МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

першого (бакалаврського) рівня вищої освіти

на тему: **“ Розробка програмного каркасу розподіленої системи обробки природномовних текстів ”**

Виконав: студент гр. Іт-22сп_____
Спеціальності 126 – «Інформаційні системи та технології»_____
(шифр і назва)

_____Зінько Роман Андрійович_____
(Прізвище та ініціали)

Керівник: _____к.т.н., в.о. доц. Боярчук О.В.___
(Прізвище та ініціали)

Рецензенти: _____
(Прізвище та ініціали)

_____
(Прізвище та ініціали)

ДУБЛЯНИ-2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Освітній ступінь «Бакалавр»

126 – «Інформаційні системи та технології»

“ЗАТВЕРДЖУЮ”
Завідувач кафедри_____
д.т.н., проф. А.М. Тригуба
“____”_____2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу студенту

Зінько Роман Андрійович

1. Тема роботи: «Розробка програмного каркасу розподіленої системи обробки природномовних текстів»

Керівник роботи Боярчук О.В, к.т.н.,в.о. доц.
Затверджені наказом по університету 30.12.2022 року № 453/к-с.

2. Строк подання студентом роботи 10.06.2023 р.

3. Початкові дані до роботи: 1. Вимоги до побудови програмного каркасу. 2. Науково-технічна і довідкова література. 3. Засоби використання та структура API Visual Studio Code. 4. Методика побудови архітектури додатку.

4. Зміст розрахунково-пояснювальної записки:

1. *АНАЛІЗ ІТ-ЗАСОБІВ ДЛЯ НАПИСАННЯ ТЕКСТІВ.*
2. *РОЗРОБКА АРХІТЕКТУРИ ІНСТРУМЕНТУ ДЛЯ НАПИСАННЯ ТЕКСТІВ*
3. *ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗРОБКИ*
4. *ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ Висновки*
   ВИСНОВКИ
   СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
   ДОДАТКИ

5. Перелік презентаційного матеріалу : 1.Особливості написання природномовних текстів, 2. Аналіз інструментів для перевірки природномовних текстів, 3. Архітектура та базові засоби розширення visual studio code, 4. Спосіб публікації розширення, 5. програмна реалізація розробки в мовному клієнті visual studio code, 6. Розробка модуля діагностики та доповнення, 7. Протоколи мовного сервера, 8. Результати порівняльної оцінки протоколів мовного сервера, 9. Мовний сервер, 10. Програмна підтримка текстів граматичним словником mphdict, 11. Висновки.

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3	Боярчук О.В., в.о. доцента кафедри інформаційних технологій		
4	Городецький І.М., доцент кафедри управління проектами та безпеки виробництва		

7. Дата видачі завдання 30.12.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Етапи виконання кваліфікаційної роботи	Строк виконання етапів роботи	Приміт-ка
1.	*Написання першого розділу та означення головних завдань роботи*	30.12.22-01.01.23	
2.	*Виконання другого розділу та формування головних показників для розрахунків*	01.01.23-01.02.23	
3.	*Виконання третього розділу та формування початкових даних*	01.02.23-01.03.23	
4.	*Виконання четвертого розділу та узагальнення отриманих результатів роботи*	01.02.23-01.03.23	
5.	*Завершення оформлення розрахунково-пояснювальної записки та презентації*	01.04.23-01.05.23	
6.	*Завершення роботи в цілому*	01.05.23-16.06.23	

Студент _____ Зінько Р.А.
(підпис)

Керівник роботи_____ Боярчук О.В.
(підпис)

УДК 004.6. : 003.03.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновку, переліку посилань з 30 найменувань, 1 додатку, і містить 20 рисунків, 31 таблицю. Повний обсяг роботи складає 68 сторінки.

Розробка програмного каркасу розподіленої системи обробки природномовних текстів. Зінько Р.А. Кафедра ІТ. – Дубляни, Львівський НУП, 2023.

Проблема асистування створенню текстів здебільшого зводиться лише до перевірки — орфографії та синтаксису, а підказки рідко виходять за межі доповнення префіксу до леми чи найчастотнішої словоформи, саме тому тема роботи є актуальною, вихідний продукт роботи підвищить ефективність та якість написання текстів.

В роботи розвинуто мовну підтримку для написання текстів за рахунок: створення синтаксичних аналізаторів з врахуванням афіксів, удосконалення провайдерів розумного доповнення, створення мовного сервера, удосконалення протоколу мовного сервера.

Створення виділеного мовного сервера забезпечує зменшення навантаження на мовний клієнт, можливості оновлення мовних провайдерів без необхідності оновлення мовного клієнта та покращило швидкість надання мовної підтримки.

Протокол мовного сервера удосконалено за рахунок обрання протоколу більш низького рівня в якості транспортного, це забезпечує зменшення обсягу повідомлення.

Ключові слова: *текст, мовний клієнт, мовний сервер, visual studio code, розширення, аналізатор*.

ЗМІСТ

ВСТУП

Така функція редактора текстів, як інтелектуальне асистування, допомагає значно скоротити час написання, рецензування та редагування текстів.

На даний момент майже для кожної штучної мови існує велика кількість редакторів чи повнофункціональних середовищ розробки, оскільки для штучної мови заздалегідь зазначені граматика, алфавіт та словник, вони є повністю формальними мовами (мають скінченну послідовність символів, які описуються правилами певного виду).

Разом з тим проблема асистування створенню природномовних текстів здебільшого зводиться лише до перевірки — орфографії, синтаксису та, можливо, певних стилістичних показників тексту, а підказки рідко виходять за межідоповнення префіксу до леми чи найчастотнішої словоформи.

Необхідність розширення репертуару перевірок та підказок для підвищення ефективності роботи користувача, який створює чи редагує природномовний текст і визначає актуальність обраної теми.

Мета роботи — побудова підсистеми інтелектуального редактора природномовних текстів на основі Visual Studio Code.

Об’єкт — підсистема інтелектуального редактора природномовних текстів.

Предмет— підсистема інтелектуального редактора природномовних текстів на основі Visual Studio Code.

Для досягнення поставленої мети необхідно виконати такі завдання:

- розглянути особливості природномовних текстів та інструментальне забезпечення для їх використання;
- розробити архітектуру розширюваного середовища для написання природномовних текстів;
- розробити архітектуру мовного сервера для написання природномовнихтекстів,

- описати та розробити протокол мовного сервера;
- побудувати програмну реалізацію системи;
- розробити опис використання програмної системи.

Потенційними користувачами даного програмного забезпечення стануть рецензенти, редактори текстів, письменники, журналісти та інші корисутвачі текстових редакторів.

РОЗДІЛ 1

АНАЛІЗ ІТ-ЗАСОБІВ ДЛЯ НАПИСАННЯ ТЕКСТІВ

1.1 Особливості написання та обробки текстів

На відміну від предметно-орієнтованих мов програмування [8-10], природні мови не є формальними мовами, не мають скінченну послідовність символів, які описуються правилами певного виду, також їх граматика та алфавіт змінюються час від часу.

Текст, написаний природною мовою неможливо записати формальною мовою, тому наявні ІТ-інструменти для написання текстів природною мовою досить складні, деякі задачі неможливо виконати повністю правильно.

Кожна природна мова має свої морфологічні та синтаксичні особливості, які слід враховувати при створенні інструментів для написання природномовних текстів даною мовою.

Отже, характеристиками природної мови є:

- алфавіт;
- набір правил для словотворення;
- набір синтаксичних та морфологічних правил [11-14].

Одиниця алфавіту — графема.Ознаки графеми:

- тип знаку;
- фонетичні ознаки;
- розмір графеми;
- належність до алфавіту.

Набір правил для словотворення з використанням регулярних виразів наведений на рисунку 1.1.

Код класу	Найменування лексеми	Опис правила	Приклад
L01	Слово з малої літери	([а-я]+)	танк
L02	З великої літери	([А-Я][а-я]*)	Михаил
L03	Іншомовне слово	[a-zA-Z]+	tomahawk
L04	Неповна лексема	([а-я]+)-	полу-
L05	Слово через дефіс	([а-я]+)-([а-я]+)	полу-

Рисунок 1.1 — Можливий набір правил створення лексем для природної мови.

Правила синтаксису поділяються на:

- контекстні синтаксичні правила;
- правила виокремлення присудка і підмета;
- правила виокремлення другорядних членів речення.

Набір контекстних синтаксичних правил з використанням узгодження, керування та прилягання наведений на рисунку 1.2.

Який клас	З яким класом	рід	число	відмінок	Порядковий № гол. сл.	Синтаксичний тип	Приклад
2*	1*	+	+	+	(2)	С1	державні органи
23*	1*			2	(2)	У1	від форми
1*	1*			2	(1)	У2	генератор шуму
14*	8*				(2)	П1	швидко біг

Рисунок 1.2 — Можливий набір синтаксичних та морфологічних правил.

Набір правил для словоутворення формують лематизаційні словники, а синтаксичних правил — синтаксичні. Обробка природної мови на цих рівнях фактично зводиться до обробки даних відповідних таблиць.

В даній роботі, в якості природної мови було вибрано українську мову. Головною особливістю української природної мови є використання словотворчих та формотворчих афіксів (морфем, приєднаних до кореня), належить до типу флективних мов.

Усі природні мови поділяються на:

- кореневі (китайська, бірманська);
- аглютинативні (тюркські й фінно-угорські мови);

- флективні (українська, англійська, французька, болгарська);
- полісинтетичні (ацтекська мова).

Кореневі мови — мови, які не мають афіксів і граматичні значення виражають способом прилягання одних слів до інших або за допомогою службових слів.

В аглютинативних мовах граматичні форми й похідні слова утворюються додаванням однозначних та стандартних афіксів до незмінюваних основ слів.

У флективних мовах граматичні форми й похідні слова утворюються за допомогою додавань флексій (багатозначних закінчень).

Усі флективні мови поділяються на:

- синтетичні (українська, російська, польська, литовська);
- аналітичні (англійська, французька, болгарська).

У синтетичних мовах відношення між словами виражаються завдяки формам слів, а в аналітичних — за допомогою службових слів і порядком розташування повнозначних слів.

В полісинтетичних мовах морфеми приєднуються разом і групують єдине

ціле.

Алфавіт української мови складається з 33 літер, особливостями українського

алфавіту у порівнянні з іншими кириличними є наявність букв “Ґ”, “Є” та “Ї”.

В українській мові виділяють лише два види морфем (найменша частина слова, яка складається з фонем): корені, афікси.

У залежності від позиції в слові афікси розрізняють на: префікси, постфікси, інтерфікси (афікс, що розташований між коренями складного слова), циркумфікси (одночасне поєднання префікса і суфікса) та інфікси.

В українській мові виділяються десять частин мови: 6 самостійних та службових.

В українській мові іменники мають такі граматичні характеристики:

- мають один з трьох родів: чоловічий, жіночий, середній;
- змінюються за числами: одниною, множиною та двоїною;
- змінюються за відмінками.

В української мові іменник має узгодження с прикметником.

Дієслово має 4 часи (давноминулий, минулий, теперішній, майбутній), усі дієслова поділяються на два види: доконаний і недоконаний, також деякі дієслова є двовидові. У теперішньому та майбутньому часі дієслова відмінюються за особою тачислом. У минулому часі — за родом та числом.

Основним способом утворення форм множини в українській мові є зовнішня флексія, іноді супроводжувана змінами кореневих приголосних.

Основним типом граматичних афіксів в українській мові є флексія (належать до постфіксів).

Флексія — закінчення, за допомогою якого створюються граматичні форми слів у певній системі словозміни змінюваних частин мови.

Українська мова має наступні національно-своєрідні морфологічні особливості: невідмінювання першого компонента у складних числівниках, які означають назви десятків, інфінітивна форма дієслова закінчується на “-ти” та інші.

Українська мова має наступні національно-своєрідні синтаксичні особливості: використання словосполучень, утворені підрядним зв’язком керування, використання конструкцій з предикативними формами на “-но”, “-то” з дієслівною власне-зв’язкою “бути” та інші.

