НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.4'24

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___»_____2021 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою
«Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Спосіб та програмне забезпечення інтелектуального асистування при написанні природномовних текстів»

Виконав:
студент ІІ курсу, групи КП-01мп
Мединський Микола Сергійович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Тетяна Миколаївна _____

Консультант з нормоконтолю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри СПіСКС, к.т.н., доцент,
Петрашенко Андрій Васильович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2021 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність (спеціалізація) – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___»_____2020 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Мединському Миколі Сергійовичу

1. Тема дисертації «Спосіб та програмне забезпечення інтелектуального асистування при написанні природномовних текстів», науковий керівник дисертації Заболотня Тетяна Миколаївна, к.т.н., доцент, затверджені наказом по університету від «05» листопада 2021 р. № 3682-С.

2. Термін подання студентом дисертації «14» грудня 2021 р.

3. Об'єкт дослідження: автоматизоване оброблення природномовних текстів.

4. Предмет дослідження: способи та програмні засоби для інтелектуального редагування природномовних текстів.

5. Перелік завдань, які потрібно розробити:
   - аналіз предметної області, постановка задачі;
   - дослідження існуючих способів та програмних засобів для автоматизованого оброблення природномовних текстів;
   - розробка модифікованого способу для інтелектуального редагування природомовних текстів;
   - створення програмного забезпечення, що реалізує запропонований спосіб інтелектуального асистування при написанні природномовних текстів;
   - визначення ефективності запропонованого способу;

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
   - схема взаємозв'язку підрозділів системи;
   - схема модифікованого способу;
   - схема роботи системи в автоматизованому режимі;
   - результати роботи розробленого методу;

7. Орієнтовний перелік публікацій:
   - Тези доповіді “Підсистема інтелектуального редактора природномовних текстів”

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання «04» жовтня 2020 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	17.10.2020	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	04.12.2020	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	15.02.2021	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	05.04.2021	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	15.05.2021	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	15.06.2021	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу; підготовка матеріалів доповіді на конференції ПМК-2021	05.11.2021	
8.	Оформлення текстової і графічної частини магістерської дисертації	04.12.2021	

Студент Микола МЕДИНСЬКИЙ

Науковий керівник Тетяна ЗАБОЛОТНЯ

РЕФЕРАТ

Структура та обсяг магістерської дисертації. Магістерська дисертація складається зі вступу, 4 розділів, висновку, переліку посилань з 25 найменувань, 3 додатків, і містить 19 рисунків, 3 формули. Повний обсяг магістерської дисертації складає 128 сторінок.

Актуальність теми. Проблема асистування створенню природномовних 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 природномовних текстів.

Об'єкт дослідження. Автоматизоване оброблення природномовних текстів.

Предмет досліження. Способи та програмні засоби для інтелектуального редагування природномовних текстів.

Мета дослідження. Метою даного дослідження є вдосконалення інтелектуального асистування при написані природномовних текстів з точки зору збільшення швидкості надання мовної підтримки, зменшення витраченого часу при написанні групи лексем та зменшення обсягу повідомлення при взаємодії з мовним сервером через побудову підсистеми інтелектуального редактора природномовних текстів та створення мовного сервера.

Апробація роботи. Основні положення і результати роботи представлені та обговорені на XIV науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2021 (м. Київ, 17-19 листопада 2021 р.).

Наукова новизна. Вперше запропоновано модифікований спосіб інтелектуального асистування при написанні природномовних текстів, який

відрізняється від існуючих способів автоматизованого оброблення природномовних текстів та інтелектуального асистування тим, що:

1. Вперше було запропоновано спосіб удосконалення протокола мовного сервера за рахунок розробки мовного протоколу та обрання протоколу більш низького рівня в якості транспортного, що забезпечує зменшення обсягу повідомлення.
2. Вперше було розроблено синтаксичні аналізатори з врахуванням афіксів для програмного забезпечення інтелектуального редагування природномовних текстів, що забезпечує більш точне обчислення релевантності лексеми шляхом врахування контексту, за рахунок чого було удосконаленно провайдери розумного доповнення, що забезпечує зменшення витраченого часу при написанні групи лексем.
3. Було створено виділений мовний сервер, що забезпечує зменшення навантаження на мовний клієнт, можливість оновлення мовних провайдерів без необхідності оновлення мовного клієнта та збільшення швидкості надання мовної підтримки.

