УДК 004.91; 004.8 DOI: 10.30837/0135-1710.2024.182.073

М.В. ЄВЛАНОВ, Б.І МОРОЗ, Д.М. МОРОЗ, В.В. ЛУЧИЦЬКИЙ

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ВИЯВЛЕННЯ ТЕРМІНІВ ТА АРТЕФАКТІВ ПРОЄКТУ У ВИМОГАХ ДО ІНФОРМАЦІЙНОЇ СИСТЕМИ

Проаналізовано теоретичні і прикладні можливості застосування методів обробки природної мови для автоматизації елементів процесу «Визначення вимог до системи». Встановлено формальну основу побудови інформаційної технології виявлення термінів та артефактів проєкту у вимогах до інформаційної системи. Розроблено опис архітектури і технологічний стек, розглянуто особливості реалізації основних елементів цієї інформаційної технології. Проведено успішну експериментальну перевірку отриманих результатів.

1. Вступ

Управління ІТ-проєктами в сучасному динамічному технологічному середовищі є складним завданням, яке вимагає системного та структурованого підходу для досягнення успішних результатів. Для ефективного керування такими проєктами важливо мати чітке розуміння процесів, що складають життєві цикли проєкту та ІТ-продукту, який розглядається як результат цього проєкту. Одним із цих процесів є процес «Визначення вимог до системи», який відіграє ключову роль у розробці успішних ІТ-продуктів. Цей процес спрямований на систематичний збір та аналіз вимог до системи з метою точного визначення функціональних і нефункціональних характеристик, які система повинна втілювати [1]. Під системою тут і надалі будемо розуміти ІТ-продукти, описи яких повинні базуватися на системному підході.

Центральну роль у процесі «Визначення вимог до системи» відіграє діяльність «Визначення системних вимог». Під час цієї діяльності проводиться аналіз потреб користувачів та зацікавлених сторін щодо функціональності, продуктивності, безпеки та інших аспектів системи. Результатом цієї діяльності є визначення та обґрунтування системних вимог, яке охоплює визначення вимог до системи відповідно до вимог зацікавлених сторін, функціональних меж, функцій, обмежень, цільових показників вартості, визначених інтерфейсів і критичних характеристик якості [1].

Але виконання діяльності «Визначення системних вимог» ускладнюється тим, що вимоги зацікавлених сторін, які є початковою інформацією для цієї діяльності, є такими, що постійно змінюються під час планування та виконання ІТ-проєкту. Потрібно відзначити, що практика виконання ІТ-проєктів підтверджує, що внесення змін до ІТ-продуктів на ранніх етапах їхнього життєвого циклу є найефективнішим та економічно найдоцільнішим підходом. Це означає, що вирішення проблем та удосконалення системи на початковому етапі її розвитку вимагає менших витрат часу, грошей та ресурсів порівняно з такими ж діями на пізніших етапах проєкту [2]. Такий підхід свідчить про важливість ретельного аналізу визначення вимог до системи і важливість врахування потреб користувачів, технічних обмежень, а також вимог щодо забезпечення безпеки, надійності та інших якостей системи.

Для успішного вирішення проблеми внесення змін до вимог до системи на ранніх етапах її життєвого циклу необхідне розуміння цих вимог та особливостей описів архітектури і проєкту системи, засноване на попередньому досвіді. Таке розуміння дозволяє заздалегідь передбачити можливі складнощі та проблеми, які можуть виникнути під час реалізації актуальних вимог до системи або внесення змін до цих вимог.

Одним із способів використання попереднього досвіду у вирішенні цієї проблеми є

аналіз архітектурних особливостей існуючої системи, її архітектурних сутностей та їхніх зв'язків з іншими компонентами системи під час визначення вимог до системи. Таким способом можна ідентифікувати області системи, які можуть бути найвразливішими до змін, а також визначити потенційні ризики та виклики, пов'язані з цими змінами. Попередній досвід також дозволяє оцінити вплив запропонованих змін на вже існуючі компоненти системи та їхні функціональні можливості. Це допомагає уникнути неочікуваних негативних наслідків та забезпечити сумісність нових функцій з вже існуючими.

Але, хоча попередній досвід є важливим джерелом знань, він може також викликати упередженість, що може негативно впливати на процес прийняття рішень щодо створюваної або вдосконалюваної системи. У [3] показано, що більшість випадків негативного впливу локальних рішень на загальний дизайн і якість великої програмно-технічної системи усувалося шляхом раннього поділу системи на окремі елементи. Однак зворотні випадки виникали, головним чином, внаслідок неправильного тлумачення вимог або упередженості особистого досвіду [3]. Це дозволяє зробити висновок про доцільність використання для ідентифікації конфігурації великих систем та, зокрема, для визначення вимог до системи на основі вимог зацікавлених сторін людино-машинних або машинних методів, у яких суб'єктивний вплив окремого аналітика зведено до мінімуму. Такий підхід допомагає уникнути упередженості особистого досвіду та забезпечити об'єктивний та систематичний аналіз складнощів та потенційних проблем в процесі розробки та управління великими системами.

2. Аналіз сучасного стану теоретичних та прикладних робіт з аналізу системних вимог

2.1. Аналіз особливостей застосування методів штучного інтелекту в сучасних інструментальних засобах аналізу та управління вимогами

Одним з найперспективніших напрямів досліджень у галузі створення людино-машинних або машинних методів інженерії вимог є дослідження методів обробки природної мови (Natural Language Processing, NLP). Такі методи використовуються для аналізу, розуміння та генерації природної мови комп'ютерними системами, що може допомогти у виявленні та аналізі вимог користувачів, коментарів, відгуків та інших текстових джерел, які стосуються розробки та управління великими системами. Зокрема, ці методи можуть використовуватися для автоматизації процесу аналізу тексту з метою виявлення ключових сутностей, патернів або проблем, які можуть впливати на розвиток системи. Крім того, методи NLP можуть допомогти у створенні об'єктивних критеріїв та метрик для оцінки якості та ефективності системи, що дозволяє уникнути суб'єктивності та упередженості в процесі оцінки системних вимог та описів архітектури і дизайну системи. Як зазначається у [4], в середньому 60 % помилок у IT-проєктах спричинені виникненням помилок на етапах визначення та аналізу вимог через неоднозначності або плутанину в критеріях прийняття цих вимог.

Як інструментальні засоби, що використовуються для аналізу та управління вимогами та засновані на методах NLP, у [4] особливо відзначають такі IT-продукти та інформаційні технології (IT):

а) Visual Narrator;

б) QuARS (Quality Analyzer for Requirements Specification);

в) Qualicen Requirements Scout (QRS)

г) Semantha;

д) ReqSuite;

е) IBM Engineering Requirements Quality Assistant;

ж) генеративний штучний інтелект (ШІ), такий як GPT-4 від OpenAI.

Visual Narrator є автоматизованим рішенням, що виводить концептуальну модель з вимог користувачів, використовуючи такі методи як POS-tagging для ідентифікації лінгвістичних патернів речень. Результатом є «доменна онтологія», яка формує модель, кількісно описуючи об'єкти та відносини між ними таким чином, що програмне забезпечення може їх інтерпретувати [5].

QuARS є методологією, призначеною для оцінки якості специфікацій вимог, та інструментом на основі цієї методології. Основна мета QuARS полягає у виявленні аспектів, де вимоги можуть бути неясними, непослідовними або неповними, що сприяє покращенню загальної якості вимог та зниженню ризику непорозумінь або помилок у процесі розробки. Серед особливостей QuARS можна виділити контрольні списки або керівництва для оцінки якості вимог, інструменти для автоматизованого аналізу документів вимог з метою ідентифікації потенційних проблем, а також панелі управління або звіти, які надають уявлення про якість вимог і виокремлюють області для покращення [6].

QRS є програмним інструментом, який надає рішення для інженерії та управління вимогами від моменту їх створення до остаточного впровадження системи. Він пропонує централізоване сховище для вимог, які можна організувати, відстежувати та контролювати протягом виконання IT-проєкту, забезпечуючи можливість відстеження вимог від їх початкового створення до остаточного впровадження. Таке відстеження гарантує, що вимогами належним чином управляють та що будь-які зміни контролюють та відстежують. Крім того, QRS включає функції аналізу вимог, такі як аналіз впливу, аналіз простежуваності та аналіз якості, які допомагають організаціям забезпечити повноту, послідовність та високу якість своїх вимог [7].

Інструментальний засіб Semantha призначений для порівняння документів на семантичному рівні. Це означає, що Semantha може розуміти ідеї та значення, що перевищують буквальний рівень окремих слів і фраз, дозволяючи виявляти спільні концепції в різних документах, порівнювати їх та виділяти відмінності. Ця здатність робить інструмент надзвичайно корисним для класифікації вимог, ідентифікації пов'язаних ризиків та документування порівняння на семантичному рівні [8].

Особливістю ReqSuite є функція допомоги, яка дозволяє автоматично виявляти кілька ключових проблем у вимогах: концептуальну неповноту (коли важливі вимоги повністю забуваються зацікавленими сторонами або коли вимоги передбачають додаткові вимоги, які відсутні), неточність вимог (коли не зрозуміло, чи є вимога обов'язковою чи опціональною) та суперечності між вимогами. Цей інструмент допомагає забезпечити повноту, точність та відсутність внутрішніх суперечностей для усієї множини вимог, що значно знижує ризик виникнення помилок у процесі розробки та впровадження проєктів. Використання ReqSuite також може спростити процес управління вимогами, забезпечуючи краще розуміння та узгодженість вимог усіма учасниками проєкту, зменшуючи час та витрати на перегляд і корекцію вимог [9].

