023. – Випуск №1. С.94-100.
3. Провотар О.І., Ількун О.П. Про один підхід до знаходження оцінок достовірності при нечіткому моделюванні // Кібернетика та Системний Аналіз. – 2023. – Том 59, № 6. С.30-39.

Праці, які засвідчують апробацію матеріалів дисертації:

4. Провотар О.І., Ількун О.П., Провотар Т.М. Достовірність нечітких рішень в задачах розпізнавання // Матеріали міжнародного наукового симпозіуму «Інтелектуальні рішення». м. Ужгород, Україна, 2021. С. 66.
5. Ількун О.П. Оптимізація коефіцієнта конверсії вебсистеми через пошук найвпливовішого за швидкодією компонента // Матеріали XXI Міжнародної науково-практичної конференції «Шевченківська весна – 2023», Секція «Прикладна математика, комп'ютерні науки, інженерія програмного забезпечення, системний аналіз». м. Київ, Україна, 2023. С.82-83.
6. Провотар О.І., Ількун О.П. Категорії як формальні моделі обчислень // Міжнародний науковий симпозіум «Питання оптимізації обчислень (ПОО-XLVIII)», Фізико-математичне моделювання та інформаційні технології (37). м. Львів, Україна, 2023. С.98-102.