Загальна схема складових обробки природньої мови представлена на рис.1.3 [15-17]:

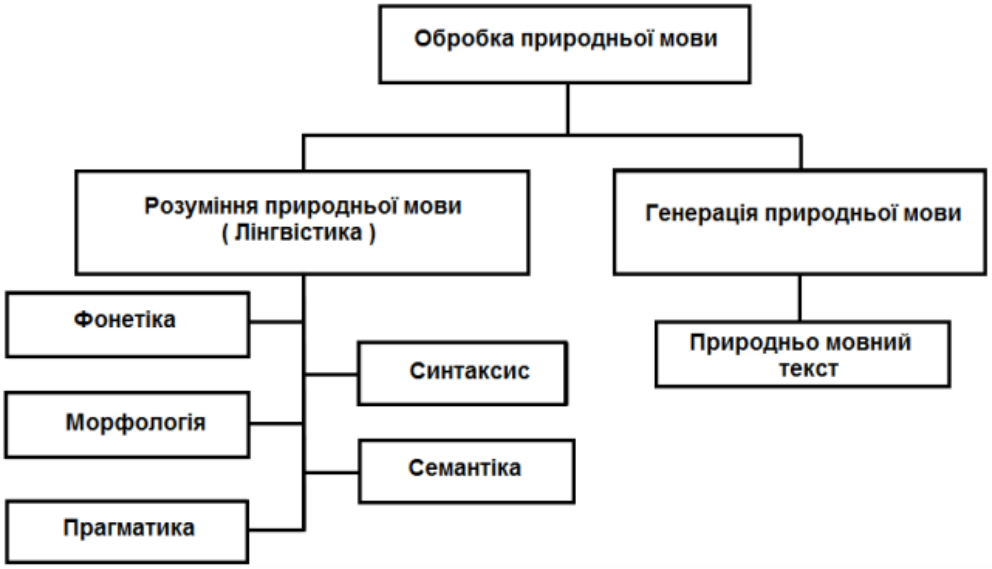

Рис.1.3 Схема складових обробки природньої мови

Морфологічний аналіз — перевірка правильності написання слів за словниками.

Синтаксичний аналіз — аналіз вхідної послідовності символів з метою розбору граматичної структури з перевіркою відповідності граматичним правилам.

Семантичний аналіз — встановлення смислової правильності синтаксичних конструкцій в реченнях.

Прагматичний аналіз — встановлення смислової правильності між абзацами, діалогами та текстами.

Найбільшою проблемою обробки природномовних текстів є проблема омонімії, оскільки неможливо використовуючи лише граматичний словник завжди правильно зіставити вхідну лексему слову, яке використав користувач. Розпізнавання вхідного тексту потребує величезних обсягів інформації, зазвичай це реалізовується у вигляді онтології [18].

Існують наступні обмеження для правильної обробки природномовних текстів:

- референційна неоднозначність;

- відмінкова неоднозначність;
- синтаксична неоднозначність;
- смислова неоднозначність.

Останнім часом, дані обмеження вдається уникати за допомогою систем штучного інтелекту [19], а сама обробка текстів зводиться до AI-повної задачі (еквівалентність обчислювальної складності цих задач створенню комп'ютерів, настільки ж розумних, як і люди).

1.2 Аналіз інструментів для написання текстів

Інструментом для написання текстів є текстовий редактор.

Текстовий редактор — програмний застосунок для написання та редагування текстів.

Більшість наявних текстових редакторів надають певний функціонал для прискорення написання текстів: розумне доповнення, підсвічування синтаксису, створення шаблонів, сортування рядків, мають розширені функції форматування тексту, наприклад вставка графіків, посилань, таблиць.

Деякі з інструментів для написання природномовних текстів забезпечують перевірку орфографії, що є досить складною для реалізації можливістю, тому не завжди виконується повністю правильно.

Перевірка орфографії — це процес пошуку слів, написаних неправильно та створення можливих корекцій. Перевірка орфографії та автокорекція широко застосовуються для таких завдань, як обробка слів і постобробка тексту з фотографії.

Більшість орфографічних систем потребують певних мовних ресурсів, такихяк лексика, списки неправильних написань чи бази правил.

Для забезпечення мовної підтримки, інструменти для написання

природномовних текстів використовують словники слів, які зазвичай зберігаються у реляційних або постреляційних баз даних. Якість такого інструменту в першу чергу залежить від об’єму словника та дати створення словника.

Текстові редактори зазвичай використовуються наступні типи словників:

- граматичний словник;
- етимологічний словник;
- словник синонімів.

Граматичний словник зазвичай є словником квазіфлексій [20] та містить мінімальну послідовність букв, яку необхідно відкинути від шуканого слова та

послідовність букв, яку необхідно додати, номер граматичної категорії, номер словозмінного класу. Граматичний словник приводить словоформу до початкової форми.

Недоліками наявних інструментів для написання природномовних текстів є:

- використання неактуальних граматичних словників, або повна їх відсутність;
- відсутність стандартного мовного протоколу;
- більшість з них не використовують мовний сервер;
- відсутність можливості розширення їх мовно-підтримуючого функціоналу. Відсутність мовного сервера приводить до переліку інших побічних проблем,

такі як:

- оновлення бази словника слів потребує оновлення усього інструменту длянаписання текстів;
- зменшення продуктивності мовного клієнта;
- низька швидкість аналізу

текстів. Перевагами відсутності мовного

сервера є:

- відсутність вимоги до постійного з’єднання до мережі Інтернет;
- дані користувача не залишають його пристрій та не відправляються

третімособам.

Прикладами інструментів для перевірки написання природномовних текстів є:Hunspell, Windows Spellchecker.

Веб-сервісами для перевірки орфографії є LanguageTool, Grammarly, Onlinecorrector, Languagetool.

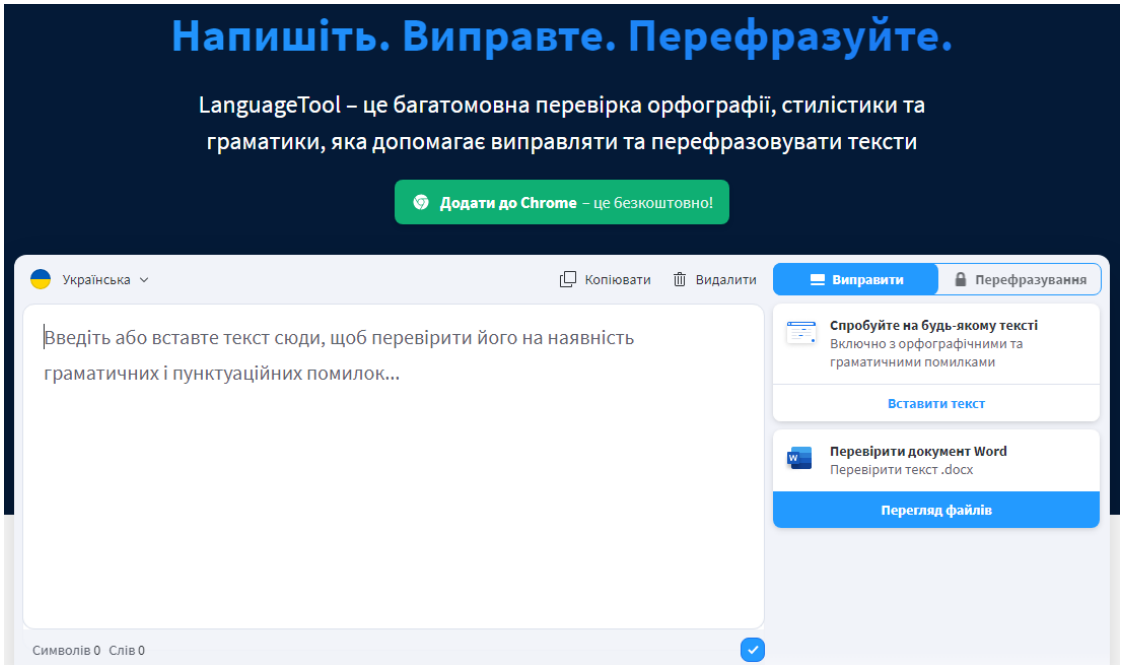

Рисунок 1.4. Інструментів для перевірки написання текстів «LanguageTool»

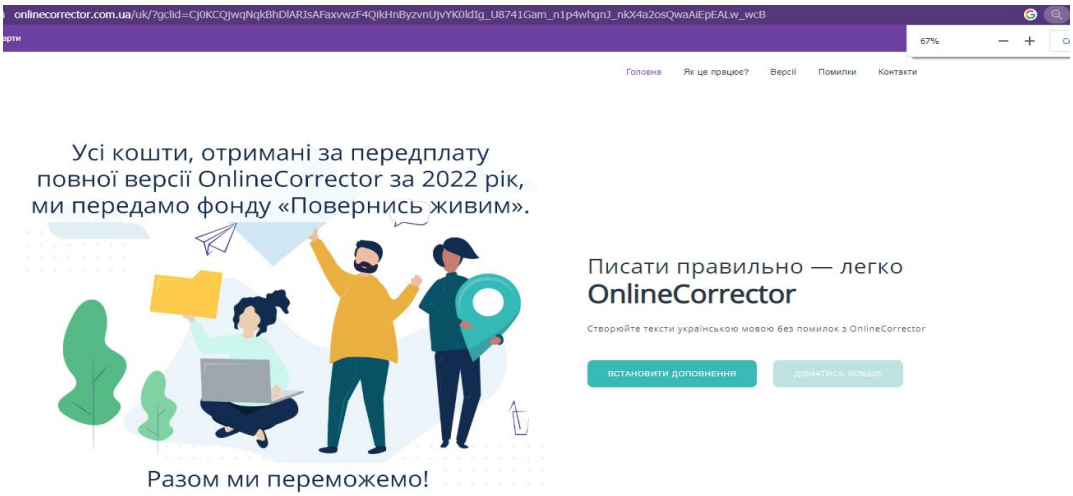

Рисунок 1.5. Інструментів для перевірки написання текстів «OnlineCorrector»

Вказані сервіси надають лише обмежену мовну підтримку для перевірки орфографії, не підтримують створення певних правил користувачем і не є текстовими редакторами, в той час як наявні текстові редактори зазвичай не надають можливості для знаходження орфографічних та синтаксичних помилок.

1.3 Написання текстів у Visual Studio Code

Серед наявних розширень для написання текстів у Visual Studio Code на офіційному магазині розширень українською мовою, який доступний з меню VS Code, або за посиланням “https://marketplace.visualstudio.com/” на даний момент існують лише наступні: Ukrainian — Code Spell Checker, Ukrainian support for LteX та Ukrainian Support for LanguageTool.

Вказані розширення не використовують мовні сервери та потребують інсталяцію інших розширень.

Оскільки кожне з розглянутих розширень має перелік недоліків, доцільно було би створити нове розширення на базі наявних, яке вміщає в собі переваги кожного з розширень та має більш широкий функціонал та підвищену людино-машинну взаємодію.

Метою даної роботи є побудова мовного клієнта — розширення для середовища розробки Visual Studio Code для написання текстів та створення мовного сервера.

Для досягнення даної мети потрібно спроектувати та розробити наступні модулі мовного клієнта:

- модуль розумного доповнення;
- модуль діагностики;
- модуль підказок.

Мовний клієнт повинен мати наступні функціональні можливості:

- можливість одночасної роботи з декількома відкритими

документами;

- можливість зміни режиму роботи модуля діагностики;
- збереження поточних налаштувань користувача до глобальних налаштувань Visual Studio Code;
- можливість зміни активної локалізації.

РОЗДІЛ 2
РОЗРОБКА АРХІТЕКТУРИ ІНСТРУМЕНТУ ДЛЯ НАПИСАННЯ ТЕКСТІВ

2.1 Базові засоби розширення Visual Studio Code

Visual Studio Code — високопродуктивний редактор коду, який містить служби для мов програмування та потужне середовище розробки. Розроблений на базі іншого редактора коду — Atom.

Visual Studio Code має вбудовану підтримку мов JavaScript та TypeScript, вбудований термінал, який можна використовувати для виконання команд оболонки, вбудовану систему керування версій та включає підтримку систему керування версій Git. Багато інших систем керування версій доступні через розширення на VS Code Marketplace.

Visual Studio Code використовує Node.js. Node.js — це платформа для створення швидких і масштабованих серверних додатків за допомогою JavaScript. Node.js — це середовище виконання, а NPM — менеджер пакетів для модулів Node.js. Щоб запустити додаток Node.js, на комп'ютері потрібно бути встановлене середовище виконання Node.js.

VS Code містить інші програмні продукти:

- Yeoman — інструмент створення додатків;
- Aspnet генератор — генератор Yeoman для додатків ASP.NET Core;
- Hottowel генератор — генератор Yeoman для швидкого створення AngularJS-додатків;
- Express — фреймворк для Node JS застосунків;
- Gulp - система виконання завдань;
- Mocha — фреймворк на JavaScript для тестування, що працює на Node.js;
- Bower — менеджер пакетів клієнтської сторони.

Всі розширення [21] VS Code мають загальну модель реєстрації, активації(завантаження) та доступу до інтерфейсу VS Code.

Існують два спеціальних типи розширень VS Code, мовних серверів і налагоджувачів.

Основними засобами VS Code є:

- розширення — базовий будівельний модуль;
- сервер мови — основний модуль обробки тексту певною мовою;
- дебагери — зовнішній налагоджувач, під'єднаний через адаптер.

Загальна архітектура розширення для VS Code подана на рисунку 2.1.