IBM ERQA використовує передові можливості обробки природної мови для автоматизованого аналізу вимог. Це дозволяє виявляти потенційні двозначності та генерувати реальні оцінки в реальному часі для оцінки якості вимог. Система оцінювання базується на кількох концепціях, таких як складні вимоги, неточні дієслова, неповнота, відсутність обмежень, відсутні одиниці виміру, негативні твердження, незрозумілі займенники тощо. Інструмент дозволяє швидко ідентифікувати можливі проблеми з якістю та забезпечити чіткість та однозначність вимог перед початком розробки. Використання IBM ERQA може значно знизити ризики, пов'язані з неправильним тлумаченням вимог, та покращити якість кінцевого продукту, забезпечуючи ефективнішу співпрацю між усіма

учасниками проєкту [10].

Генеративний ШІ, такий як GPT-4 від OpenAI, в контексті інженерії вимог може використовуватися для уточнення, ідентифікації потенційних конфліктів або проблем та надання рекомендацій для поліпшення якості вимог, а також генерування тестових випадків і підтримки управління та документації вимог [7].

На основі аналізу цих інструментів можна зробити висновок, що багато з них вирішують завдання аналізу та класифікації вимог, виявлення неоднозначностей та навіть деякою мірою валідації вимог щодо їхньої повноти та консистентності. Проте ці інструменти часто обмежуються рамками текстового аналізу та не здатні прямо аналізувати та інтегрувати виявлені сутності з існуючими системами або візуалізувати зв'язки між сутностями у формі графів. Можливість такої візуалізації може відіграти важливу роль у виявлення системних вимог [11], а її відсутність значно обмежує застосування розглянутих інструментів та ІТ для комплексного розуміння структур даних системи.

2.2. Аналіз особливостей методів обробки природної мови, які застосовуються для розпізнавання та класифікації термінів із текстових даних

Методи NLP охоплюють широкий спектр від базового синтаксичного аналізу до складних методів глибокого навчання, які здатні вловлювати семантичні зв'язки у тексті. Зокрема, методи NLP дозволяють автоматизувати процес розпізнавання та класифікації термінів із текстових даних, що може значно підвищити ефективність аналізу та управління вимогами.

Серед доступних методів класифікації термінів як ключових сутностей і використання їх для подальшого аналізу пропонується приділити особливу увагу методам стемінгу та лематизації. Стемінг є процесом видалення афіксів зі слів, щоб повернутися до кореня або базової форми слова [12]. Цей метод може бути корисним для спрощення текстових даних, але він може призвести до втрати значущої інформації через недоліки у врахуванні словоформ. Лематизація, у свою чергу, є складнішим процесом, що залучає морфологічний аналіз для перетворення слова на його базову форму або лему. Це процес групування різних словоформ одного слова, так щоб їх можна було аналізувати як єдиний елемент, який можна ідентифікувати за лемою слова або його словниковою формою [12].

Результати SWOT-аналізу методу стемінгу наведено у табл. 1.

Таблиця 1

Результати SWOT-аналізу методу стемінгу

	Корисні фактори	Шкідливі фактори
Внутрішні фактори	Сильні сторони	Слабкі сторони
	Швидкість обробки	Неточність Мовна залежність
Зовнішні фактори	Можливості	Загрози
	Розвиток алгоритмів Інтеграція з іншими методами NLP	Зміни в обробці природної мови Зростання вимог до якості обробки тексту Обмеження в обробці неструктурованих даних

Результати SWOT-аналізу методу лематизації наведено у табл. 2.

Таблиця 2

Результати SWOT-аналізу методу лематизації

	Корисні фактори	Шкідливі фактори
Внутрішні фактори	Сильні сторони	Слабкі сторони
	Точність Контекстна обробка Універсальність Підтримка алгоритмами різноманітних мов	Ресурсоємність Складність реалізації Залежність від мови
Зовнішні фактори	Можливості	Загрози
	Інтеграція з іншими NLP-інструментами; Покращення машинного навчання; Розширення застосування.	Зміни в обробці природної мови Обмеження в обробці неструктурованих даних

На основі проведеного SWOT-аналізу як метод виявлення сутностей у тексті пропонується обрати метод лематизації через його здатність забезпечувати високу точність аналізу. Цей метод враховує контекст слова, що дозволяє точно ідентифікувати базові форми слів, ця характеристика робить лематизацію особливо цінною для задач, де важлива точність розпізнавання сутностей. Хоча лематизація може вимагати більше обчислювальних ресурсів порівняно зі стемінгом та висувати вищі вимоги до реалізації, багато сучасних алгоритмів підтримують українську мову, що свідчить про її застосовність у широкому спектрі лінгвістичних середовищ.

2.3. Формулювання проблеми дослідження

На основі проведеного аналізу особливостей застосування методів штучного інтелекту в сучасних інструментальних засобах аналізу та управління вимогами та особливостей методів обробки природної мови, які застосовуються для розпізнавання та класифікації термінів із текстових даних, можна зробити такі висновки.

По-перше, існує велика потреба в інтеграції досвіду з попередніх проєктів та використання передових практик при визначенні системних вимог. Недостатня увага до досвіду минулих проєктів та відсутність систематичного підходу до збору та аналізу цього досвіду обмежує можливість врахування вже існуючих рішень та помилок. В сучасному контексті розробки та модифікації систем суб'єктивний вплив, заснований на попередньому досвіді, може ініціювати упередженість, що, у свою чергу, негативно позначається на процесі ухвалення рішень та прогресу систем.

По-друге, існуючі методи та підходи до визначення системних вимог часто стикаються з проблемою неоднозначності та невизначеності. Це призводить до труднощів у визначенні вимог до системи, що може спричинити помилки на ранніх етапах проєктування та збільшити витрати на внесення змін у майбутньому.

По-третє, на сьогодні існує дефіцит повноцінних теоретичних та практичних рішень, призначених для автоматизованої обробки вимог до систем. Особливо це стосується забезпечення глибокої інтеграції виявлених сутностей з існуючими системами та візуалізації їх для детального аналізу. Така ситуація ускладнює розробку та адаптацію ІТ-продуктів та, зокрема, інформаційних систем (ІС) управління підприємствами та організаціями, підвищуючи ризик неврахування важливих аспектів на ранніх етапах проєктування.

Дані висновки дозволяють припустити, що забезпечення успішного виконання процесу визначення системних вимог критично потребує розвитку і впровадження автоматизованих методів аналізу текстових даних для виявлення та розуміння системних вимог, особливо в контексті управління ІТ-проєктами. Цей процес також має включати

ефективне використання уроків, винесених з минулого досвіду, і водночас застосування сучасних аналітичних інструментів для уникнення упередженості.

Виділені проблеми є наслідком недостатніх теоретичних та практичних досліджень у сфері формування та аналізу вимог до ІС. Основні публікації зарубіжних дослідників [13], [14] переважно спрямовані на осмислення практичного досвіду у цій області. Проте в цих та інших роботах майже не розглянуто можливості використання автоматизованих інструментів для визначення та аналізу системних вимог.

Тому проблему даного дослідження пропонується визначити як проблему створення спеціалізованих ІТ визначення, аналізу та управління системними вимогами, які застосовують методи NLP для виявлення окремих термінів предметної області та створюваної ІС і аналізу можливості опису цих термінів через артефакти попередніх ІТ-проєктів.

3. Мета і задачі дослідження

Метою даного дослідження є розробка ІТ виявлення термінів та артефактів проєкту ІС. Досягнення цієї мети дозволить автоматизувати діяльність «Визначення системних вимог» і, зокрема, дозволить визначати можливість повторного використання артефактів системи в результаті встановлення їхньої відповідності виявленим під час визначення системних вимог термінам предметної галузі та створюваної системи.

Для досягнення цієї мети треба вирішити такі задачі:

– розробити опис архітектури та визначити технологічний стек для розробки ІТ виявлення термінів та артефактів проєкту ІС;

– розробити елементи ІТ для виявлення та класифікації термінів та артефактів проєкту ІС в текстових документах, з акцентом на групування відмінюваних форм;

– провести експериментальну перевірку отриманих результатів.

4. Матеріали і методи дослідження

Об’єктом дослідження є діяльність «Визначення системних вимог» процесу «Визначення вимог до системи» [1]. Предметом дослідження є ІТ виявлення термінів та артефактів проєкту ІС.

Як основну гіпотезу даного дослідження запропоновано розглядати можливість формувати та зберігати у спеціалізованому репозиторії описи окремих елементів ІС як сукупностей структур даних із встановленими назвами кожної окремої структури. Застосування цієї гіпотези дозволяє представити задачу зіставлення виявлених у текстах системних вимог термінів предметної області та створюваної системи з існуючими артефактами ІС як задачу порівняння формальних представлень цих сутностей з назвами існуючих артефактів. Під артефактом в даному дослідженні будемо розуміти [15]:

а) опис окремого елемента ІС на різних етапах його життєвого циклу;

б) опис ІС в цілому як елементу більшої системи.