Пояснювальна записка магістерської дисертації містить результати дослідження основних методів та засобів для спрощення роботи з редагуванням природномовних текстів. Проведено дослідження предметної області та виділено задачі розробки. Проаналізовано сукупності вхідних та вихідних даних задачі. Розроблено діаграми, що показують основну логіку створеного програмного продукту. Надано опис функціональності для користувачів.

Практичне значення одержаних результатів визначається тим, що запропонований спосіб побудови підсистеми інтелектуального редактора природномовних текстів та створення мовного сервера дозволили підвищити точність та швидкість надання мовної підтримки та розробити відповідні програмні засоби, придатні до використання в межах будь-якого

мовного клієнта, що має підтримку всіх розроблених можливостей мовного сервера. Програмний продукт має практичну цінність для спрощення написання, редагування та аналізу природномовних текстів.

Ключові слова: vs code, NLP, LSP, мовний клієнт, мовний сервер, природномовний текст, редактор, UML, ПЗ.

ABSTRACT

The structure and scope of the thesis. The master's dissertation consists of an introduction, 4 chapters, a conclusion, a list of references from 25 titles, 3 appendix, and contains 19 figures, 3 formulas. The full volume of the master's dissertation is 128 pages.

Actuality of theme. The problem of assisting in the creation of natural language texts is mostly limited to spelling and syntax, and hints rarely go beyond supplementing the prefix to the lemma or the most frequent word form, which is why the topic is relevant.

Object of study. Automated processing of natural language texts.

Subject of research. Methods and software for intelligent editing of natural language texts.

The aim of the study. The aim of this study is to improve intelligent assistance in writing natural language texts in terms of increasing the speed of language support, reducing time spent writing a token group and reducing the amount of messages when interacting with a language server by building a subsystem intelligent text editor and creating a language server.

Approbation of work. The main provisions and results of the work were presented and discussed at the XIV scientific conference of undergraduates and graduate students "Applied Mathematics and Computing" PMK-2021 (Kyiv, November 17-19, 2021).

Scientific novelty. For the first time, a modified method of intellectual assistance in writing natural language texts was proposed, which differs from existing methods of automated processing of natural language texts and intellectual assistance in that:

1. For the first time, a method was proposed to improve the protocol of the language server by developing a language protocol and choosing a lower-level protocol as a transport, which reduces the volume of the message.

2. For the first time, affix-based parsers have been developed for intelligent text editing software to more accurately calculate token relevance by context, thereby improving smart add-on providers to reduce the time spent writing a token group.

3. A dedicated language server has been created, which reduces the load on the language client, the ability to update language providers without the need to update the language client and increase the speed of language support.

The practical significance of the obtained results is determined by the fact that the proposed method of building a subsystem of intelligent text editor and creating a language server allowed to increase the accuracy and speed of language support and develop appropriate software suitable for use within any language client that supports all developed language server capabilities. The obtained research results can be used in the educational process for visual presentation of the work with tools for editing natural language texts. The software product has practical value for simplifying the writing, editing and analysis of natural language texts.

Keywords: vs code, NLP, LSP, language client, language server, natural language text, editor, UML, software.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

VS Code – Visual Studio Code, текстовий редактор, середовище розробки програмного забезпечення.

LSP – Language Server Protocol, протокол мовного сервера.

NEM – Natural Element Method, розпізнавання природних об’єктів.

NLP – Natural Language Processing, обробка природної мови.

NLTK – Natural Language Toolkit, інструменти обробки природної мови.

UML – Unified Modeling Language, уніфікована мова моделювання.