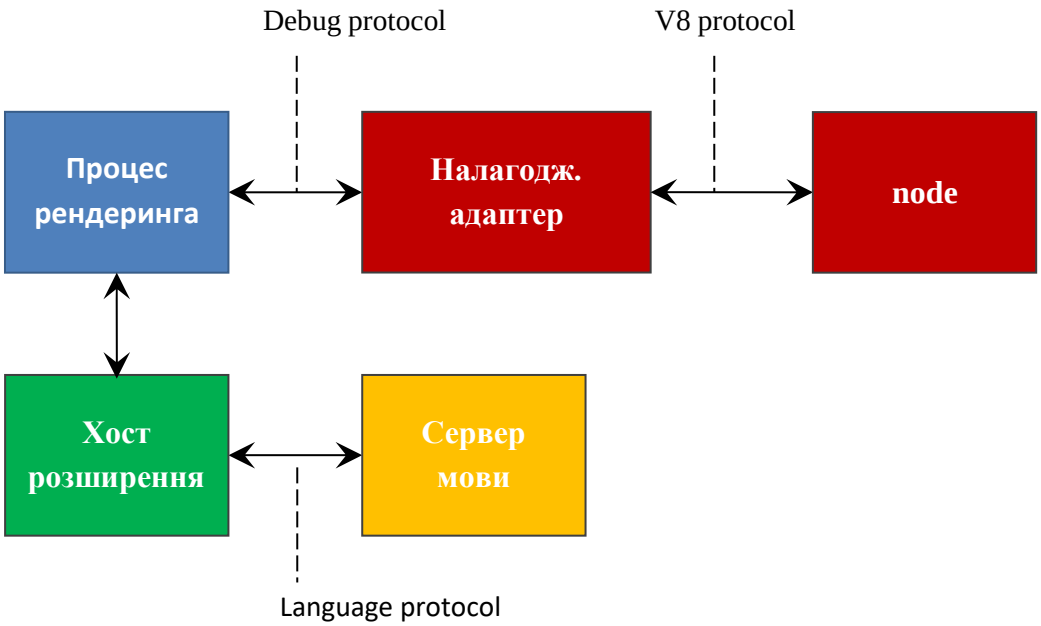

Рисунок 2.1 — Архітектура VS Code розширення.

VS Code та розширення для неї, використовують для рендерингу фреймворк Electron, який надає можливість створювати десктопні програми за допомогою веб –програмування, який використовує рушій Chromium.

Усі розширення після активації виконуються в процесі хосту спільного доступу, який не дає можливості для прямого доступу до об'єктної моделі Visual Studio Code. Дана реалізація розширень у виділеному хості має ряд недоліків та обмежує розробників розширень, проте цей окремий процес для розширень гарантує, що VS Code буде правильно реагувати на усі події та буде надавати

чуткий інтерфейс користувачу.

Окремий хост та ізоляція розширення забезпечує стабільність VS Code, як програмного продукту, внаслідок чого, користувач повністю контролює роботу VS Code, розширення не може заборонити відкривання певного файлу, створення чи видалення, не може виконати включення чи виключення інших розширень, не може виконати ті дії, які не бажає користувач. Хост розширення — це процес Node.js, який відкриває API VS Code для розробників розширень.

VS Code надає підтримку налагодження для розширень, що працюють всередині хосту розширення.

VS Code активує розширення настільки пізно, наскільки це можливо, для забезпечення правильного завантаження VS Code та забезпечення режиму максимальної потужності. Розширення VS Code, які виключені в активній сесії не завантажуються взагалі, і тому вони не споживають оперативну пам'ять. Для забезпечення лінивого завантаження VS Code визначає події, які ініціюють завантаження розширення. Це може бути відкриття певного типу файлу, команда розширення, яка вибирається за допомогою меню команд або комбінації клавіш.

Розширення включають підтримку:

- активація розширення;
- редактор — робота з контентом редактора, наприклад читання та маніпулювання текстом;
- робоча область — відкриті редактори, панель статусу, інформаційні повідомлення та інше;
- події — підключення до подій життєвого циклу редактора, таких як:відкриття, закриття, зміна тощо;
- розвинуте редагування — створення провайдерів для підтримки багатьохмов, включаючи IntelliSense, Peek, Hover.

Можливості розширень в VS Code:

- підсвічування синтаксису;
- розумні доповнення (IntelliSense);
- діагностика коду та виправлення;

- показ інформації про сигнатури функцій;
- навігація коду (перейти до визначення, знайти всі посилання);
- налагодження;
- форматування коду;
- рефакторинг.

Загальним паттерном створення розширення у VS Code є виконання коду розширення в окремому процесі, який зв’язується з VS Code через протокол. Прикладами цього в VS Code є мовні сервери та налагоджувальні адаптери. Як правило, цей протокол використовує stdin / stdout для взаємодії між процесами, використовуючи JSON.

Використання окремих процесів забезпечує хороші межі розділення виконання коду, що допомагає VS Code зберегти стабільність основного редактора.

2.2. Сервер мови

Сервери мови дозволяють створювати виділений процес для розширення. Перевірка та обробка певної мови може займати великі ресурси, особливо коли перевірка виконується одночасно для декількох файлів і створюються дерева лексичного та синтаксичного розбору. Для того, щоб мінімізувати втрату продуктивності, мовні сервери в VS Code виконуються в окремому процесі.

Мовні сервери можуть бути написані будь-якими мовами, проте VS Code надає прикладний інтерфейс лише для мов Javascript та Typescript.

Зазвичай сервер мови використовується, коли розширення працює над інтенсивними завданнями CPU або IO, які можуть сповільнити роботу інших розширень.

VS Code дозволяє застосувати сервер мови як автономний сервер, що реалізує протокол мовного сервера, або безпосередньо реєструвати постачальників у методі активування розширення:

- protocol language server protocol;
- direct implementation.

Для створення мовного сервера на мові Typescript можна використати офіційну бібліотеку “vscode-languageserver”, яка надає інтерфейси IConnection IPC MessageReader, IPC Message Writer. IConnection використовує Node JS IPC для міжпроцесорної взаємодії (рисунок 2.2). За допомогою методу listen класу Tex Document, менеджер документів реєструє підписку на зміну, відкриття нового та закриття документів для переданого до методу параметру — підключення.

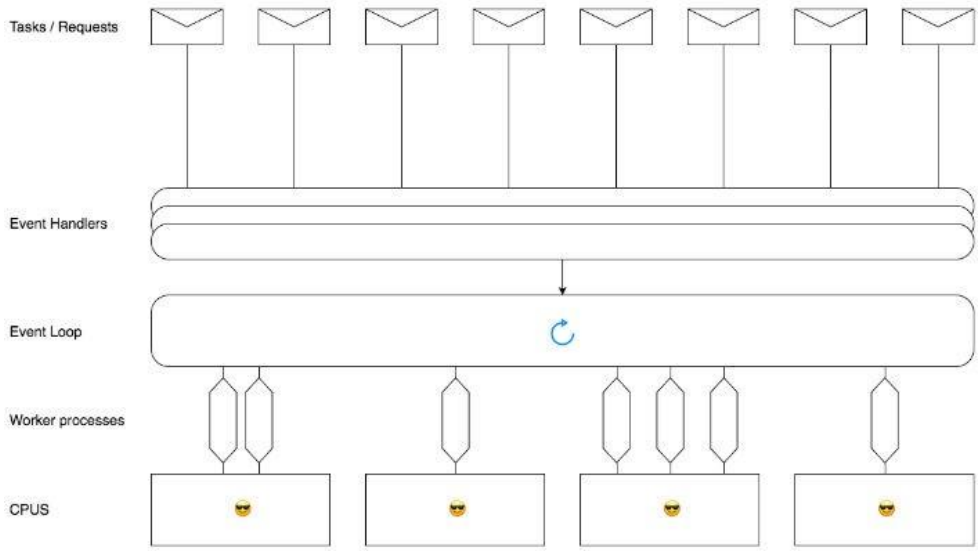

Рисунок 2.2 – Node JS IPC.

У даній роботі, в якості мовного сервера використовується явна реєстрація провайдерів у методі активації розширення — direct implementation.

Функція активації розширення — activate, яка є експортованою, спочатку завантажує та обробляє збережені налаштування користувача, реєструє команди дій

користувача, реєструє провайдери, реєструє події, налаштовує інтерфейс користувача та робить первинну обробку відкритого файлу (парсинг).

2.3. Маніфест розширення

Кожне розширення VS Code потребує файл маніфесту (опис розширення). Файл маніфесту — це файл package.json, який знаходиться у корені каталогу розширення. Він забезпечує VS Code всією необхідною інформацію, яка в першу чергу також використовується при опублікуванні розширення в магазині розширень. Маніфест даного розширення включає інформацію про назву розширення,

опис, версію, категорію, події які ініціюють включення розширення, список тем та команди, список налаштувань користувача, список клавіш швидкого доступу, інформацію про ліцензію та залежності від інших бібліотек, посилання на репозиторій, мінімальну підтримуючу версію VS Code, категорії до яких відноситься розроблене розширення та події, які активують розширення, шлях до виконуваного файлу.

Розширення, які відносяться до однієї категорії групуються разом у магазині розширень, що прискорює фільтрацію та пошук. Інформація про категорії може включати наступні значення: Programming Languages, Snippets, Linters, Themes, Debuggers, Formatters, Keymaps, SCM Providers, Other, Extension Packs, Language Packs. Дане розширення відноситься до наступних категорій: Languages, Themes, Other.

Список налаштувань користувача (ключ contributes.configuration) надає інформацію про налаштування за замовчуванням робочої області та глобальних налаштувань користувача. Дане розширення вказує глобальні налаштування локалізації — “en-US” та режиму роботи модулю діагностики — “true” (активований).

Ключ contributes.keybindings вказує перелік клавіш швидкого доступу з відповідної приєднаною командою користувача та можливою умовою виконання. Список клавіш швидкого доступу даного розширення вказаний у таблиці 2.1.

Таблиця 2.1. Список клавіш швидкого доступу

Назва команди	Опис команди	Клавіша швидкого доступу
extension.aboutVersion	Version	ctrl+f0
extension.startWork	Запуск розширення	ctrl+f1

Слід зазначити, існує можливість, що у Visual Studio Code, може бути встановлене інше розширення, яке має команди з тими ж самими вказаними клавішами швидкого доступу. Тоді виникає деякий конфлікт, оскільки клавіша прив’язана одночасно до двох чи більше команд, що може спричинити неправильну поведінку кожного з розширень.

Меню налаштувань швидкого доступу надає можливість користувачу дізнатися про перелік наявних конфліктів та змінити комбінацію на іншу.

Файл конфігурації мови включає інформацію про типи коментарів, автозакриваючі символи, лексеми, які автоматично виконують форматування тексту та пари лексем, які автоматично підсвічуються.

Оскільки хост розширення — це процес Node.js, розробник може використовувати API Node.js у своїх розширеннях, або використовувати наявні модулі Node.js при розробці розширення. В цьому випадку розробник визначає свої модульні залежності в package.json (ключі dependencies та devDependencies), і використовує npm для встановлення Node.js модуля. Приклад використання залежностей наведений на рисунку 2.3.

```
"dependencies": {
    "mssql": "^4.0.4",
    "mysql2": "^1.5.1"
},
"devDependencies": {
    "typescript": "^2.6.1",
    "vscode": "^1.1.6",
    "@types/node": "^7.0.43",
    "@types/mocha": "^2.2.42"
},
```

Рисунок 2.3 – Використання маніфесту розширення.

2.4. Особливості API Visual Studio Code.

API VS Code представляє асинхронні операції з “обіцянками”. З розширень можна повернути будь-який тип “обіцянок”, як ES6, WinJS, A + тощо.

Незалежність від конкретної бібліотеки з “обіцянкою” виражається в API за допомогою типу-інтерфейсу Thenable. Thenable інтерфейс повертає об’єкт з методом then. Thenable інтерфейс повертається у кожному провайдері розширення.

У більшості випадків використання обіцянок є необов'язковим, і коли VS Code викликає розширення, він може явно повернути результат. Коли використання обіцянки є необов'язковим, API це вказує, повертаючи обидва типи:

provideNumber(): number | Thenable<number>.

Часто операції VS Code мають можливість змінюватися, перш ніж операції можуть закінчитися. Наприклад, починається обчислення модулю розумного доповнення і користувач продовжує вводити текст, в результаті чого результат цієї операції стає застарілим.

API, які реалізовують таку поведінку, отримують токен CancellationToken, за яким розробник може перевірити скасування (isCancellationRequested) або отримувати сповіщення, коли відбувається скасування (onCancellationRequested). Токен скасування зазвичай є останнім параметром виклику функції та є необов’язковим.

API VS Code використовує шаблон звільнення ресурсів (Disposable pattern), отриманих з VS Code. Це стосується прослуховування подій, команд, взаємодії з інтерфейсом користувача та різними частинами мови.