Для утворення формального опису термінів предметної області та створюваної системи, які присутні у текстових публікаціях вимог до системи, пропонується задіяти методи лематизації. Базою цих методів є [12]:

– статистичні моделі;

– правила на основі морфології;

– комбінація статистики та правил.

Статистична модель у контексті NLP є моделлю машинного навчання, яка навчається на великих корпусах тексту з метою вивчення зв'язків між словами та їхніми лемами. Ці моделі аналізують частоту вживання слів у різних контекстах та інші мовні ознаки, щоб визначити ймовірність того, що дане слово в певному контексті буде мати певну лему. Позначимо множину усіх слів у тексті як S, множину контекстів як C, множину усіх

можливих лем у мові як L. Тоді статистичні моделі визначають умовну ймовірність *P* того, що дане слово в певному контексті буде мати певну лему, як

$$P = f(l \mid w,c), \tag{1}$$

де $l \in L$ – лема слова $w \in S$ у контексті $c \in C$.

Спочатку статистична модель аналізує контекст кожного слова у текстовій публікації системної вимоги, порівнюючи цей контекст із статистичними зв'язками між словами та їхніми лемами. На основі цього аналізу модель визначає умовну ймовірність для кожної можливої базової форми слова у даному контексті. За допомогою цих умовних ймовірностей модель вибирає найімовірнішу базову форму для кожного слова у текстовій публікації системної вимоги.

Іншим способом визначення умовної ймовірності того, що дане слово в певному контексті буде мати певну лему, є застосування правил на основі морфології. Цей спосіб використовує правила лематизації, які базуються на морфологічних особливостях мови. Для цього способу вираз (1) матиме вигляд:

$$P = f(w,c,\{r_i\}), \tag{2}$$

де $\{r_i\}$ – множина морфологічних правил, які застосовуються для визначення леми (якщо $\{r_i\} = \varnothing$, то правила ігноруються).

Для розроблюваної ІТ виявлення термінів та артефактів проєкту ІС найкращим варіантом лематизації запропоновано визнати комбінацію статистики та правил. Ця комбінація поєднує в процесі вибору леми *l* для слова *w* у контексті *c* результати статистичного моделювання та застосування правил на основі морфології. Це означає, що умовна ймовірність вибору леми розраховується на основі статистичних даних. Правила на основі морфології застосовуються для вибору леми в тому випадку, якщо ці леми не враховуються статистичною моделлю, що використовується. Така комбінація формально може бути представлена як функція мети

$$l^* = arg\,max_{l \in L}\, f(l \mid w,c,\{r_i\}). \tag{3}$$

Таким чином, тепер можливо отримати у вигляді лем формальні представлення термінів, присутніх у текстових публікаціях системних вимог, для подальшого формального співставлення їх із термінами, присутніми в описах артефактів ІС.

Аналогічно можливо отримати у вигляді лем формальні представлення термінів, присутніх в описах артефактів ІС. Відмінність полягає в тому, що функцію (3) в цьому випадку було запропоновано застосовувати для аналізу описів структур даних, використаних в описах досліджуваних артефактів ІС.

Тоді задачу виявлення термінів та артефактів проєкту ІС пропонується розглядати як задачу пошуку такого відображення множини лем термінів текстової публікації системної вимоги $L_R^* = (l_j^*(R_i))$ у множину лем термінів, присутніх в описах артефактів ІС, призначених для повторного використання з метою реалізації системної вимоги $L_{Ar}^* = (l_j^*(Ar_x))$, яке в оптимальному випадку може бути зведене до повної еквівалентності, тобто до ситуації

$$(l_j^*(R_i)) = (l_j^*(Ar_x)), \tag{4}$$

де R_i – текстова публікація системної вимоги, *i=1,2,…,n*; *n* – кількість системних вимог,

які обробляються ІТ за один раз; $l_j^*(R_i)$ – *j*-та лема, визначена на текстовій публікації системної вимоги R_i, $j=1,2,\ldots,m$; m – кількість лем, виявлених у текстовій публікації системної вимоги R_i; Ar_x – опис *x*-го артефакта ІС, який пропонується для повторного використання у реалізації системної вимоги R_i, $x=1,2,\ldots,z$; z – кількість артефактів, які пропонуються для повторного використання у реалізації системної вимоги R_i; $l_j^*(Ar_x)$ – *j*-та лема, визначена на описі артефакта ІС Ar_x.

У випадку отримання раціонального рішення задачі виявлення термінів та артефактів проєкту ІС відображення (4) матиме вигляд

$$(l^*_{j=1,\ldots,m}(R_i)) \geq (l^*_{j=1,\ldots,k\leq m}(Ar_x)), \quad (5)$$

де *k* – кількість лем, виявлених у описі артефакта ІС Ar_x.

5. Результати дослідження

5.1. Розробка опису архітектури та визначення технологічного стека інформаційної технології

Розроблювану ІТ виявлення термінів та артефактів проєкту ІС було запропоновано представити як послідовність таких етапів і кроків.

Етап 1. Перетворення функціональних вимог, які висуваються до ІС, до початкової форми.

Крок 1.1 Формування та зберігання текстової публікації функціональної вимоги до ІС.

Крок 1.2. Токенізація текстової публікації функціональної вимоги до ІС.

Крок 1.3. Формування початкової форми текстової публікації функціональної вимоги до ІС шляхом лематизації множини токенів, отриманої в результаті виконання Кроку 1.2.

Етап 2. Перетворення описів артефактів ІС, придатних для повторного використання, до початкової форми.

Крок 2.1. Токенізація опису артефакту ІС, придатного для повторного використання.

Крок 2.2. Формування початкової форми опису артефакту ІС, придатного для повторного використання, шляхом лематизація множини токенів, отриманої в результаті виконання Кроку 2.1.

Крок 2.3. Оновлення множини початкових форм описів артефактів ІС, придатних для повторного використання.

Етап 3. Визначення відповідностей між початковими формами текстових публікацій функціональних вимог та описів артефактів ІС, придатних для повторного використання.

Крок 3.1. Визначення відповідності лем з початкової форми текстової публікації функціональної вимоги до ІС та лем з початкової форми опису артефакту ІС, придатного для повторного використання.

Крок 3.2. Відбір початкових форм описів артефактів ІС, придатних для повторного використання, відповідної функціональної вимоги за результатами виконання Кроку 3.1.

Крок 3.3. Візуалізація описів артефактів ІС, придатних для повторного використання, початкові форми яких було відібрано в результаті виконання Кроку 3.2. Завершення застосування ІТ.

Опис архітектури ІТ виявлення термінів та артефактів проєкту ІС формувався під впливом двох конкуруючих вимог:

а) необхідно інтегрувати всі необхідні компоненти обробки, аналізу та візуалізації даних у єдиний цілісний сервіс, що вимагає застосування монолітної архітектури;

б) з огляду на використання новітніх технологій, опис архітектури ІТ повинен бути

розроблений таким чином, щоб у майбутньому можна було б легко адаптувати його, наприклад, до мікросервісної моделі з метою підвищення масштабованості та ефективності обробки публікацій вимог, що надходять з великої кількості територіально розподілених джерел.

Додатковою вимогою, яку треба було враховувати під час розробки опису архітектури ІТ, є вимога можливості налаштування сервісу, який реалізує ІТ, для обробки публікацій вимог незалежно від мови користувача. У випадку, якщо вимоги опубліковані українською мовою, таке налаштування включатиме підключення до бази даних, де описи артефактів визначені українською мовою, а для обробки вимог завантажується статична модель лематизації, відповідна до мови користувача.

Виходячи з цих та ряду інших вимог, в процесі створення опису архітектури ІТ виявлення термінів та артефактів проєкту ІС було запропоновано виділяти такі архітектурні сутності (компоненти) цієї ІТ:

а) компонент перетворення вимог до початкової форми, який забезпечує виконання токенізації текстової публікації вимоги, розбиваючи її на окремі слова, які, в свою чергу, служать вхідними даними для процесу лематизації;

б) компонент декорування схеми бази даних ІС, який забезпечує виконання токенізації та лематизації назв таблиць і полів, з яких складаються описи артефактів, придатних для повторного використання;

в) компонент створення діаграми зв'язків, який забезпечує встановлення відповідностей між лемами вимог користувача, отриманими з компонента перетворення вимог до початкової форми, та лемами опису артефактів, отриманими з компонента декорування схеми бази даних ІС, а також генерацію представлення встановлених відповідностей у вигляді візуальної діаграми для подальшого аналізу вимоги.

Оскільки основною забезпечуючою системою [1] для ІТ виявлення термінів та артефактів проєкту ІС є програмна система, подальша деталізація опису архітектури системи повинна враховувати результати формування технологічного стека цієї ІТ. Під технологічним стеком тут і далі будемо розуміти набір технологій, які використовуються разом для розробки та підтримки програмного забезпечення [16]. Зазвичай технологічний стек для програмної забезпечуючої системи складається з таких компонентів [16]:

– фронтенд (клієнтська частина): технології, що використовуються для створення інтерфейсу користувача;

– бекенд (серверна частина): технології для роботи з бізнес-логікою, базами даних та серверними процесами;

– база даних: системи для зберігання та управління даними;

– система управління версіями: інструменти для відстеження змін у коді;

– інструменти для розгортання та підтримки: технології для автоматизації розгортання та підтримки проєкту.