ПЗ – програмне забезпечення.

ШІ – штучний інтелект.

ВСТУП

Для виконання завдання магістерської дисертації необхідно дослідити особливості аналізу природномовних текстів. Інтелектуальний редактор текстів має велику практичну цінність, так як він значно зменшує час на роботу з великими об’ємами інформації. На даний час існує багато розрізнених рішень та додатків, які оперують мовними процесорами та аналізаторами. Проте проблема всеодно є, бо немає єдиного рішення, яке змогло б вирішити всі поставлені питання. Актуальність розробки полягає у тому, що необхідно постійно розширювати вже існуючі системи, адаптаючи їх під потреби кінцевих користувачів для отримання максимально точного результату та підвищення ефективності аналізу. Отже, основною задачею даної кваліфікаційної роботи магістра є розробка таких розширень, які допомогли б вдосконалити вже існуючі системи для написання природномовних текстів.

Об’єктом дослідження є мовні інструменти та засоби, що будуть використані для розробки підсистеми для інтелектуального редагування природномовних текстів.

Предметом дослідження є створювані класи, об’єкти, атрибути та встановлені зв’язки між ними, а також модель програмного застосунку, який допоможе спростити написання, перевірку, аналіз та пошук у природномовному тексті.

Метою даного магістерської дисертації є вивчення принципів проектування складних систем для роботи з природномовними текстами, використовуваних практичних методів, а також дослідження предметної області та факторів, впливають на ефективність редагування природномовних текстів. Також важливим аспектом виконання магістерської дисертації є закріплення теоретичних знань та набутих навичок на практичному проекті. У результаті виконання магістерської

дисертації буде розроблена підсистема інтелектуального редактора природномовних текстів за допомогою засобів Visual Studio Code.

Для досягнення поставленої мети були виділені наступні задачі:

1) закріплення теоретичного матеріалу: знайомство з мовою TypeScript, засобами програмування у Visual Studio Code, вивчення основних бібліотек для отримання результатів дослідження;
2) детальний аналіз предметної області для створення засобів редагування природномовних текстів;
3) опис документообігу предметної області, визначення основних джерел і споживачів даних;
4) аналіз інформаційних потреб користувачів;
5) побудова UML діаграми варіантів використання, або діаграми прецедентів;
6) визначення основних класів та їх атрибутів, підготовка словника даних, виділення відношень між об’єктами створюваної системи;
7) вибір і обґрунтування мови програмування для реалізації функціоналу;
8) побудова схем взаємодії об’єктів таопис алгоритмів виконання методів об’єктів;
9) опис програмної реалізації створюваного ПЗ;
10) наведення контрольного прикладу для демонстрації функціональності та інструктажу користувачів.

Для того, щоб досягти поставленої мети, необхідно провести аналіз предметної області, виділити основні значущі об’єкти системи, сформувати класи, виділити атрибути та встановити зв’язки між створеними класами. Після цього необхідно розробити моделі, на основі яких створюватиметься програмний код.

Однією з основних вимог до розроблюваної системи є працездатний веб-додаток, який забезпечує роботу згідно з зазначеним функціоналом. Для

цього необхідно реалізувати зручний та зрозумілий інтерфейс програмного продукту для кінцевого користувача, а також забезпечити ефективне управління усіма елементами розроблюваного програмного продукту. Необхідно забезпечити зрозуміле управління елементами додатка, щоб спростити пошук та аналіз тексту. Використовуючи засоби NLP та функціональність Visual Studio Code, розробити та проаналізувати основні методи, які допоможуть працювати з природномовними текстами, спрощуючи їх редагування та аналіз. Здійснити програмну реалізацію усіх обробників подій на необхідних компонентів для користувача. Організувати обробку виключень у додатку та зручне використання основних компонентів програмного забезпечення. Забезпечити коректне відображення результатів дослідження та інтуїтивно зрозуміле управління компонентами для того, щоб користувач зміг швидко скористатись всіма наявними засобами.