Наприклад, функція setStatusBarMessage (значення: string) повертає Disposable об’єкт, який після виклику методу dispose видаляє повідомлення з панелі показу статусу.

Події в API VS Code є функціями, які підписуються на певну подію DOM. Функції, які повертаються після підписання, мають тип Disposable і можуть автоматично відписатися після виконання методу dispose. Приклад:

var subscription = fsWatcher.onDidDelete (listener);subscription.dispose ();

Назви подій мають on[Will|Did]VerbNoun? паттерн. Назва сигналізує, що подія відбудеться (onWill) або вже відбулася (onDid), що сталося (дієслово) і контекст (іменник), якщо це не є очевидним з контексту.

Прикладом VS Code API є window.onDidChangeActiveTextEditor, що відбувається після зміни активного текстового документа.

В даній роботі виконується підписка на:

- відкривання документу – onDidOpenTextDocument;
- зміна контенту документа – onDidChangeTextDocument;
- зміна активного документа – onDidChangeActiveTextEditor.

При виконанні будь-якої події із списку вказаного вище, у даному розширенніініціюється виконання лексичного аналізатора.

2.5. Публікація розширення

Для публікації розширення VS Code (включаючи розроблене), використовується модуль vsce (The Visual Studio Code Extension Manager). VSCE — це інструмент командного рядка, який використовується для публікації розширень на Extension Marketplace. Користувач також можете завантажувати розширення локально (як vsix файл).

Через проблеми безпеки, vsce не публікує розширення, що містять SVG зображення.

Visual Studio Code використовує служби Visual Studio Team Services для своїх служб Marketplace. Це означає, що через цю службу здійснюється аутентифікація, хостинг та управління розширеннями. VSCE публікує

розширення за допомогою особистих токенів доступу. Щоб опублікувати розширення, потрібно створити принаймні один токен доступу.

Для публікації розширення використовується команда vsce publish, для видалення пакету — vsce unpublish.

Для створення локального VSIX файлу використовується команда vsce package, яка поміщає створений файл у корінь проекту розширення.

Перед публікацією розширення потрібно вказати мінімальну підтримуючу версію VS Code у файлі маніфесту — ключ engines.vscode. Граматика значення співпадає з семантикою ведення версій пакету Node.js та підтримує квантифікатори
^, ~, *.В розробленому розширенні для публікації також вказані категорії, до яких відноситься дане розширення (рисунок 2.4).

Рисунок 2.4 — Маніфест розширення для публікації.

2.6. Архітектура середовища написання текстів

Для підтримки написання текстів з дотриманням функціональних вимог було вирішено розробити:

- мовний клієнт;
- мовний сервер;
- протокол мовного сервера.

Мовний клієнт повинен використовувати протокол мовного сервера для зв’язку з мовним сервером.

Мовний клієнт повинен мати наступні модулі:

- модуль доповнення тексту;
- модуль діагностики;
- модуль підказок.

Модуль діагностики повинен надавати користувачу інформацію про помилки вводу тексту запиту, наприклад невірний символ. Вхідними даними є природномовний текст. Вихідними даними є текст помилки, індекси рядків та колонок початку та кінця помилки, можливі автоматичні виправлення.

Модуль розумного доповнення повинен надавати інформацію про список можливих доповнень, які починаються з тексту, який був введеним користувачем, що забезпечить прискорення написання тексту. Вхідними даними є природномовнийтекст. Вихідними даними є список лексем.

Модуль підказок повинен надавати опис відповідної лексеми. Вхідними даними є природномовний текст, індекс наведеного на лексему курсору. Вихідними даними є опис лексеми.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗРОБКИ

3.1. Мовний клієнт

Мовний клієнт — текстовий редактор, призначений для створення й зміни текстових файлів через графічний інтерфейс користувача.

Мовний клієнт отримує повідомлення про зміну тексту користувача, активного документа, місцеположення каретки вводу та передає відповідну інформацію на мовний сервер по спеціально розробленому мовному протоколу.

Мовний клієнт отримує повідомлення від мовного сервера та надає оброблені дані користувачу через графічний інтерфейс користувача. Фактично з використанням мовного сервера на стороні клієнта залишається лише одна вимога—наявність стабільного швидкого доступу до мережі Інтернет.

Мовними клієнтами можуть бути нативні додатки, веб-застосунки, гібридні додатки (зазвичай це використання веб-застосунків всередині нативних додатків).

Зазвичай розробники мовних клієнтів надають можливість для створення розширень для їх продукту, та викладення їх на маркет розширень. Реалізація мовних розширень залежить від операційної системи та архітектури мовного клієнта, наприклад, розширення для Microsoft Outlook для Windows використовуютьCOM технологію.

В даній роботі в якості мовного клієнта вибраний VS Code, створюючи розширення для нього, архітектура якого наведена на рисунку 3.1.

Реалізований мовний клієнт складається з провайдерів підтримки мови, процесорів документів, модулей для ASN кодування і декодування, обгортки над TCP та процесора TCP повідомлень.

Задля мінімізації часових витрат написання текстів та забезпечення мовної підтримки було розроблено наступні мовні провайдери:

- модуль доповнення тексту;
- модуль діагностики;
- модуль підказок.

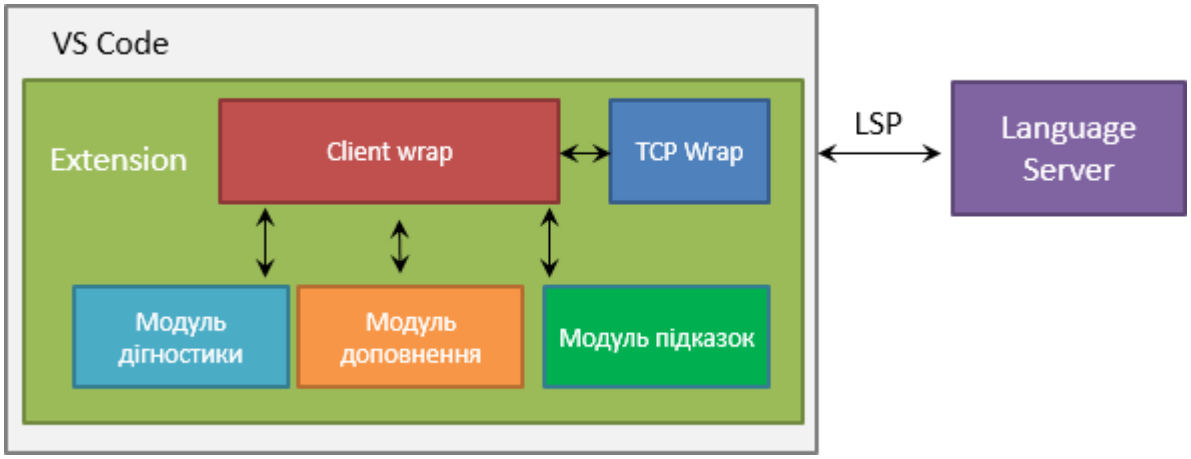

Рисунок 3.1 — Архітектура мовного клієнта.

Модулі діагностики, доповнення та підказок взаємодіють з відповідним процесором документу через посилання на vscode.TextDocument. Процесори документів зберігаються у обгортці для роботи з клієнтом (Client wrap), це необхідно для забезпечення одночасної роботи з декількома відкритими документами у Visual Studio Code.

Процесор TCP повідомлень викликає відповідні методи обгортки над клієнтомв залежності від типу повідомлення.

Оскільки програмний продукт повинен мати можливість зміни активної локалізації та збереження поточних налаштувань було вирішено розробити менеджер налаштувань та менеджер локалізації.

3.2. Модуль діагностики

Модуль діагностики надає список помилок (проблем) у тексті. Модуль діагностики отримує необхідні дані для відображення від мовного сервера.

Приклад роботи модуля діагностики показаний на рисунку 3.2.

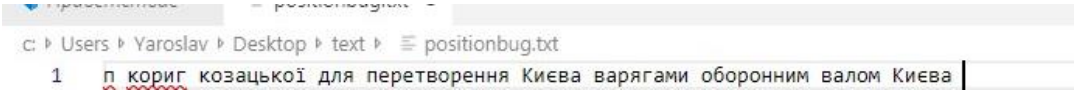

Рисунок 3.2 — Робота модуля діагностики.

На відміну від інших модулей, які є інструментальними засобами, модуль діагностики не імплементує провайдер, він підписує до контексту VS Code колекцію vscode.DiagnosticCollection. Після чого в колекцію неявно додається об’єкт Diagnostic, який вміщає інформацію про тип діагностики, повідомлення діагностики та діапазон — номери рядків та колонок тексту.

Типи підтримуючих типів діагностики на стороні клієнта:

- DiagnosticSeverity.Error;
- DiagnosticSeverity.Warning;
- DiagnosticSeverity.Information;
- DiagnosticSeverity.Hint.

Загальна кількість помилок модуля діагностики надається в графічному інтерфейсі VS Code у статус панелі.

Для видалення помилок тексту використовується видалення відповідного елементу Diagnostic з колекції помилок.

Реалізований модуль надає можливість виправляти помилки автоматично за допомогою “Code Actions” функціоналу, а саме змінювати неправильно написані слова на найбільш імовірні, які надаються мовним сервером.

Якщо дії доступні, поруч із помилкою чи попередженням з’являється

лампочка. Коли користувач натискає на лампочку, з’являється список доступних дій з текстом. Для цього імплементується CodeActionProvider інтерфейс, а саме provideCodeActions метод.

3.3. Модуль розумного доповнення

Модуль розумного доповнення надає інформацію про контекстно-залежні пропозиції для користувача. Модуль діагностики отримує необхідні дані для відображення від мовного сервера, надсилаючи йому місцеположення каретки вводу.

Для реєстрації модуля у розширенні, що має виділений сервер мови потрібно у методі ініціалізації з’єднання повернути об’єкт, який має у ключі capabilities ключ completionProvider з значенням resolveProvider:true.

Для реєстрації модуля у розширення з явною імплементацією провайдерів потрібно імплементувати інтерфейс vscode.CompletionItemProvider, а саме метод provideCompletionItems, що і використовується у даній реалізації мовного клієнта.

При додаванні модуля до контексту розширення додатково були вказані тригер-символи “*” для запуску модуля.

Кожного разу коли користувач вводить букви або тригер-символи, викликається метод provideCompletionItems, який повертає “обіцянку”, значенням якого є масив vscode.CompletionItem. CompletionItem складається з назви, типу, тексту по якому відбувається фільтрація автоматично середовищем VS Code, тексту який вставляється до редактора документу, документації.

В даній роботі у методі provideCompletionItems повертається масив CompletionItem, де поля “insertText”, “filterText”, “documentation” та “label” встановлюються значеннями, отриманих від мовного сервера. Для поля “documentation” використовується об’єкти типу MarkdownString, який є реалізацією синтаксису MarkDownSyntax (полегшеного синтаксису розмітки даних) та мови розмітки Setext.

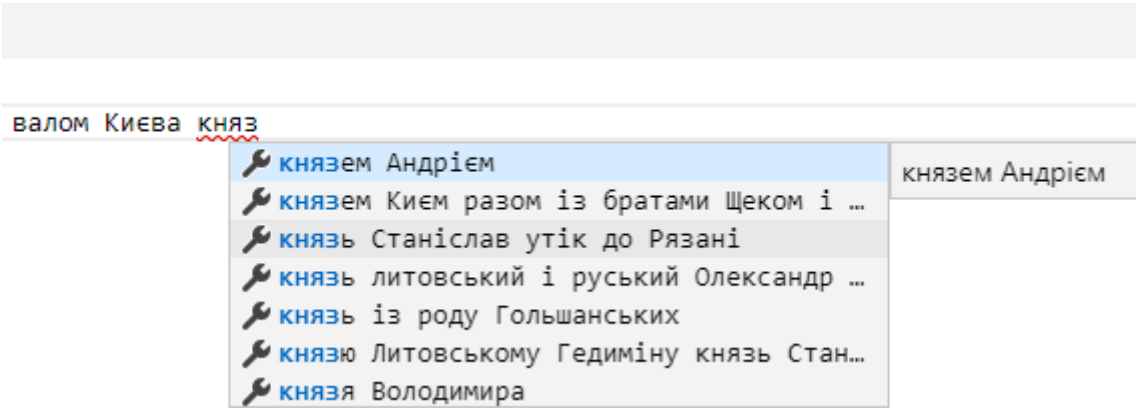

Рисунок 3.3 — Робота модуля розумного доповнення.

Модуль розумного доповнення дозволяє значно прискорити час вводу лексеми. Чим більша довжина лексеми, тим більше час, який модуль розумного доповнення дозволяє зменшити.