Технологічний стек ІТ виявлення термінів та артефактів проєкту ІС має такі складові:

а) для розробки загальних елементів сервісу, що повинен реалізувати ІТ, запропоновано використати мову програмування C# та кросплатформенну технологію .NET Core [17] від компанії Microsoft;

б) для розробки компонента перетворення вимог до початкової форми запропоновано використати:

– мову програмування Python та, зокрема, бібліотеку SpaCy [18], яка є ключовим інструментом надання розширених можливостей для лематизації та морфологічного аналізу, включаючи підтримку багатьох мов, включаючи українську мову;

– офіційну бібліотеку SpacyDotNet [19], яка дозволяє використовувати можливості SpaCy в середовищі .NET, зокрема в .NET Core, і базується на Python.NET для забезпечення

взаємодії між Python та .NET;

в) для розробки компонента декорування схеми бази даних ІС запропоновано використати:

– СУБД MS SQL Server;

– бібліотеку Microsoft.SqlServer.Management.Smo, можливості якої забезпечують вилучення та аналіз метаданих як елементів описів артефактів ІС, необхідних для кореляції із лематизованими токенами, отриманими з текстових публікацій системних вимог;

г) для розробки компонента створення діаграми зв'язків запропоновано використати інструмент PlantUML [20], який призначений для створення діаграм програмного забезпечення і дозволяє описувати структуру програми за допомогою текстового опису та автоматично генерувати візуальні діаграми на основі цього опису, використовуючи простий синтаксис, що базується на UML;

д) для розробки частини сервісу, яка забезпечить взаємодію користувача з цим сервісом та можливість подання текстових публікацій вимог до обробки, запропоновано використати фреймворк React [21], який був обраний за рахунок свого розширеного набору готових до використання стилізованих компонентів і, зокрема, бібліотеки компонентів BlueprintJs [22] для створення інтуїтивного інтерфейсу.

Цей технологічний стек був обраний з огляду на здатність до швидкої розробки, необхідності підтримки різних людських мов, а також легкості інтеграції різних компонентів у єдиний цілісний сервіс.

З врахуванням обраного технологічного стека, опис архітектури сервісу, який реалізує ІТ виявлення термінів та артефактів проєкту ІС, було приведено до діаграми компонентів, наведеної на рис. 1.

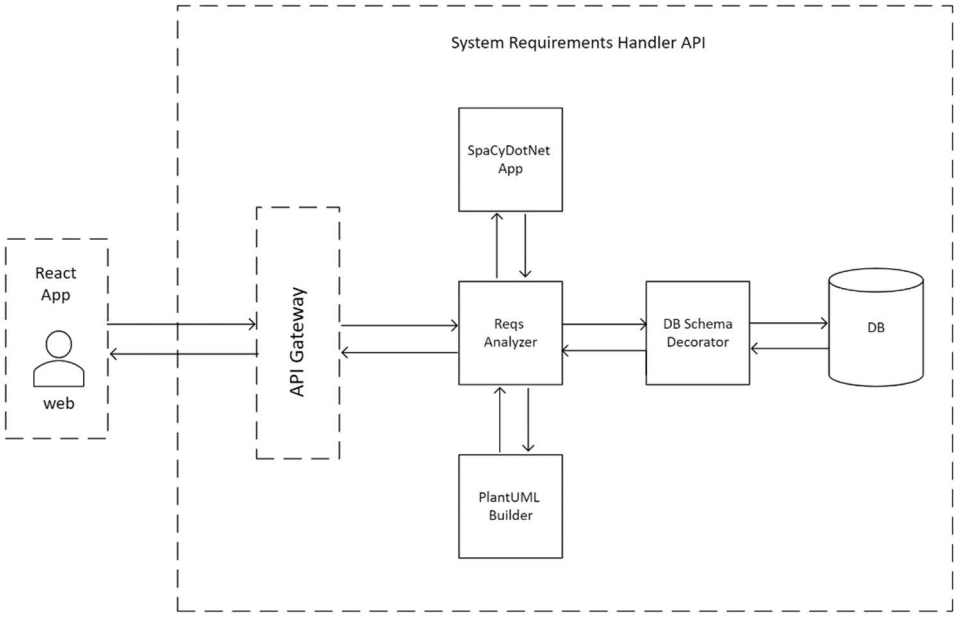

Рис. 1. Діаграма компонентів, яка описує архітектуру сервісу

На рис. 1 прийняті такі позначення:

– Reqs Analyzer – модуль, який забезпечує реалізацію компонента перетворення вимог до початкової форми та елементів компонента створення діаграми зв'язків, які відповідають за встановлення відповідностей між лемами вимог користувача та лемами опису артефактів;

– SpaCyDotNetApp – модуль, який забезпечує інтеграцію бібліотеки SpaCy у модуль Reqs Analyzer із використанням можливостей бібліотеки SpacyDotNet;

– DB Schema Decorator – модуль, який забезпечує реалізацію компонента декорування схеми бази даних ІС;

– DB – сховище даних, яке забезпечує реалізацію репозиторія артефактів ІС, придатних для повторного використання;

– PlantUML Builder – модуль, який забезпечує реалізацію елементів компонента створення діаграми зв'язків, які відповідають за генерацію представлення встановлених відповідностей у вигляді візуальної діаграми для подальшого аналізу вимоги;

– API Gateway – модуль, який реалізує шлюз для інтеграції серверної та клієнтської частин сервісу;

– React App – модуль, який реалізує частину сервісу на основі фреймворку React, яка забезпечує взаємодію користувача з цим сервісом та можливість подання текстових публікацій вимог до обробки.

5.2. Особливості реалізації елементів інформаційної технології автоматизованого виявлення та аналізу сутностей вимог до ІС

Однією з основних вимог до ІТ виявлення термінів та артефактів проєкту ІС є її здатність обробляти вимоги користувачів різними мовами, зокрема англійською та українською. Для задоволення цієї вимоги сервіс, який реалізує зазначену ІТ, було спроєктовано з використанням гнучкої конфігурації API, що дозволяє легко інтегрувати підтримку додаткових мов.

Конфігураційний файл сервісу містить окремі секції для кожної мови, де визначаються параметри для обробки тексту, взаємодії зі сховищем даних, ідентифікації колонок та регулярних виразів для обробки тексту, що дозволяє адаптувати сервіс під конкретні мовні особливості. Моделі для обробки тексту використовують можливості бібліотеки SpaCy, що дозволяє користувачам обрати будь-яку мову, що підтримується цією бібліотекою. Для апробації налаштовано використання української мови за допомогою попередньо навченої моделі uk_core_news_sm та англійської мови за допомогою попередньо навченої моделі en_core_web_sm. Фрагмент програмного коду, який описує конфігурацію мовних моделей, наведено на рис. 2. Такі параметри, як LangModel, ConnectionDb, ColumnIdentifierSuffix та ColumnSplitRegex, забезпечують не тільки адекватне розпізнавання та обробку мовних структур, але й належну взаємодію зі сховищем даних та адаптацію до специфічних мовних ідентифікаторів у даних.

```
"DataSet": {
  "ua": {
    "LangModel": "uk_core_news_sm",
    "ConnectionDb": "Data Source=localhost;Initial Catalog=ФутбольнаЛіга;User ID=nure;
    "ColumnIdentifierSuffix": "Код",
    "ColumnSplitRegex": "[А-ЯЁЇІЄҐ][^А-ЯЁЇІЄҐ]*"
  },
  "en": {
    "LangModel": "en_core_web_sm",
    "ConnectionDb": "Data Source=localhost;Initial Catalog=FootbalLeague;User ID=nure;
    "ColumnIdentifierSuffix": "Id",
    "ColumnSplitRegex": "[A-Z][^A-Z]*"
  }
}
```

Рис. 2. Фрагмент програмного коду, який описує конфігурацію мовних моделей інформаційної технології

В рамках реалізації модулів, зазначених на рис. 1, а саме ReqsAnalyzer для визначення лем з тексту вимог та DB Schema Decorator для визначення і декорації лемами описів артефактів ІС, використовується метод nlp з бібліотеки SpaCy. Цей метод є частиною стандартного інтерфейсу обробки тексту в SpaCy.

C# Lang.cs

```
13      public class Lang : IXmlSerializable
            1 usage  luchytskyi
24          public Doc GetDocument(string text)
25          {
26              using (Py.GIL())
27              {
28                  var pyString = new PyString(text);
29                  var doc :dynamic = PyLang.__call__(pyString);
30                  return new Doc(doc, text);
31              }
32          }
33
```

Рис. 3. Фрагмент програмного коду, який описує клас Lang

Оскільки в технологічному стеку було запропоновано використовувати бібліотеку SpacyDotNet для інтеграції функцій бібліотеки SpaCy у додатки, написані на C#, в сервісі було реалізовано клас Lang і його метод GetDocumnet, який повертає об'єкт Doc (рис. 3). Цей об’єкт містить структуровану інформацію про оброблений текст і його властивості, такі як Text, Lemma, POS, Tag тощо.