Отже, для виконання завдання кваліфікаційної роботи магістра необхідно дослідити особливості та фактори, що впливають на ефективність аналізу та редагування природномовних текстів, а також створити відповідний програмний продукт, використовуючи засоби Visual Studio Code.

1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ, ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

Обробка природної мови (NLP) – це здатність комп'ютерної програми розуміти людську мову, як вона розмовляється та пишеться – називається природною мовою. Це компонент штучного інтелекту (ШІ).

НЛП існує більше 50 років і має коріння в галузі лінгвістики. Він має різноманітні реальні застосування в ряді областей, включаючи медичні дослідження, пошукові системи та бізнес-аналітику.

Підприємства використовують величезну кількість неструктурованих, важких для тексту даних, і їм потрібен спосіб їх ефективної обробки. Велика частина інформації, яка створюється в Інтернеті та зберігається в базах даних, є природною людською мовою, і донедавна підприємства не могли ефективно аналізувати ці дані. Тут корисна обробка природної мови.

Переваги обробки природної мови можна побачити, якщо розглянути наступні два твердження: «Страхування хмарних обчислень має бути частиною кожної угоди про рівень обслуговування» і «Хороший SLA забезпечує легший нічний сон – навіть у хмарі». Якщо користувач покладається на обробку природною мовою для пошуку, програма визнає, що хмарні обчислення – це сутність, що хмара – це скорочена форма хмарних обчислень, а SLA – це галузевий акронім для угоди про рівень обслуговування.

Це типи нечітких елементів, які часто з'являються в людській мові, і алгоритми машинного навчання історично погано інтерпретували. Тепер, з удосконаленням методів глибокого навчання та машинного навчання, алгоритми можуть ефективно їх інтерпретувати. Ці покращення розширюють широту та глибину даних, які можна використовувати для аналізу.

Простіше кажучи, НЛП являє собою автоматичну роботу з природною людською мовою, як-от мовлення або текст, і хоча сама концепція є захоплюючою, справжня цінність цієї технології виходить із випадків використання [1-3].

НЛП може допомогти вам у виконанні багатьох завдань, а сфери застосування, здається, збільшуються щодня. Згадаємо кілька прикладів:

1. НЛП дозволяє розпізнавати та прогнозувати захворювання на основі електронних медичних карт та власної мови пацієнта. Ця здатність досліджується при захворюваннях, які йдуть від серцево-судинних захворювань до депресії і навіть шизофренії. Наприклад, Amazon Comprehend Medical – це служба, яка використовує НЛП для вилучення захворювань, ліків і результатів лікування із записок пацієнтів, звітів про клінічні випробування та інших електронних медичних записів.
2. Організації можуть визначити, що клієнти говорять про послугу чи продукт, виявляючи та витягуючи інформацію в таких джерелах, як соціальні мережі. Цей аналіз настроїв може надати багато інформації про вибір клієнтів та чинники, що їх впливають на рішення.
3. Винахідник з IBM розробив когнітивного помічника, який працює як персоналізована пошукова система, дізнаючись все про вас, а потім нагадуючи вам ім’я, пісню чи щось, що ви не можете згадати, коли вам це потрібно.
4. Такі компанії, як Yahoo і Google, фільтрують і класифікують ваші листи за допомогою НЛП, аналізуючи текст в електронних листах, які проходять через їхні сервери, і зупиняючи спам ще до того, як він увійде у папку "Вхідні".
5. Щоб допомогти ідентифікувати фейкові новини, NLP Group в Массачусетському технологічному інституті розробила нову систему, щоб визначити, чи є джерело точним чи політично

упередженим, щоб визначити, чи можна довіряти джерелу новин чи ні.