3.4. Модуль підказки

Модуль підказки надає додаткову інформацію про лексему або помилку модуля діагностики при наведенні курсору маніпулятора типу “миша” на відповіднулексему або помилку.

Повідомлення, яке надається модулем діагностики, реалізується автоматично засобами VS Code, для цього лише необхідно вказати правильний діапазон початкового та кінцевого індексів помилки тексту.

Повідомлення іншого типу, розраховується вручну, за допомогою масиву лексем (токенів) запиту. З масиву лексем, береться та лексема, індекси якої вміщають в себе індекс наведеного курсора. Для цього було створений клас HoverProvider, який імплементує інтерфейс HoverProvider, а саме реалізацію методу provideHover().

3.5. Людино-машинна взаємодія

Задля забезпечення максимальної продуктивності та максимальної швидкості програмного застосунку слід підтримувати на достатньо-високому рівні коефіцієнт людино-машинної взаємодії.

Людино-машинна взаємодія [23-24] являє собою якісну характеристику взаємодії між користувачем та комп'ютером. Це може бути взаємодія як і на рівні користувацького інтерфейсу (GUI), так і на апаратному рівні.

Критерії людино-машинної взаємодії діляться на:

- швидкість роботи з інтерфейсом;
- напруженість праці;
- кількість помилок під час роботи;
- швидкість навчання;
- продуктивність роботи.

Швидкість роботи з інтерфейсом залежить від:

- витраченого часу на аналіз вихідної інформації;
- витраченого часу на розумову діяльність;
- витраченого часу на механічну дію;
- витраченого часу зворотний зв'язок від системи.Тривалість розумової діяльності залежить від:

- формування мети діяльності над системою;
- визначення послідовності команд;
- виконання певної сукупності команд;
- оцінки отриманого результату;
- аналізу стану системи;
- інтерпретації поточного стану системи.

Прямим чином програмна система не може покращити швидкість формування мети діяльності користувача, проте вона може значно покращити визначення послідовності команд та швидкість їх виконання.

Задля цього, усі дії, які може робити користувач над системою, були зібрані в меню команд VS Code. Перевагою цього є те, що користувач має змогу бачити повний перелік дій, команд в одному меню. Недоліком є те, що крім команд даного розширення, користувачу доступні усі команди усіх встановлених розширень для VS Code, навіть ті, які є у виключеному режимі, це є побічний ефект того, що усі розширення виконуються в процесі спільного хоста. Тому, якщо бажана дія користувачем не виконувалась недавно або не виконувалась взагалі, ці команди будуть відображатись останніми у списку команд.

Тривалість механічних дій користувача в даному розширенні повністю залежить від VS Code. Наприклад, VS Code дозволяє при відкритті вікна вводу команд обрати команду за допомогою клавіатури (використання стрілок для вибору команд та “Enter” для підтвердження дії), що прискорює в 3-5 раз час, у порівнянні зманіпулятором типу “миша”.

Також, починаючи з версії 1.21.1 VS Code, користувач має змогу налаштувати місце відкриття вікна вводу команд.

Слід зазначити, що у різних версіях VS Code комбінації клавіш по замовчуванню можуть відрізнятися один від одного, тому рекомендовано переназначити комбінації на інші.

Для того, щоб показати, що середовище розробки реагує на вхідні дані та виконує певну операцію, VS Code показує деякий індикатор виконання цього процесу. Наприклад, на етапі завантаження даних з модуля підказок, VS Code може показати вікно з написом “Завантаження” та надає інформацію користувачу про те, що реагує на його дії.

Найбільшим недоліком даного розширення та розширень для VS Code взагалі є те, що оскільки розширення працюють у окремому процесі спільного хосту та завантажуються якомога пізніше, до тих пір, поки завантаження успішно активується та виконає функцію активації, користувач, при спробі виконати певну команду, наприклад отримати підказку чи використати інші можливості, які забезпечує розширення, отримає помилку (рисунок 3.4) або

взагалі нічого не отримає.

Тому дане розширення не забезпечує необхідну тривалість зворотнього зв’язку системи на етапі завантаження середовища розробки, і користувачу необхідно дочекатися повного завантаження розширення.

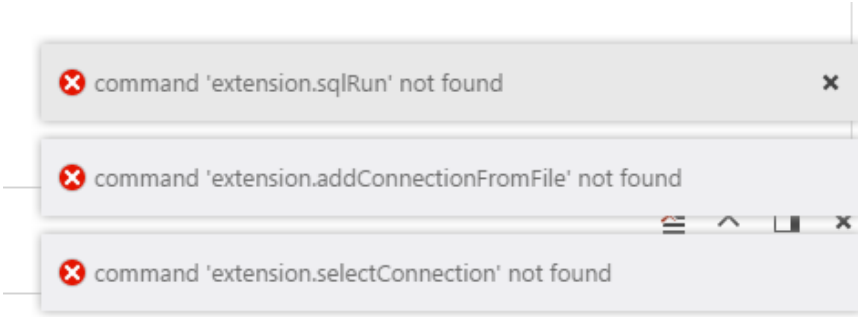

Рисунок 3.4 — Помилка при виборі команди.

Повне завантаження розширення може займати 5-20 секунд в залежності відпродуктивності процесора, пріоритету процесу та інших причин.

Великий показник людино-машинної взаємодії визначає невелику кількістьлюдських помилок.

Людська помилка — діяльність користувача, що не співпадає з його безпосередньою метою діяльності.

Людські помилки при написанні та використанні програмного забезпеченняділяться на наступні типи:

- помилки, через хибне або недостатнє знання предметної області;
- описки;
- помилки, викликані хибним аналізом показань системи;
- моторні помилки.

Помилки, викликані хибним знанням про предметну область, легко вирішуються за допомогою навчання користувача в процесі користування з системою або в процесі вивчання довідкового матеріалу. Розширення забезпечує швидке навчання за допомогою однотипності та групування дій.

Описки залежать від використовуваного маніпулятора та від тривалості інтелектуальної роботи користувача. В даному розширенні при виникненні

помилок автоматично спрацьовує модуль діагностики, який забезпечує правильність вводу запиту, і користувач завжди має доступ до списку та кількості помилок.

Помилки, викликані хибним аналізом системи характеризують стан, при якому бачення активного стану систему користувача не співпадає з реальним станомсистеми.

Прикладом є вибір вже активної мови в меню налаштувань, тому для активної мови надається відповідне додаткове повідомлення в меню.

Також слід зазначити, що для підвищення показника людино–машинної взамодії в даному розширенні використовується збереження поточних налаштувань до глобальних налаштувань, що забезпечує автоматизацію дій та зменшення фізичних дій користувача.

3.6. Мовний сервер

Мовний сервер (рисунок 3.5) забезпечує функціональну підтримку підсистеми обробки текстів. Завдяки цьому, зменшується навантаження на мовні клієнти, створюється окремий модуль, що надає однаковий функціонал різним мовним клієнтам.

Мовний сервер обмінюється повідомленнями з мовними клієнтами за допомогою протоколу мовного сервера.

Мовний сервер підтримує одночасну роботу декілька мовних клієнтів та підтримує одночасну обробку декілька відкритих документів на клієнтську сесію. Для цього створена підтримка на рівні протоколу мовного сервера та на всіх модулях мовного сервера. Усі повідомлення між модулями сервера та повідомлення у протоколі мовного сервера мають посилання на порядковий номер документа та сесії користувача.

Мовний сервер складається з трьох збірок:

- DocumentAnalyzer.dll – бібліотека для аналізу текстів та забезпечення підтримки природномовних текстів;
- Server.exe – консольний додаток, який обмінюється повідомленнями з клієнтом та використовує DocumentAnalyzer;
- Base.dll – бібліотека, яка надає базові типи для Server.exe та DocumentAnalyzer.

Було вирішено розробити наступні базові модулі для сервера у DocumentAnalyzer.dll:

- лексичний аналізатор;
- синтаксичний аналізатор;
- адаптер для роботи з словником;
- модуль забезпечення мовної підтримки.

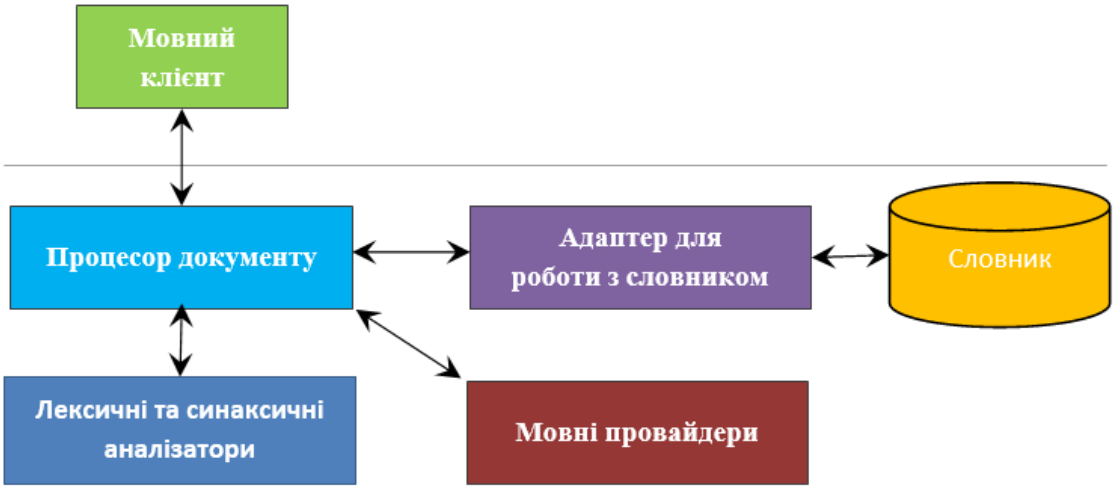

Рисунок 3.5 — Архітектура мовного сервера.

Server.exe (рисунок 3.6) складається з ID генератору для операцій, TCP Listener, ClientConnector (обробка TCP повідомлень), ASN кодерів (ASN Encoder, ASN Decoder), eClient (обгортки для роботи над клієнтом), TCP Message Wrap, та допоміжних модулей.

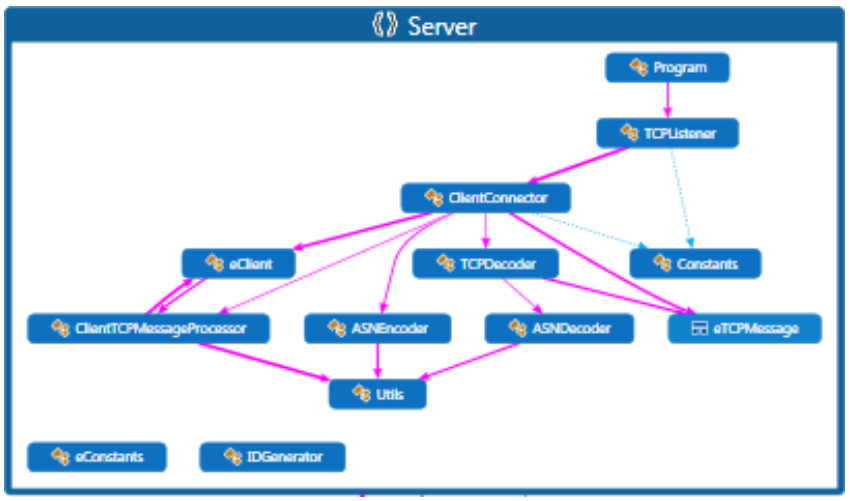

Рисунок 3.6 — Діаграма зв’язків Server.exe.

DocumentAnalyzer.exe (рисунок 3.7) складається з TokenParser (лексичного аналізатора), DiagnosticProvider (провайдер надання діагностики для тексту), DocProcessor (обгортка над документом), SpellCorrector (провайдер надання можливих дій для діагностики), DocClient (обгортка над клієнтом), класів операцій для надання мовної підтримки та допоміжних модулей.