Фрагмент програмного коду, який описує клас DbEntitiesLemmatizer, що декорує описи артефактів ІС лемами (зроблені на основі таблиць, їхніх назв і полів), наведений на рис. 4. DecorateEntitiesWithLemma – основний публічний метод цього класу, призначений для обробки колекцій таблиць. Він приймає колекцію об'єктів типу Microsoft.SqlServer.Management.Smo.TableCollection і повертає колекцію об'єктів типу ReqsTable, кожен з яких містить леми для відповідних елементів таблиці.

```
using System.Text.RegularExpressions;
using Microsoft.IdentityModel.Tokens;
using Microsoft.SqlServer.Management.Smo;
using ReqsHandler.Core.Configuration;
using ReqsHandler.Core.Services.Models;

namespace ReqsHandler.Core.Services;

public class DbEntitiesLemmatizer(ISpacyInstance spacyInstance, ICurrentContext context)
    : IDbEntitiesLemmatizer
{
private Regex ColumnSplitRegex => new Regex(context.DataSet.ColumnSplitRegex);
    public IEnumerable<ReqsTable> DecorateEntitiesWithLemma(TableCollection collection)
    {
      var tables = new List<ReqsTable>();
      foreach (Table table in collection)

      {
        var isSplitName = SplitAndLemmatize(table.Name, out var splitList, out var lemmas);
        tables.Add(new ReqsTable
        {
          Name = table.Name,
          Lemmas = lemmas,
          IsSplitName = isSplitName,
          SplitNames = splitList,
          BaseEntity = table,
          Columns = MapColumnLemma(table.Columns)
        });
      }
      return tables;
    }
```

Рис. 4. Фрагмент програмного коду, який описує клас DbEntitiesLemmatizer

```
  private bool SplitAndLemmatize(string name, out IList<string> splitList, out IEnumerable<string> lemmas)
  {
    splitList = Array.Empty<string>();
    lemmas = Array.Empty<string>();
    if (name.IsNullOrEmpty())
    {
      return false;
    }

    var isSplitName = SplitIfNeed(name, out splitList);
    var doc = spacyInstance.GetDocument(string.Join(" ", splitList));
    lemmas = doc.Tokens.Where(t => !t.IsPunct).Select(t => t.Lemma.ToLower());

    return isSplitName;
  }

  private bool SplitIfNeed(string entityName, out IList<string> result)
  {
    var isSplit = false;
    result = Array.Empty<string>();
    var parts = ColumnSplitRegex.Matches(CleanUpName(entityName));
    if (parts.Count > 1)
    {
      isSplit = true;
    }

    result = new List<string>();
    foreach (Match part in parts)
    {
      result.Add(part.Value);
    }
    return isSplit;
  }
  private IEnumerable<ReqsColumn> MapColumnLemma(ColumnCollection tableColumns)
  {
    var columns = new List<ReqsColumn>();
    foreach (Column column in tableColumns)
    {
      var isSplitName = SplitAndLemmatize(column.Name, out var splitList, out var lemmas);
      columns.Add(new ReqsColumn
      {
        Name = column.Name,
        Lemmas = lemmas,
        BaseEntity = column,
        IsSplitName = isSplitName,
        SplitNames = splitList
      });
    }

    return columns;
  }

  private string CleanUpName(string name)
  {
    return name.Replace(context.DataSet.ColumnIdentifierSuffix, "");
  }
}
```

Рис. 4, аркуш 2

Метод SplitAndLemmatize призначений для визначення лем. В цьому методі реалізовано аналіз назв об'єктів описів артефактів ІС із застосуванням регулярних виразів, які використовуються в конфігурації (див. рис. 1) для визначення необхідності розбиття складних назв на окремі слова. За допомогою методу SplitAndLemmatize перш за все перевіряється, чи потрібно розділити назву на менші фрагменти, що значно покращує точність подальшої обробки вимог.

Треба також зазначити, що перед розбиттям або аналізом текст обробляється методом CleanUpName, який за допомогою зазначеної в конфігурації властивості ColumnIdentifierSuffix видаляє суфікс, яким зазвичай помічають в структурованих базах даних поле з ідентифікатором (наприклад, суфікс «Код» для української локалізації та «Id», відповідно, для англійської локалізації артефактів ІС).

Клас PlantUmlBuilder відіграє ключову роль у створенні коду для візуалізації діаграми за допомогою PlantUML. Фрагмент програмного коду з описом основного методу цього класу BuildUml, який створює текстове представлення діаграми, організовуючи таблиці та їхні взаємозв'язки, наведено на рис. 5. Особливість PlantUmlBuilder полягає в його здатності деталізувати кожну сутність за допомогою лем, отриманих з тексту вимог, що дозволяє відобразити можливі структурні і логічні зв'язки між елементами досліджуваної ІС. Такий підхід робить зрозумілішими та кориснішими діаграми, які використовуються для аналізу і планування системних вимог.

```
C# PlantUmlBuilder.cs
8   public class PlantUmlBuilder(List<TableDto> tables)
        1 usage  Viacheslav Luchytskyi +1
28      private string BuildUml(IList<string> inputLemmas, HashSet<string> includedTables)
29      {
30          var diagram = new StringBuilder();
31          diagram.AppendLine("@startuml");
32          diagram.AppendLine(DefinedFunctions);
33          diagram.AppendLine();
34
35          // Definition of tables
36          var result:string = BuildTableEntities(inputLemmas, includedTables);
37          if (result.IsNullOrEmpty())
38          {
39              return string.Empty;
40          }
41
42          diagram.Append(result);
43
44          // Definition of relationships between tables
45          diagram.Append(BuildDependentTables(inputLemmas, includedTables));
46
47          diagram.AppendLine("@enduml");
48          return diagram.ToString();
49      }
```

Рис. 5. Фрагмент програмного коду, який описує метод BuildUml класу PlantUmlBuilder

Як зазначено у підрозд. 5.1, архітектура ІТ передбачає впровадження різних додаткових модулів. Для таких модулів доступ до функцій сервісу, який реалізує розроблену ІТ, здійснюється через REST API, реалізовані на платформі ASP.NET Core, що забезпечує високу продуктивність і масштабованість.

Для оптимізації процесів налагодження та документації кожного модуля було вирішено забезпечити доступ до API та впровадити використання Swagger [23], як це представлено на рис. 6. Використання цього інструментального засобу значно спрощує інтеграцію та перевірку взаємодії між модулями сервісу.

5.3. Опис експериментальної перевірки отриманих результатів

Для проведення експериментальної перевірки отриманих результатів було використано ІС «Футбол». Цю ІС було спроєктовано спеціально для проведення експериментального дослідження, вона містить набір основних функцій, що дозволяють автоматизувати вирішення задач планування та обліку проведення футбольних матчів. Для перевірки можливостей обробки вимог, опублікованих різними мовами, було реалізовано українську та англійську версію цієї ІС. Як описи артефактів цієї ІС у експериментальних

дослідженнях розглядаються описи таблиць бази даних цієї ІС, полів цих таблиць та зв'язків між цими таблицями.

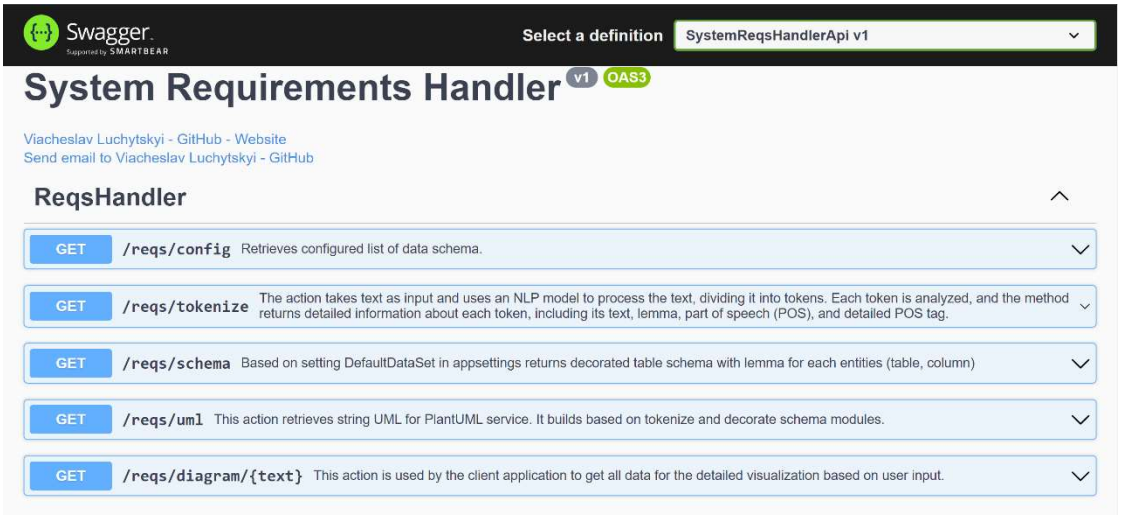

Рис. 6. Інтерфейс для тестування API модулів сервісу

План проведення експериментальної перевірки результатів розробки ІТ виявлення термінів та артефактів проєкту ІС передбачав послідовне виконання таких етапів:

а) етап 1: перевірка працездатності сервісу, який реалізує ІТ, на прикладі системної вимоги, опублікованої англійською мовою;

б) етап 2: перевірка працездатності сервісу, який реалізує ІТ, на прикладі системної вимоги, опублікованої українською мовою;

в) етап 3: перевірка релевантності результатів обробки системної вимоги, опублікованої українською та англійською мовами.