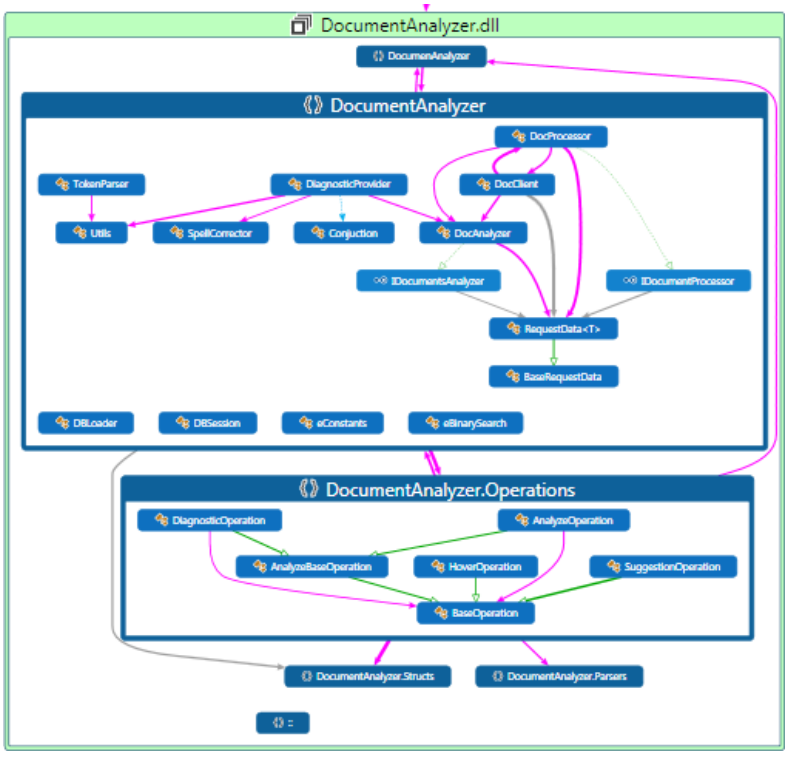

Рисунок 3.7 — Діаграма зв’язків DocumentAnalyzer.exe.

TCP Listener підтримує можливість зв’язку з декількома клієнтами. TCP

Message Wrap отримує дані від ASN Decoder та являє собою абстракцію для розбиття повідомлень на тип повідомлення (перша частина повідомлення) та корисне навантаження повідомлення (друга частина повідомлення).

Реалізований мовний сервер має наступні недоліки:

- відсутність балансеру навантаження;
- відсутність можливості перезапуску після падіння;
- відсутність збору статистики та запису лог-інформації;
- відсутність захищеного каналу передачі даних.

3.7. Протокол мовного сервера

Протокол мовного сервера – LSP (Language Server Protocol), являє собою контракт, набір правил, який підтримується мовним сервером і використовується клієнтами для користування послугами сервера.

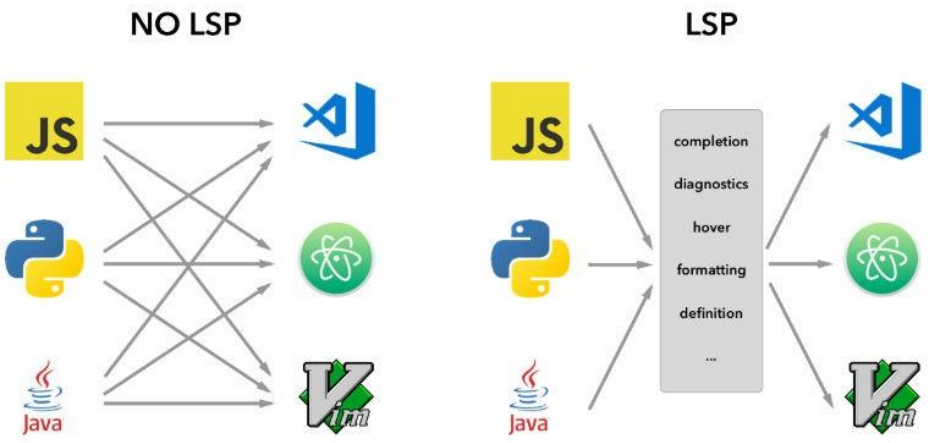

Рисунок 3.8 — Перевага використання LSP.

Єдиний контракт забезпечує стандартизацію повідомлень між множиною мовних клієнтів та множиною мовних серверів (рисунок 3.8).

Існує велика кількість різних видів протоколів та способів кодування даних.

Наприклад, широко використовується у готових реалізаціях мовних серверів протокол виклику віддалених процедур JSON-RPC [10-11] (рисунок 3.9), проте він має ряд недоліків, а саме: великий обсяг пакету через використання JSON, необхідність підтримування єдиного називання методів між клієнтами і серверамита інші. Перевагою є невелика кількість типів даних та властивостей, повідомлення має зручний вигляд. Приклад JSON RPC повідомлення: --> {"jsonrpc": "2.0",

"method": "update", "params": [1,2,3,4,5]}. JSON-RPC в якост транспортного протоколу використовує HTTP протокол.

```
--> {"method": "echo", "params": ["Hello JSON-RPC"], "id":1}
<-- {"result": "Hello JSON-RPC", "error": null, "id":1}
```

Рисунок 3.9 — Приклад JSON RPC протоколу.

Для зменшення обсягів TCP пакету, збільшення корисної пропускної здатності каналу було вирішено створити новий протокол. В якості транспортного проколу обрано TCP протокол, тому що він має наступні переваги на відміну від UDP (User Datagram Protocol):

- гарантує доставку даних до призначення;
- забезпечує широкі механізми перевірки помилок;
- гарантує послідовність даних;
- можливість повторної передачі втрачених пакетів;
- орієнтованість на використання з’єднання.

Пакет повторно передається лише протягом двох подій: коли відправник отримує три дублікати підтвердження (ACK) або після закінчення таймера retransmission timer, припускаючи, що він втрачений по шляху проходження.

В якості транспортного протоколу можна було би використати UDP протокол оскільки він швидше, простіше та ефективніше, що досить важливо для надання швидкої мовної підтримки текстів, ніж TCP, але для цього потрібно було розроблятидодатковий функціонал для перевірки помилок.

В якості виду кодування даних вибрано DER [16-17] кодування. DER

імплементує ASN.1 Abstract Syntax Notation One синтаксис та задовільняє X.690 стандарт ASN.1 представлення даних при їх передачі.

Кожен елемент даних кодується наступною послідовністю: ідентифікатор типу, опис довжини, фактичні елементи даних та, де необхідно, маркер кінця вмісту
(рисунок 3.10).

Identifier octets *Type*	Length octets *Length*	Contents octets *Value*

Рисунок 3.10 — Кодування даних протоколу.

Структура ідентифікатора показана на рисунку 3.11

Octet 1								Octet 2 onwards							
8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Tag class		P/C	Tag number (0–30)					N/A							
			31					More	Tag number						

Рисунок 3.11 — Кодування типу.

Біти 8 і 7 визначають клас даних, в даній роботі вони дорівнюють 0 (типи, які визначені тільки в X.690 і мають однаковий сенс у всіх додатках).

Біт 6 показує чи є тип простим (дорівнює 0) або складеним (дорівнює 1). Розроблений протокол використовує лише прості типи даних. Складені типи серіалізуються та десеріалізуються у послідовності на рівні процесорів TCP повідомлень.

Біти 5 - 1 означають тег даних. Розроблений протокол використовує наступні типи даних ASN.1:

- INTEGER = 0x2 (число);
- UTF8String = 0xC (рядок);
- SEQUENCE = 0x10 (послідовність);

- OCTET_STRING = 0x4 (масив байтів).

В реалізованому протоколі складені типи кодуються наступним чином:

- vscode.Range – SEQUENCE {Start, End};
- vscode.Position – SEQUENCE {Line, Character};
- Diagnostic – SEQUENCE {Range, Message, Severity, PossibleCorrection};
- BaseRequestData – SEQUENCE {ReqId, Data}.

У випадку, якщо довжина блоку даних для збереження відомі та довжина не перевищує 127 октетів (байт), то вона просто записується у відповідний октет довжини, біт 8 першого октету встановлюється в 0. Така форма представлення октетів називається короткою формою (DEFINITE_SHORT).

Загальний алгоритм кодування довжин поданий на рисунку 3.12

First length octet

Form	Bits							
	8	7	6	5	4	3	2	1
Definite, short	0	Length (0–127)						
Indefinite	1	0						
Definite, long	1	Number of following octets (1–126)						
Reserved	1	127						

Рисунок 3.12 — Кодування довжин.

У випадку, якщо довжина блоку даних для збереження відомі та довжина більше 127 байт, то: в біти другого і наступних октетів записується значення довжини блоку закодованої інформації; в перший октет записується кількість додаткових блоків довжини, починаючи з другого; біт 8 першого октету встановлюється в 1. Така форма представлення октетів називається довгою формою (DEFINITE_LONG).

Розроблений протокол використовує лише наступні види довжин ASN.1:

- DEFINITE_SHORT;
- DEFINITE_LONG.

В реалізованому протоколі пусті списки надсилаються за допомогою типу SEQUENCE з довжиною 0.

Протокол використовує надає наступні TCP повідомлення типу “запит клієнта”:

- повідомлення сесії клієнта (створення та знищення);
- повідомлення аналізу первинного тексту;
- повідомлення сесії документу (створення та знищення);
- повідомлення діагностики документу;
- повідомлення розумного доповнення.

Кожне повідомлення типу “запит клієнта” має відповідне повідомлення типу “відповідь сервера”.

Протокол легко масштабується та доповнюється, можлива реалізація приведення протоколу старих версій клієнта до нових версій сервера.

Розмір повідомлень розробленого протоколу в середньому у 15-30 разів менше у порівнянні з JSON RPC протоколом (рисунок 3.13), залежить від формату кодування символів (UTF-8, UTF-16…) та способу кодування даних.

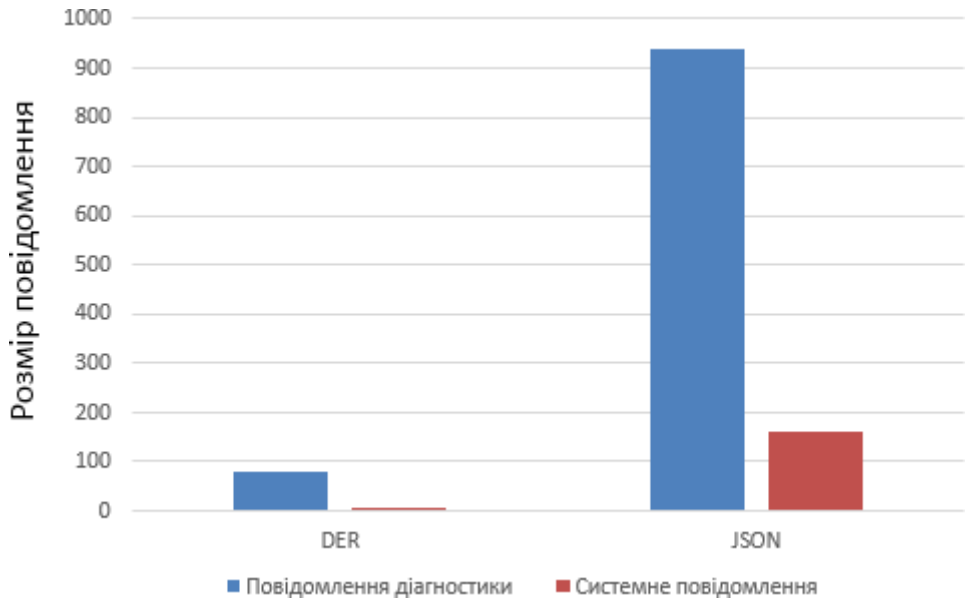

Рисунок 3.13 — Порівняння розробленого протоколу відносно JSON RPC.

3.8. Лексичний аналізатор

Лексичний аналіз (розпізнавання вхідної послідовності на масив лексем) є першою фазою аналізу тексту.

Програма, яка здійснює лексичний аналіз, називається лексичним аналізатором або сканером.

Реалізований лексичний аналізатор підтримує наступні типи лексем:

- WHITESPACE;
- NEW_LINE;
- WORD;
- SEPARATOR;
- NUMBER;
- UNKNOWN (у випадку, якщо вхідна лексема містить непідтримуючі символи або не задовільняє правилам лексемотворення).

Для типу SEPARATOR підтримуються наступні символи: '.', '!', '?', ',', '-', ';'.

Лексичний аналізатор підтримує апостроф ('’', '\'') та римську систему числення. Підтримка римської системи числення відбувається наступним чином: якщо поточна лексема має тимчасовий тип WORD і якщо вона складається лише з відповідних символів римської системи числення ('I', 'V', 'X', 'L', 'C', 'D', 'M'), відповідна лексема змінює свій тип на NUMBER.

Перед проведення лексичного аналізу перший символ тексту перевіряється на збіжність с маркером порядку байтів (Byte-order mark, BOM) та видаляється з аналізу, оскільки в такому випадку не будуть збігатися діапазони значень лексем (наодиницю до першого переносу строки), що призведе до неправильної роботи модулю діагностики клієнта. Після чого сам маркер порядку байтів використовується за призначенням.

Після роботи лексичного аналізатора кожна лексема містить інформацію про: контент, тип, діапазон.

На наступному етапі відбувається розпізнавання речень по спеціальним

символам ('.', '!', '?'). Після роботи аналізатора речень на виході отримується об’єкт типу “Document”, який має значення, діапазон та масив речень. Об’єкт типу “Sentence” має значення, діапазон та посилання на документ.

3.9. Виконання запитів мовного клієнта

Для надання підтримки одночасної роботи з декількома клієнтами, мовний сервер зберігає по одній обгортці над клієнтом, відповідно до кожного TCP клієнта.

Для надання підтримки одночасної роботи з декількома документами, мовнийсервер зберігає в обгортці над клієнтом множину процесорів документів.

Процесор документу зберігає множину поточних операцій від мовного клієнта над документом, операції виконуються послідно за допомогою черги операцій.

Мовний сервер виконує наступні типи операцій:

- BaseOperation — абстрактний базовий клас для операцій;
- HoverOperation — операція забезпечення підказки;
- SuggestionOperation — операція забезпечення розумного доповнення;
- AnalyzeBaseOperation — абстрактний базовий клас для операцій аналізу;
- DiagnosticOperation — операція аналізу поточного тексту;
- AnalyzeOperation — операція аналізу первинного тексту.

Кожна операція зберігає дані про порядковий номер запиту та підтримує можливість її скасування під час виконання.

3.10. Забезпечення підтримки текстів

Реалізований мовний сервер підтримує наступну функціональність:

- доповнення тексту без врахування флексій;
- доповнення тексту врахуванням флексій;
- діагностика тексту (перевірка орфографії).

Для можливості надання даної мовної функціональності, використовується граматичний словник mphdict [19].

mphdict вільно розповсюджується та його реєстр складає 261499 слів (на вересень 2016 року) і є найбільшим словником такого типу. Він має опис внутрішньої структури бази даних словника. mphdict надає вільний доступ до словника словозмінної класифікації. Усі бази даних словника доступні під ліцензією Open Database License і надаються для використання, правок та змін відповідно до потреб користувачів.

Проект містить SQLite бази даних граматичних словників, за основу був взятий "Орфографічний словник української мови" С. Головащука, створений на основі 4-го видання "Українського правопису" (1993).

Для можливості використання словника були додані наступні Nuget-пакети: Microsoft.Data.Sqlite.Core, Microsoft.Data.Sqlite, SQLitePCLRaw.core, SQLitePCLRaw.provider.e_sqlite3.net45.

mphdict вмішає наступні словники:

- граматичний словник (рисунок 3.14);
- словник синонімів;
- етимологічний словник (містить відомості про етимологію слів).

Граматичний словник вміщає наступні основні таблиці:

- nom — таблиця реєстрових одиниць словника;
- indents — таблиця словозмінних класів;
- flexes — таблиця квазіфлексій словозмінних класів;
- parts — таблиця частин мови;

- accents_class — таблиця класів наголосів (акцентуаційних класів);
- gr — таблиця граматичних категорій.

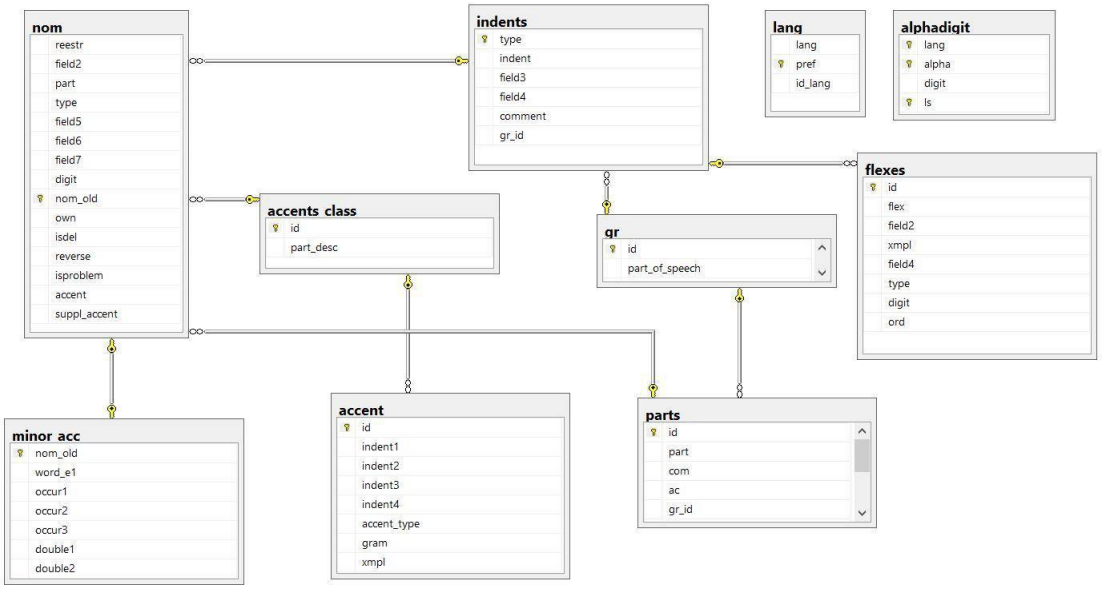

Рисунок 3.14 — Структура бази даних граматичного словника.

В даній роботі використовуються наступні таблиці та відповідні в них колонки: nom (таблиця 3.1), indents (таблиця 3.2), flexes (таблиця 3.3), parts (таблиця 3.4).

Таблиця 3.1. Структура колонки nom

Name	Type	Comments
reestr	nvarchar(255)	Реєстрове слово
part	smallint	Частина мови (табл. [parts])
type	smallint	Номер словозмінного класу (табл. [indents])

Таблиця 3.2. Структура колонки indents

Name	Type	Comments
type	smallint	унікальний ідентифікатор запису (номерсловозмінного класу)

indent	integer	кількість літер, які потрібно відрізати від кінцяслова, щоб залишилась квазіоснова
gr_id	smallint	ідентифікатор гр.кат. з табл. [gr]

Таблиця 3.3. Структура колонки flexes

Name	Type	Comments
id	integer	унікальний ідентифікатор запису
flex	nvarchar(255)	квазіфлексія
field2	integer	номер граматичної категорії
type	smallint	номер словозмінного класу

Таблиця 3.4. Структура колонки parts

Name	Type	Comments
id	smallint	унікальний ідентифікатор запису, код частини мови
gr_id	smallint	ідентифікатор гр.кат. з табл. [gr]
Name	Type	Comments
rid	integer	рід
mnozh	integer	

Номер граматичної категорії використовуються для знаходження інформації про відмінок, кількість та рід.

Ідентифікатор граматичної категорії використовується для знаходження частини мови.

Для оптимізації аналізу вхідного тексту, а саме пошуку словозмінної форми, було проведено створення спеціального методу у словнику mphdict, який під час першого запуску сервера та доступу до бази, виконує прохід по всім словам та формує словник флексій до цієї словоформи. Це призвело до відмови від використання неоптимізованих алгоритмів пошуку флексій у mphdict, яке

включає SQL операції LINQ-to-Entities через Entity Framework та пришвидшило пошук слова приблизно у 30 - 50 раз.

Словник флексій формується наступним чином:

- береться запис з таблиці «nom»;
- визначається основа слова;
- видаляється у слова наголос;
- робиться вибірка записів з таблиці «flexes», що мають словозмінний клас.

В даній роботі використовується лише граматичний словник, тому для прискорення запуску сервера було видалено завантаження інших словників. З лексичного словника використовуються лише таблиці nom, flexes, parts, тому завантаження інших таблиць також було видалено.

Для роботи з базою даних створений адаптер для роботи з даними словника. Під час запуску сервера, адаптер проходить по всьому словнику та зберігає інформацію про слово, при цьому роблячи маршалинг даних, формуючи загальний список слів. Після чого усі слова сортуються по постійній частині слова для

оптимізації пошуку слова. Первинна обробка даних бази займає приблизно 10-20 секна процесорі i7-5500u.

Перевірка орфографії складається з наступних етапів:

- пошук слів з вхідного тексту у словнику;
- пошук можливих корекцій для невідомих слів;
- додаткова дія для корекції з урахуванням морфології.

Під час викликів аналізаторів тексту для пошуку слова з даних словника, адаптер повертає слово з кешу (між усіма сесіями), а при його відсутності в кеші, з загального списку.

Пошук слова починається з першого слова, яке починається з тієї ж букви, що і вхідне, відбувається прохід по всім лексіям, та закінчується у разі неуспішного пошуку останнім слово, яке починається з тієї ж букви, що і вхідне. При цьому потрібно обробляти спеціальні випадки слів, коли постійна частина

слова пуста, наприклад: я-мені, він-йому, вона-їй.

Кожне слово з бази надає наступну інформацію: тип, постійну частину слова, рід, масив флексій, додаткову інформацію, яка залежить від типу.

Тип слова є значенням з наступного списку: NOUN, ADJECTIVE, NUMERATOR, PRONOUN, VERB, PARTICIPLE, GERUND, ADVERB, CONJUNCTION, PREPOSITION, FRACTION.

Додаткова інформація містить інформацію про: тип числівника та тип дієсловадля відповідних типів слова.

Теперішня реалізація мовного сервера не підтримує дієслова які мають одночасно доконаний та недоконаний вид — двовидові дієслова, наприклад: мовити, ранити, женити.

Флексія містить інформацію про: закінчення змінних морфем-афіксів, відмінок, число, рід та час.

Пошук можливих корекцій складається з наступних частин:

- механізм селекції;
- модель кандидатів;
- мовна модель P(c);
- модель помилок P(w|c).

Механізм селекції — обрання найбільш ймовірних кандидатів.

Модель кандидатів — множина можливих змін невідомого слова, складаєтьсяз наступних підмножин, де n — довжина слова:

- множина видалень (n елементів);
- множина вставок (34 * (n + 1) елементів);
- множина переміщень (n - 1 елементів);
- множина заміщень (34 * n елементів).

Усього модель кандидатів складається з 70 * n + 34 елементів. Після створення множини кандидатів відбувається пошук кожного потенційного кандидата условнику.

Мовна модель P(c) формує ймовірність використання слова “c” у тексті, наприклад, слово “та” зустрічається частіше, ніж слово “там”.

Модель помилок P(w|c) формує ймовірність, що для слова “c” автором помилково було написано слово “w”.

Якщо невідоме слово зустрічається в тексті, що надав користувач для аналізу перед роботою з документами, це слово провайдером діагностики не враховується.

Провайдер діагностики підтримує наступні типи повідомлень діагностики:

- ERROR;
- WARNING.

Провайдер доповнень використовує відсортований по алфавіту масив суфіксних закінчень для кожного слова у реченні, який формується при первинному аналізі тексту. Масив суфіксних закінчень включає лише токени типу “Word” та “Whitespace”.

Після отримання повідомлення від клієнта про запит на отримання закінчень, провайдер доповнень отримує слово, чиї координати вміщають позицію курсору. Після чого за допомогою бінарного пошуку знаходиться підмасив суфіксів.

Якщо результат пустий та виконуються необхідні умови створюються суфікси з використанням додаткових даних. В даній роботі необхідною умовою є використання прикметника та іменника, а додатковими даними є список флексій.

Вихідні дані відсортовуються за частотою використання, відокремлюються перші доповнення, після чого вони посилаються на мовний клієнт.

РОЗДІЛ 4.
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Структурно-функціональний аналіз технологічного процесу

Розробка та вживання ефективних заходів запобігання аварійним і травмонебезпечних ситуаціям можливі лише при завчасному виявлення тих небезпек, з яких починаються процеси їх формування. Оскільки небезпечні умови не завжди завчасно можна виявити, а для вивчення небезпечних дій іноді потрібно багато часу, щоб зібрати статичний матеріал, то і методи виявлення цих небезпек повинні бути відповідно диференційовані (табл. 5.1).

Таблиця 4.1. - Моделі формування та виникнення травмонебезпечних і аварійних ситуацій

Вид робіт, виробничий підрозділ, робоче місце, виробниче обладнання, склад агрегату	Виробнича небезпека			Можливі наслідки	Заходи запобігання небезпечним ситуаціям
	Небезпечна умова (НУ)	Небезпечна дія (НД)	Небезпечна ситуація (НС)		
Виконання робіт із електрообладнанням	Не вимкнено живлення. Відсутність заземлення.	Нехтування правилами ТБ	Ураження струмом	Травма (Т)	Проведення повторного інструктажу з ТБ. Розробка нових способів захисту. Встановлення заземлення.
НД НУ → НС → Т					

Відповідно до аналізу небезпечних умов, які існують у виробничому процесі виокремлено такі наступні за характером дії на працівника їх групи:

- характеризують стан або рівень небезпеки обладнання, які використовуються.
- сприяють виникненню технологічних помилок обслуговуючого персоналу впродовж виробничого процесу;
- створювати умови та можливість проникнення працівника в небезпечну зону;
- приводять до виникнення небезпечних дій (внаслідок низького рівня професійної підготовки працівників та організації навчання з охорони праці).

Моделі формування та виникнення травмонебезпечних і аварійних ситуацій в комп'ютерному кабінеті представлено у вигляді моделі формування та виникнення травмонебезпечних і аварійних ситуацій – табл. 5.1.