Під час виконання першого етапу експериментальної перевірки було використано системну вимогу, яка висувається до англійської локалізації ІС «Футбол». Текстова публікація цієї вимоги виглядає так: «Give the coach the opportunity to train several teams». Результати токенізації цієї вимоги з використанням можливостей бібліотеки SpaCy наведено на рис. 7. Фрагмент результату декорування лемами описів відповідних артефактів ІС, що були реалізовані у попередніх версіях ІС «Футбол» та можуть бути повторно використані для проєктування та реалізації вказаної вище вимоги, наведено на рис. 8. Результат виділення сукупності артефактів ІС, яка може бути повторно використана для проєктування та реалізації зазначеної вище системної вимоги, наведено на рис. 9. Візуальна діаграма, яку бачить користувач сервісу як результат застосування ІТ, наведена на рис. 10.

Під час виконання другого етапу експериментальної перевірки було використано системну вимогу, яка висувається до української локалізації ІС «Футбол». Текстова публікація цієї вимоги виглядає так: «Додати можливість для гравців виступати за декілька команд». Візуальну діаграму, яку бачить користувач сервісу як результат застосування ІТ для обробки цієї системної вимоги, наведено на рис. 11.

Під час виконання третього етапу експериментальної перевірки текстову публікацію системної вимоги, яка була використана на другому етапі, було перекладено англійською мовою. Результатом перекладу став текст «Add ability for players to participate for several team». Візуальну діаграму, яку бачить користувач сервісу як результат застосування ІТ для обробки цього англійського перекладу системної вимоги, наведено на рис. 12.

Рис. 7. Результати токенізації опублікованої системної вимоги англійською мовою

Рис. 8. Результати декорування лемами описів артефактів англійської локалізації інформаційної системи

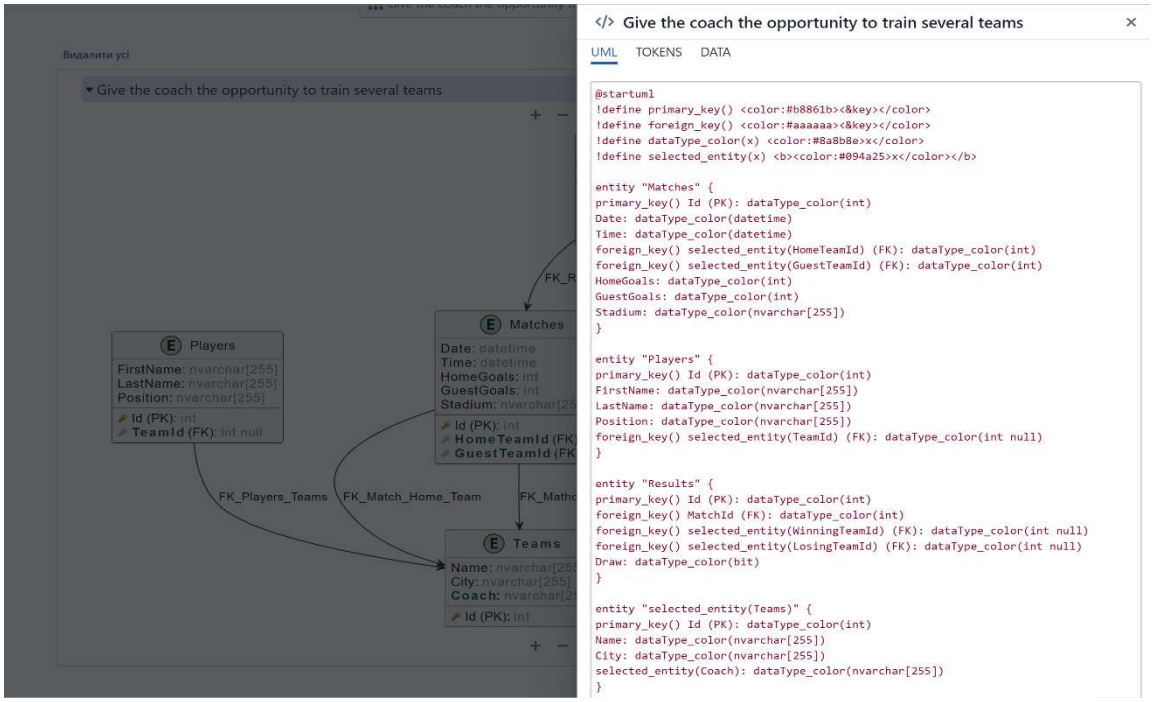

Рис. 9. Результат виділення сукупності артефактів інформаційної системи, яка може бути повторно використана для проєктування та реалізації опублікованої системної вимоги

Порівняння результатів обробки, наведених на рис. 11 та рис. 12, підтверджує релевантність отриманих результатів.

6. Обговорення результатів дослідження

За результатами проведення експериментальної перевірки розробленої ІТ виявлення термінів та артефактів проєкту ІС слід зробити такі висновки:

а) сервіс, який реалізує розроблену ІТ, дозволяє визначати терміни системних вимог до ІС «Футбол», тексти яких опубліковано англійською та українською мовами;

б) сервіс, який реалізує розроблену ІТ, дозволяє визначати терміни, що використовуються в описах артефактів ІС, які можуть бути повторно застосовані для реалізації опублікованих системних вимог у англійській та українській локалізаціях ІС «Футбол»;

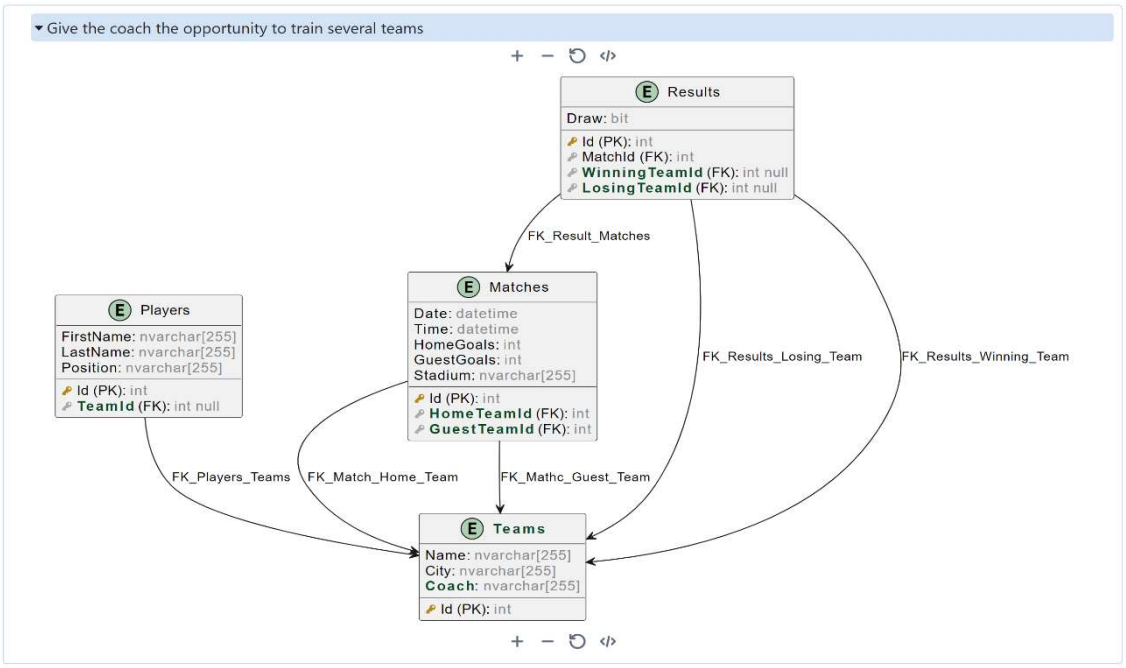

Рис. 10. Візуальна діаграма, яку бачить користувач сервісу як результат обробки опублікованої системної вимоги

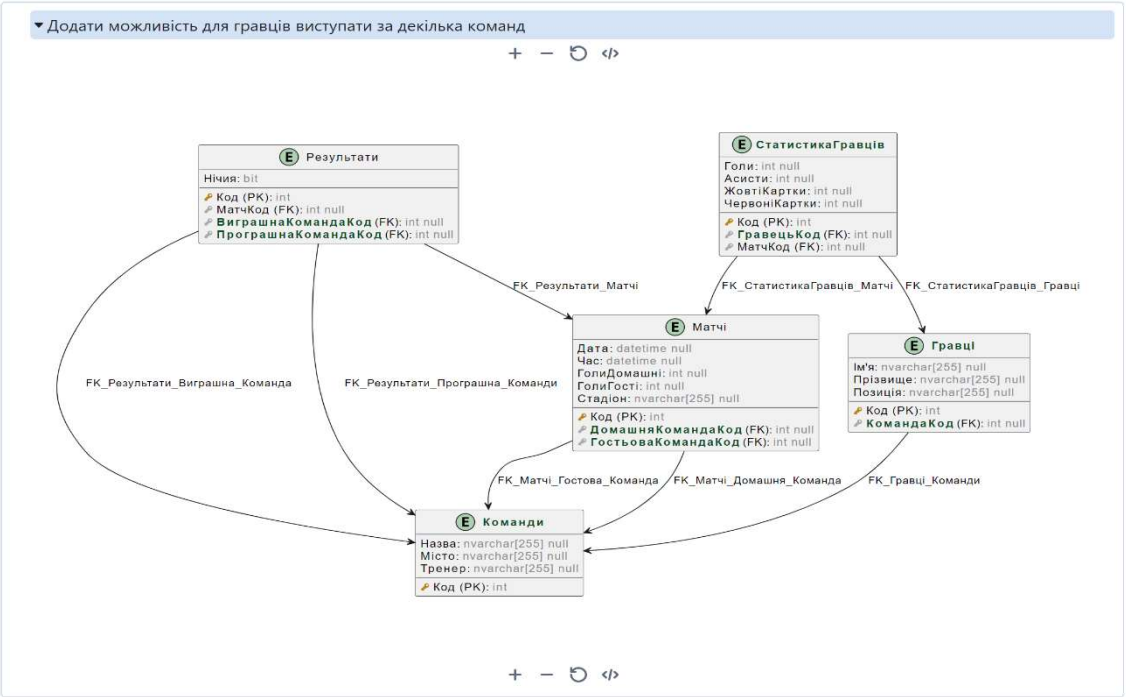

Рис. 11. Візуальна діаграма, яку бачить користувач сервісу як результат обробки системної вимоги, опублікованої українською мовою