4.2. Розрахунок освітлення приміщення комп'ютерного кабінету

Освітленість виробничих приміщень може бути штучною і природною. Природне освітлення при правильному обладнанні найбільш сприятливе для людини. Основні вимоги для освітлення наступні:

- освітлення повинне бути достатнім для швидкого і легкого розпізнання об'єктів роботи;
- освітлення повинно бути рівномірне без різких тіней;
- джерело світла не повинно осліплювати працівника;
- рівень освітленості не повинен обмежуватись часом.

Природне освітлення забезпечується обладнанням вікон (бокове освітлення) фонарів і світильних покрить приміщень (верхнє освітлення). Природне освітлення нормується коефіцієнтом природної освітленості. Коефіцієнт природної освітленості – це процентне відношення фактичної освітленості Fв в будь-якій точці приміщення до освітленості Fн розсіяної світлом небозводу точки, яка лежить на відкритій місцевості. Розрахунок природного освітлення через бокові вікна по нормам освітленості ведеться для

самої дальньої від вікон точки, тобто знаходять мінімальне значення стик коефіцієнта природної освітленості:

$$e_{\min} = \frac{Fb}{F_н} \cdot 100. \qquad (4.1)$$

Значення коефіцієнта природної освітленості визначається не менше чим в п'яти точках. Значення коефіцієнта природної освітленості для сільськогосподарських виробничих приміщень в даному випадку ремонтній майстерні, беремо $e_{min} = 5\%.$

Розрахунок природного освітлення зводиться до визначення площі світлових променів.

Сумарну площу світлових променів $\sum F_o\,(м^2)$ по коефіцієнту природної освітленості для бокових променів визначаємо по формулі:

$$\sum F_o = \frac{F_н \cdot e_{\min} \cdot r_o \cdot K}{100 \cdot \tau \cdot Г_1}, \qquad (4.2)$$

де Fн – площа підлоги; e_{min} – величина мінімального коефіцієнта природного освітлення; τ – загальний коефіцієнт світловикористання віконного отвору із врахуванням його забруднення, τ = 0,25; ro – світлова характеристика вікна, ro – 9,5; Г1 – коефіцієнт, який враховує підвищення освітленості за рахунок світла, яке відбивається від стін і стелі, Г1 = 1,2; К – коефіцієнт, який враховує затінення вікон сусідніми приміщеннями і загорожею, К = 1.

$$\sum F_o = \frac{36 \cdot 0,5 \cdot 9,5 \cdot 1}{100 \cdot 0,25 \cdot 1,2} = 5,7 м^2.$$

Кількість світлових променів визначимо:

$$N = \frac{\sum F_o}{F_o}, \qquad (4.3)$$

де Fo – площа вікна згідно стандарту, м2.

$$N = \frac{5,7}{6} = 0,95.$$

Приймаємо кількість вікон – одне вікно.

При розрахунку природного освітлення найбільш поширеним і простим є метод світлового потоку. При цьому методі розраховуємо світловий потік $F_л$ (Лк), який повинна випромінювати кожна лампа (при заданій кількості ламп).

$$F_л = \frac{k \cdot S_п \cdot E}{n_л \cdot \eta \cdot r^2}, \qquad (4.4)$$

де k – коефіцієнт запасу, $k = 1,3$; $S_п$ – площа підлоги, м2; $S_п = 36$ м2. E – нормативна освітленість, Е = 300 Лк; $n_л$ – кількість встановлених ламп, $n_л$ =6 од; η – коефіцієнт використання світлового потоку, $\eta = 0,25$; r – коефіцієнт нерівномірності освітленості, $r = 0,545$.

Коефіцієнт запасу (K) враховує можливість забруднення світильників пилом, що залежить від характеру виробництва.

Розрахунок штучного освітлення починаємо із визначення висоти розташування світильника і їх кількості. Висоту $h_н$ (м) розташування світильників над робочим місцем знаходимо за формулою:

$$h_н = H - (h_1 + h_2), \qquad (4.5)$$

де H – висота приміщення, м; h_1 – віддаль від підлоги до освітлювальної поверхні, м; h_2 – віддаль від стелі до світильника, м.

$$h_н = 4,5 - (2,2 + 1,5) = 0,8 \text{ м}.$$

При симетричному розміщенні світильників по вершинах квадратів їх кількість визначається за формулою:

$$n_c = \frac{S_n}{l^2}, \qquad (4.6)$$

де l – віддаль між світильниками, м.

Підставивши значення отримаємо:

$$n_c = \frac{36}{9} = 4 \text{ од.}$$

Тоді світловий потік буде становити

$$F_л = \frac{1,3 \cdot 36 \cdot 300}{4 \cdot 0,25 \cdot 0,545} = 2576,2 \text{ Лк.}$$

При світловому потоці 2576,2 Лк для заданої лампи вибираємо тип і потужність.

Вибираємо тип лампи – люмінесцентну, потужністю 40Вт.

4.3. Безпека в надзвичайних ситуаціях

Забезпечення захисту населення і території у разі загрози і виникнення надзвичайних ситуацій є одним з найважливіших завдань держави.

Захист населення є системою загально-державних заходів, які реалізуються центральними і місцевими органами виконавчої влади, виконавчими органами влад, органами управління з питань надзвичайних ситуацій та цивільного захисту населення, підпорядкованими їм системами, та підприємств, що забезпечують виконання організаційних, інженерно – технічних, санітарно – гігієнічних, проти епідемічних та інших заходів у сфері запобігання та ліквідації наслідків надзвичайних ситуацій.

Загрози життєво важливих інтересів громадян, держави, суспільства поділяють на зовнішні та внутрішні, виконують під час надзвичайних ситуацій техногенного та природного характеру та воєнних конфліктах.

Принципи захисту випливають з основних положень Женевської конвенції щодо захисту жертв війни та додаткових протоколів до неї, можливого характеру воєнних дій, реальних можливостей держави щодо створення матеріальної бази захисту. З метою захисту населення, зменшення втрат та шкоди економіці в разі виникнення надзвичайних ситуацій має право проводитись спеціальний комплекс заходів.

Оповіщення та інформування, яке досягається завчасним створенням і підтримкою в постійній готовності загально-державної, територіальних та об’єктивних систем оповіщення населення.

ВИСНОВКИ

Розглянуто особливості природномовних текстів та інструментальне забезпечення для їх використання.

Створено мовний сервер, мовний клієнт (розширення для VS Code) для написання природномовних текстів на прикладі української мови з використанням найбільшого українського граматичного словника. Мовний сервер обмінюється повідомленнями з мовними клієнтами за допомогою розробленого протоколу мовного сервера. Використання реалізованого мовного клієнта значно спрощує час для написання та перевірки текстів українською мовою. Використання реалізованого мовного сервера та розробленого протоколу розширює можливості для надання мовної підтримки.

Було розглянуто особливості природномовних текстів та інструментальне забезпечення для їх використання. Кожна природна мова має свої морфологічні та синтаксичні особливості, які слід враховувати при створенні інструментів для написання природномовних текстівданою мовою.

Було вирішено створити розширення для середовища розробки Visual Studio Code для написання природномовних текстів та створення мовного сервера. В якості інструменту написання природномовних текстів було вирішено використати Visual Studio Code, розширення для якого складається з модулей, які є інструментальними засобами, а саме: модуль розумного доповнення, модуль підсвічування тексту, модуль проведення діагностики, модуль підказок.

Модулі взаємодіють між собою явно за допомогою інтерфейсів або неявно за допомогою виконання заімплементованих методів, безпосередньо ініційованими VS Code, фактично — це дії користувача з розробленим розширенням.

Мовним клієнтом, який є розширенням для VS Code виконується в окремому процесі спільного хосту, використовує особливості мови TypeScript, використовує події VS Code. Мовний клієнт використовує явну реєстрацію

провайдерів у методі активації розширення. Дане розширення забезпечує високий рівень людино-машинної взаємодії та задовільняє основним критеріям якісного інтерфейсу та зберігає поточні налаштування користувача до глобальних налаштувань.

Було проведено аналіз ідеї інтелектуального асистента редактора природномовних текстів сервісу та зроблено висновки:

Є можливість ринкової комерціалізації проекту, перспективи впровадження з огляду на потенційні групи клієнтів, бар’єри входження, стан конкуренції, конкурентоспроможність проекту. Доцільною є подальша імплементація проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Karen S. Jones. Natural language processing: a historical review // Cambridge: Computer Laboratory, University of Cambridge, 2001. P.22.

2. D'Ulizia, A., Ferri, F., Grifoni, P. (2013) "A Survey of Grammatical Inference Methods for Natural Language Learning", Artificial Intelligence Review, Vol. 36, No. 1, pp. 1–27.

3. Hiroki Arimura; Takeshi Shinohara; Setsuko Otsuki (2012). "Finding Minimal Generalizations for Unions of Pattern Languages and Its Application to Inductive Inference from Positive Data" (PDF). Proc. STACS 11. LNCS. 775. Springer. pp. 649-660.

4. Kishorjit, N., Vidya Raj RK., Nirmal Y., and Sivaji B. (2012) "Manipuri Morpheme Identification", Proceedings of the 3rd Workshop on South and Southeast Asian Natural Language Processing (SANLP), COLING 2012, Mumbai, December 2012. pp. 95–108.

5. Ljiljana Dolamic, Jacques Savoy. Stemming Approaches for East European Languages (англ.) // CLEF. 2010. 35.

6. Lovins, Julie Beth. Development of a Stemming Algorithm // Mechanical Translation and Computational Linguistics. 2010. Т. 11. pp. 164-174.

7. Mernik - Formal and Practical Aspects of Domain-Specific Languages, 2012. 217.

8. Замаруєва І. В. Модель лінгвістичної бази даних в системах автоматичної обробки природномовної текстової інформації / І. В. Замаруєва, В. Б. Толубко, Л. О. Литвиненко, О. Ю. Ніколаєвський // Інформатика та математичні методи в моделюванні. 2015. Т. 3. 264.

9. Winograd, Terry (2012). Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. Vol. 6, No. 1, pp. 22–27.

10. Roger C. Schank and Robert P. Abelson (2013). Scripts, plans, goals, and understanding: An inquiry into human knowledge structures. Vol. 1, No. 1, pp. 42–46.

11. Olaronke G. Iroju, Janet O. Olaleke, A Systematic Review in Natural

Language Processing in Healthcare // I.J. Information Technology and Computer Science. 2015. #8. P. 45/

12. Nirenburg Sergei and Raskin Victor. Ontological Semantics, 2010. 45.

13. Goldberg, Yoav (2016). A Primer on Neural Network Models for Natural Language Processing. Journal of Artificial Intelligence Research (2016) 345. 420.

14. Gardner A. 7 Benefits Of Business Process Automation. 2020. URL:http://blog.soliditech.com/blog/7-benefits-of-business-process-autom (дата звернення: 10.04.2023).

15. Global talent Lviv. 2019. URL: https://my.globaltalent.tv/event/gtl-10- 11-18 (дата звернення: 10.04.2023).

16. How Automatons Helped Predict the Future of Robotics. Inverse. January 22, 2016. URL: https://www.inverse.com/article/10494-how-automatons- helped-predict-the-future-of-robotics (дата звернення: 10.04.2023).

17. IBM Institute for Business Value. The evolution of process automation. 2019. URL: https://public.dhe.ibm.com/gbe03885usen/intelligent-automation_GBE03885USEN.pdf (дата звернення: 10.04.2023).

18. Extending Visual Studio Code. 2018. URL: https://code.visualstudio.com/docs/extensions/overview.

19. Yo Code Extension Generator, 2018. URL: https://code.visualstudio.com/docs/extensions/yocode.

20. A Common Protocol for Languages. Microsoft. 2016. 668.

21. Krill, Paul (2016). "Microsoft-backed Language Server Protocol strives for language, tools interoperability". InfoWorld. pp. 164-172.

22. Burton S. Kaliski Jr. A Layman's Guide to a Subset of ASN.1, BER, and DER // RSA Laboratories: Technical Note. 2012. 58.

23. Information technology ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), ITU-T X6.90, 07/2010. 148.

24. Lin, Huai-An. “Estimation of the Optimal Performance of ASN.1/BER Transfer Syntax”. ACM Computer Communication Review. July 15. 2022. 542.

25. Електронні словники/цифрові лексикографічні системи української мови (серія "Цифрове лексикографічне надбання України"), 2018 . URL: https://github.com/LinguisticAndInformationSystems/mphdict/wiki.

26. Fitts, Paul M. (2014). "The information capacity of the human motor system in controlling the amplitude of movement". Journal of Experimental Psychology. 47 (6): pp. 381–391.

ДОДАТКИ

Додаток А.

СТРУКТУРА БАЗИ ДАНИХ ГРАМАТИЧНОГО СЛОВНИКА