в) сервіс, який реалізує розроблену ІТ, дозволяє візуалізувати підмножину артефактів ІС, терміни яких співпадають із термінами опублікованої системної вимоги, у вигляді візуальної діаграми;

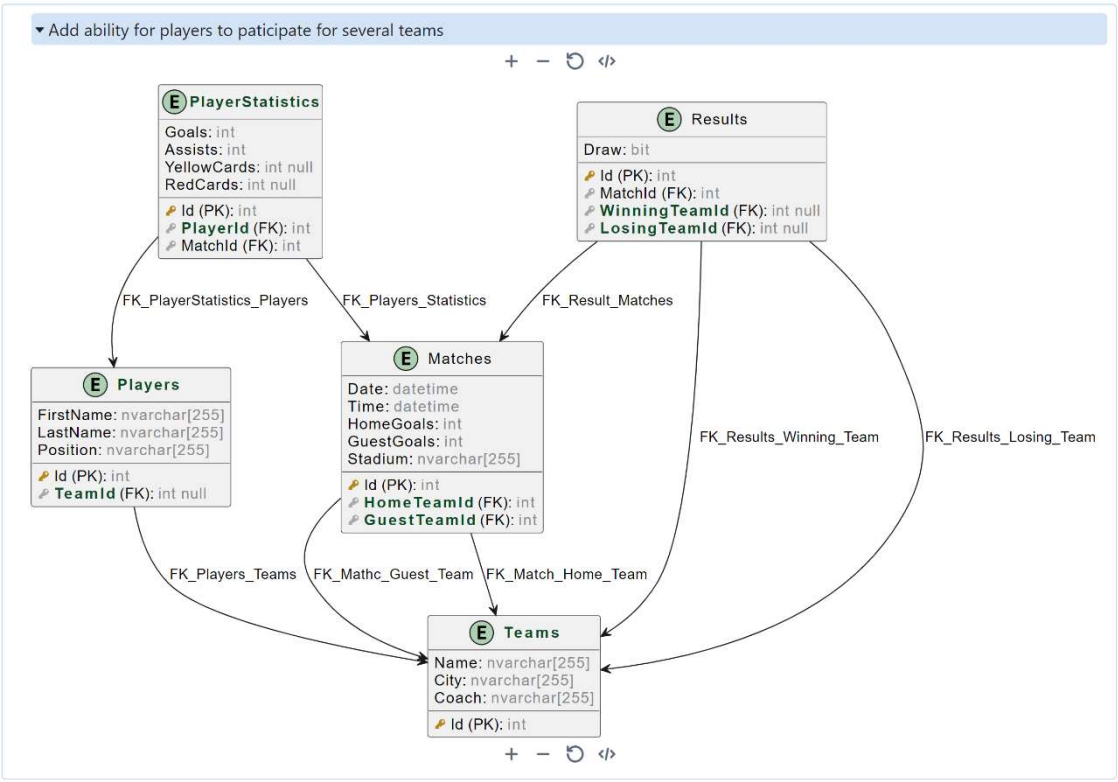

Рис. 12. Візуальна діаграма, яку бачить користувач сервісу як результат обробки системної вимоги, опублікованої українською мовою і перекладеної англійською мовою

г) сервіс, який реалізує розроблену ІТ, забезпечує релевантність результатів виявлення термінів та артефактів проєкту ІС у випадку перекладу тієї самої публікації системної вимоги різними мовами (за умови релевантного перекладу).

У попередніх дослідженнях, які проводилися, в тому числі, одним з авторів цього дослідження, для аналізу текстових публікацій найменувань функцій розроблюваної ІС було використано метод стемінгу [24], [25]. На відміну від цих досліджень, розроблена ІТ виявлення термінів та артефактів проєкту ІС використовує для аналізу текстових публікацій системних вимог методи лематизації, що дозволяє точніше визначати леми в контексті використання слова, забезпечуючи цим підвищення точності ідентифікації термінів, які мають значення для подальшого аналізу вимог.

Але розроблена ІТ виявлення термінів та артефактів проєкту ІС має і недоліки, які визначають декі обмеження її застосування. Серед цих недоліків слід особливо визначити такі:

а) проблеми визначення коректного контексту для артефактів, знайдених у вимогах (наприклад, для вимоги «Increase the number of characters for the team name» атрибут «name» буде визначено системою для кожної сутності, а не тільки для сутності «team», яка зазначена у вхідному тексті вимоги);

б) визначення правильних типів зв’язків (наприклад, «один до одного», «один до багатьох» тощо) потребує, можливо, додаткової розробки та впровадження в ІТ відповідних запитів до сховища, де зберігаються описи артефактів ІС.

Ці недоліки визначають такі основні напрями подальших досліджень з вдосконалення та подальшого розвитку розробленої ІТ виявлення термінів та артефактів проєкту ІС:

а) розвиток та апробація зазначеної ІТ для випадків, коли описами артефактів ІС є

описи патернів програмного забезпечення, фрагментів програмного коду, окремі сервіси та мікросервіси, інші представлення програмних та програмно-апаратних рішень елементів ІС;

б) модифікація зазначеної ІТ для врахування особливостей найпоширеніших підходів, методів та інструментальних засобів формування публікацій вимог зацікавлених сторін та системних вимог до створюваної або вдосконалюваної ІС;

в) дослідження можливості адаптації зазначеної ІТ для аналізу аудіо- та відеофайлів, які містять результати інтерв'ювання представників замовників ІС.

7. Висновки

В процесі виконання даного дослідження було поставлено та вирішено задачу розробки ІТ виявлення термінів та артефактів проєкту ІС. Ця ІТ дозволяє проводити дослідження текстових публікацій системних вимог, які висуваються до створюваної або вдосконалюваної ІС, і, зокрема, визначати артефакти ІТ, які можуть бути повторно використані для реалізації окремих термінів, присутніх в текстовій публікації системної вимоги. В основу цієї ІТ було запропоновано покласти рішення про доцільність застосування для аналізу текстових публікацій системних вимог методів NLP і, зокрема, методів лематизації.

Виходячи з особливостей методів лематизації та основних вимог, які були висунуті до ІТ в процесі її розробки, було визначено основні архітектурні компоненти ІТ та модулі сервісу, які відповідають за реалізацію цих компонентів. Розроблено опис архітектури ІТ у вигляді діаграми компонентів UML, яка описує основні взаємодії між виділеними модулями сервісу, а також між сервісом та сховищем, де зберігаються описи артефактів ІС, та між сервісом і можливими користувацькими інтерфейсами.

Виходячи з розробленого опису архітектури ІТ виявлення термінів та артефактів проєкту ІС, було визначено технологічний стек цієї ІТ. Він дозволяє забезпечити швидку розробку сервісу, який реалізує ІТ, підтримку можливості обробки публікацій системних вимог різними людськими мовами, а також легкість інтеграції окремих модулів у єдиний цілісний сервіс.

Під час розробки елементів ІТ виявлення термінів та артефактів проєкту ІС було використано мови програмування C# та Python. Продемонстровано результати розробки таких елементів програмного забезпечення сервісу, що реалізує ІТ, як опис конфігурації мовних моделей, клас Lang і його метод GetDocument, клас DbEntitiesLemmatizer, який декорує описи артефактів ІС лемами, а також клас PlantUmlBuilder, що відіграє ключову роль у створенні коду для візуалізації діаграми.

Для проведення експериментальної перевірки отриманих результатів було застосовано ІС «Футбол», яка включає до себе набір основних функцій, що дозволяють автоматизувати вирішення задач планування та обліку проведення футбольних матчів. Розроблений сервіс, який реалізує ІТ виявлення термінів та артефактів проєкту ІС, дозволив визначати терміни системних вимог до ІС «Футбол» та артефактів ІС «Футбол», тексти яких опубліковано англійською та українською мовами, візуалізувати підмножину артефактів ІС, терміни яких співпадають із термінами опублікованої системної вимоги, у вигляді візуальної діаграми, а також забезпечити релевантність результатів виявлення термінів та артефактів проєкту ІС у випадку перекладу тієї самої публікації системної вимоги різними мовами (за умови релевантного перекладу).

Отримані результати дозволяють в подальшому вирішити задачу розробки складнішої ІТ, головна задача якої полягає у автоматизації синтезу описів архітектури ІС, яка створюється або вдосконалюється на основі обмеженої множини текстових публікацій вимог зацікавлених сторін або системних вимог до цієї ІС.

Перелік посилань:

1. ДСТУ ISO/IEC/IEEE 15288:2016 Інженерія систем і програмного забезпечення. Процеси життєвого циклу систем. [Електронний ресурс]. Сайт «Друковані видання в цифровому вигляді». URL: https://www.yumpu.com/xx/document/read/67240647/-iso-iec-ieee-15288-2016-/7 (дата звернення: 02.04.2024).

2. Настанова до Зводу знань з управління проєктами. Настанова PMBOK. Сьоме видання. Стандарт з управління проєктами. Project Management Institute, Inc., 14 Campus Boulevard Newtown Square, Pennsylvania 19073-3299 USA, 2021. 370 с.

3. Cadavid H., Andrikopoulos V., Avgeriou P., Broekema P. Ch. System and software architecting harmonization practices in ultra-large-scale systems of systems: A confirmatory case study. *Information and Software Technology*. 2022. Vol.150. № 106984. DOI: https://doi.org/10.1016/j.infsof.2022.106984

4. Revolutionizing Requirements Engineering: Unleashing the Power of NLP and Generative AI [Електронний ресурс]. Сайт «Linkedin». URL: https://www.linkedin.com/pulse/revolutionising-requirements-engineering-unleashing-power-shaun-koon, вільний (дата звернення: 03.04.2024)

5. VisualNarrator Tool [Електронний ресурс]. Сайт «GitHub». URL: https://github.com/MarcelRobeer/VisualNarrator (дата звернення: 03.04.2024).

6. QuARS: A Tool for Analyzing Requirements [Електронний ресурс]. Сайт «QUARS». URL: https://www.quars.it/ (дата звернення: 03.04.2024).

7. Henning Femmer. Requirements Quality Defect Detection with the Qualicen Requirements Scout. Technical University Munichand Qualicen, Munich, Germany [Електронний ресурс]. URL: https://ceur-ws.org/Vol-2075/NLP4RE_paper2.pdf (дата звернення: 04.04.2024).

8. Semantha [Електронний ресурс] . Сайт «Semantha». URL: https://www.semantha.de/semantha-requirements/ (дата звернення: 04.04.2024).

9. ReqSuite [Електронний ресурс]. Сайт «ReqSuite». URL: https://www.osseno.com/en/requirements-management-tool/ (дата звернення: 04.04.2024).

10. IBM Engineering Requirements Quality Assistant [Електронний ресурс]. Сайт «IBM». URL: https://www.ibm.com/docs/en/erqa?topic=assistant-overview, вільний (дата звернення: 05.04.2024).

11. Defining Terms Used to Describe Requirements [Електронний ресурс]. Сайт «Learn Microsoft». URL: https://learn.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2015/modeling/model-user-requirements?view=vs-2015&redirectedfrom=MSDN#RequirementsClasses.

12. Lee R. S. T. Natural Language Processing. A Textbook with Python Implementation. Singapore: Springer Nature Singapore Pte Ltd., 2024. XXXII, 437 p. DOI: https://doi.org/10.1007/978-981-99-1999-4

13. Candase Hokanson, Karl Wiegers. Software Requirements Essentials: Core Practices for Successful Business Analysis. Addison-Wesley Professional. 2023. 208 p.

14. Phillip A. Laplante, Mohamad Kassab. Requirements Engineering for Software and Systems (Applied Software Engineering Series. Taylor & Francis. 2022. 404 p.

15. Levykin V., Yevlanov M., Neumyvakina O., Petrichenko O. Concept of Artifact-Event Description of Information System. Fourth International Scientific and Technical Conference «COMPUTER AND INFORMATION SYSTEMS AND TECHNOLOGIES». Kharkiv: DISA PLUS LLC, 2020. P. 57-58. DOI: https://doi.org/10.30837/IVcsitic2020201438

16. Як вибрати правильний технологічний стек для вашого проекту [Електронний ресурс]. Сайт «REDSTONE». URL: https://redstone.agency/blog/yak-vybraty-pravylnyi-tekhnolohichnyi-stek-dlia-vashoho-proektu/ (дата звернення: 30.10.2024).

17. Introduction to .NET [Електронний ресурс]. Сайт «Learn Microsoft». URL: https://dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core

18. Industrial-Strength Natural Language Processing [Електронний ресурс]. Сайт «SpaCy». URL: https://spacy.io/, вільний (дата звернення: 17.04.2024)

19. SpacyDotNet .NET wrapper [Електронний ресурс]. Сайт «GitHub». URL: https://github.com/AMArostegui/SpacyDotNet

20. Drawing UML with PlantUML [Електронний ресурс]. Сайт «PlantUML». URL: https://plantuml.com/guide

21. The library for web and native user interfaces [Електронний ресурс]. Сайт «React». URL: https://react.dev/learn

22. Blueprint React-based UI toolkit for the web [Електронний ресурс]. Сайт «BlueprintJs». URL: https://blueprintjs.com/docs/, вільний (дата звернення: 23.04.2024).

23. «OpenAPI Guide» [Електронний ресурс]. Сайт «Swagger». URL: https://swagger.io/docs/specification/about/ (дата звернення: 26.04.2024).

24. Ievlanov M., Vasiltcova N., Panforova I. Development of methods for the analysis of functional requirements to an information system for consistency and illogicality. *Eastern-European Journal of Enterprise Technologies*. 2018. T. 1. Vol. 2(91). P. 4–11. DOI: https://doi.org/10.15587/1729-4061.2018.121849

25. Vasiltcova N., Panforova I., Neumyvakina O. Improving a method to analyze the requirements for an

information system for consistency. Eastern-European Journal of Enterprise Technologies. 2020. T. 3. Vol. 2(105). P. 17–27. DOI: https://doi.org/10.15587/1729-4061.2020.205518

Надійшла до редколегії 31.05.2024 р.

Євланов Максим Вікторович, доктор технічних наук, професор, професор кафедри ІУС ХНУРЕ, м. Харків, Україна, e-mail: maksym.ievlanov@nure.ua, ORCID: https://orcid.org/0000-0002-6703-5166 (науковий керівник здобувача вищої освіти Лучицького В.В.).
Мороз Борис Іванович, доктор технічних наук, професор, професор кафедри програмного забезпечення комп'ютерних систем НТУ «Дніпровська політехніка», м. Дніпро, Україна, e-mail: moroz.b.i@nmu.one, ORCID: https://orcid.org/0000-0002-5625-0864;
Мороз Дмитро Максимович, доктор філософії, доцент, доцент кафедри програмного забезпечення комп'ютерних систем НТУ «Дніпровська політехніка», м. Дніпро, Україна, e-mail: moroz.d.m@nmu.one, ORCID: https://orcid.org/0000-0003-2577-3352
Лучицький В'ячеслав Володимирович, здобувач вищої освіти, група УПГІТм-22-3, факультет комп'ютерних наук, ХНУРЕ, м. Харків, Україна, e-mail: viacheslav.luchytskyi@nure.ua.

УДК 004.8:004.9 DOI: 10.30837/0135-1710.2024.182.093

Є.В. БОДЯНСЬКИЙ, О.С. ЧАЛА

ШВИДКА КЛАСИФІКАЦІЯ В ONLINE- ТА NEARLINE-РЕЖИМАХ В УМОВАХ КЛАСІВ, ЩО ПЕРЕТИНАЮТЬСЯ

Предметом дослідження є процес швидкої класифікації в умовах класів, що перетинаються. Метою є розробка підходу до швидкої класифікації, який поєднує online- та nearline-режими для підвищення точності класифікації в умовах класів, що перетинаються. Розроблено багатошарову нейронну мережу з ядерними дзвонуватими функціями активації для роботи в online-режимі. Запропоновано адаптивну нейро-фаззі систему для класифікації даних у матричній формі, яка використовує гібридне комбіноване навчання для роботи в nearline-режимі. Підхід орієнтований на вирішення задач уточнення границь між класами, адаптації до змін у розподілі вхідних даних, усунення дисбалансу класів та видалення шумових точок.

1. Вступ

Класифікація, яка є однією з ключових задач машинного навчання, полягає у віднесенні вхідних даних до одного з попередньо визначених класів. Ця задача має широкий спектр практичних застосувань, включаючи розпізнавання об'єктів на фотографіях, виявлення комп'ютерних вірусів, аналіз відеоданих із систем відеоспостереження тощо [1].

Швидка класифікація передбачає обробку даних у режимі реального часу (real time) або близькому до нього (near real time). Вирішення задачі швидкої класифікації потребує розробки архітектур нейронних мереж, здатних забезпечити високу точність класифікації при мінімальних обчислювальних витратах. Такі системи мають практичне застосування у задачах, де швидкість обробки є ключовим фактором, наприклад, у системах підтримки керування транспортними засобами або безпекового моніторингу [2].

Швидка класифікація поєднує online- та nearline-класифікацію. Online-класифікація орієнтована на обробку даних у режимі реального часу, що передбачає задоволення часових обмежень та, як наслідок, обмежень щодо обчислювальної складності алгоритмів обробки даних. Nearline-класифікація передбачає затримку в декілька хвилин на обробку даних, що дає можливість використовувати складніші моделі та отримати результати з вищою точністю. Як online-, так і nearline-класифікація передбачають узгодження швидкості та точності класифікації [3].