6. Alexa від Amazon і Siri від Apple – це приклади інтелектуальних голосових інтерфейсів, які використовують НЛП, щоб реагувати на голосові підказки і робити все, як-от знайти певний магазин, повідомити нам прогноз погоди, запропонувати найкращий маршрут до офісу або ввімкнути світло вдома.
7. Уявлення про те, що відбувається і про що говорять люди, може бути дуже цінним для фінансових трейдерів. NLP використовується для відстеження новин, звітів, коментарів про можливі злиття компаній, усе можна потім включити в торговий алгоритм для отримання величезного прибутку. Пам’ятайте: купуйте чутки, продавайте новини.
8. НЛП також використовується як на етапах пошуку, так і на етапах відбору кадрів, визначення навичок потенційних найнятих, а також виявлення перспектив, перш ніж вони стануть активними на ринку праці.

Завдяки технології IBM Watson NLP LegalMation розробила платформу для автоматизації рутинних судових завдань і допоможе юридичним командам заощадити час, зменшити витрати та змінити стратегічний фокус.

НЛП особливо процвітає в галузі охорони здоров’я. Ця технологія покращує надання медичної допомоги, діагностику захворювань та знижує витрати, в той час як організації охорони здоров’я все більше впроваджують електронні медичні картки. Той факт, що клінічна документація може бути покращена, означає, що пацієнти можуть бути краще зрозумілі та отримати користь завдяки кращому медичному догляду. Метою має бути оптимізація їхнього досвіду, і кілька організацій уже працюють над цим [1-3].

1.1. Особливості написання природномовних текстів

Обробка природномовних текстів прагне створити машини, які розуміють і реагують на текст або голосові дані – і відповідають власним текстом або мовленням – майже так само, як люди.

Обробка природної мови (NLP) відноситься до галузі інформатики, а точніше, до галузі штучного інтелекту або ШІ, яка займається наданням комп'ютерам здатності розуміти текст і вимовлені слова приблизно так само, як і люди.

НЛП поєднує обчислювальну лінгвістику – моделювання людської мови на основі правил – зі статистичними моделями, моделями машинного навчання та глибокого навчання. Разом ці технології дозволяють комп'ютерам обробляти людську мову у вигляді тексту або голосових даних і «розуміти» її повне значення, враховуючи наміри та настрої мовця чи письменника.

НЛП керує комп'ютерними програмами, які перекладають текст з однієї мови на іншу, реагують на голосові команди та швидко підсумовують великі обсяги тексту – навіть у реальному часі. Є велика ймовірність, що ви взаємодіяли з НЛП у формі голосових систем GPS, цифрових помічників, програмного забезпечення для диктування мови в текст, чат-ботів обслуговування клієнтів та інших зручностей для споживачів. Але НЛП також відіграє зростаючу роль у корпоративних рішеннях, які допомагають упорядкувати бізнес-операції, підвищити продуктивність співробітників і спростити критично важливі бізнес-процеси [1-3].

Людська мова сповнена неоднозначностей, які ускладнюють створення програмного забезпечення, яке точно визначає передбачуваний зміст тексту або голосових даних. Омоніми, омофони, сарказм, ідіоми, метафори, граматичні винятки та винятки з використання, варіації в структурі речень – це лише деякі з порушень людської мови, для вивчення яких у людей потрібні роки, але програмісти повинні навчати програм,

керованих природною мовою, розпізнавати та з самого початку точно зрозуміти, чи будуть ці програми корисними.

Кілька завдань НЛП розбивають людський текст і голосові дані таким чином, щоб допомогти комп'ютеру зрозуміти, що він поглинає. Деякі з цих завдань включають наступне [16-18]:

1. Розпізнавання мовлення, яке також називають мовлення в текст, є завданням надійного перетворення голосових даних у текстові дані. Розпізнавання мови необхідне для будь-якої програми, яка виконує голосові команди або відповідає на запитання. Що робить розпізнавання мовлення особливо складним, так це те, як люди розмовляють – швидко, невиразно складаючи слова разом, з різним наголосом та інтонацією, з різними акцентами та часто з використанням неправильної граматики.
2. Позначення частини мови, яке також називають граматичним тегом, – це процес визначення частини мови певного слова чи фрагмента тексту на основі його використання та контексту. Частина мови визначає «зробити» як дієслово у «Я можу зробити паперовий літак» і як іменник у «Яку марку автомобіля у вас?».
3. Розшифровка значень слова – це вибір значення слова з кількома значеннями за допомогою процесу семантичного аналізу, який визначає слово, яке має найбільший сенс у даному контексті. Наприклад, розшифровка значень слів допомагає розрізнити значення різних частин мови у реченнях.
4. Розпізнавання іменованих об'єктів, або NEM, визначає слова або фрази як корисні об'єкти. NEM визначає «Кентуккі» як місце розташування або «Фред» як ім'я чоловіка.
5. Розв'язання спільного посилання – це завдання визначити, чи два слова посилаються на один і той самий об'єкт. Найпоширенішим прикладом є визначення особи чи об'єкта, до

якого відноситься певний займенник (наприклад, «вона» = «Марія»), але це також може включати визначення метафори чи ідіоми в тексті (наприклад, приклад, у якому «ведмідь» - це не тварина, а велика волохата людина).

6. Аналіз настроїв намагається виділити з тексту суб'єктивні якості – ставлення, емоції, сарказм, розгубленість, підозру.
7. Генерація природної мови іноді описується як протилежність розпізнавання мовлення або перетворення мови в текст; це завдання передати структуровану інформацію людською мовою.

Обробка природної мови є рушійною силою машинного інтелекту в багатьох сучасних додатках реального світу. Ось кілька прикладів [16-21]:

1. Виявлення спаму. Можливо, ви не думаєте про виявлення спаму як про рішення NLP, але найкращі технології виявлення спаму використовують можливості класифікації тексту NLP для сканування електронних листів на наявність мов, які часто вказують на спам або фішинг. Ці показники можуть включати надмірне використання фінансових термінів, характерну погану граматику, погрозливу лексику, невідповідну терміновість, неправильне написання назв компаній тощо. Виявлення спаму є однією з кількох проблем НЛП, які експерти вважають «в основному вирішеними» (хоча ви можете стверджувати, що це не відповідає вашому досвіду роботи з електронною поштою).
2. Машинний переклад: Google Translate є прикладом широко доступної технології НЛП. Справді корисний машинний переклад передбачає більше, ніж заміну слів однієї мови словами іншої. Ефективний переклад повинен точно охоплювати значення та тон мови введення та перекладати його на текст з тим же значенням і бажаним ефектом на мові виведення. Інструменти машинного перекладу досягають

значного прогресу з точки зору точності. Відмінний спосіб перевірити будь-який інструмент машинного перекладу – перекласти текст однією мовою, а потім повернутися до оригіналу. Часто цитований класичний приклад: не так давно переклад «Дух охочий, але м’ясо слабке» з англійської на російську і назад давав «Горілка хороша, але м’ясо гниле». Сьогодні виходить «Дух бажає, а тіло слабке», що не є ідеальним, але вселяє набагато більше довіри до перекладу з англійської на російську.

3. Віртуальні агенти та чат-боти. Віртуальні агенти, такі як Siri від Apple і Alexa від Amazon, використовують розпізнавання мовлення, щоб розпізнавати шаблони голосових команд і генерацію природної мови, щоб відповідати відповідними діями або корисними коментарями. Чат-боти виконують ту ж магію у відповідь на введені текстові записи. Найкращі з них також вчаться розпізнавати контекстні підказки щодо людських запитів і використовувати їх для надання ще кращих відповідей або варіантів з часом. Наступним покращенням для цих програм є відповіді на запитання, можливість відповідати на наші запитання – передбачуваних чи ні – відповідними та корисними відповідями своїми словами.
4. Аналіз настроїв у соціальних мережах: НЛП стало важливим бізнес-інструментом для виявлення прихованих даних із каналів соціальних мереж. Аналіз настроїв може аналізувати мову, яка використовується в публікаціях у соціальних мережах, відповідях, оглядах тощо, щоб витягти ставлення та емоції у відповідь на продукти, рекламні акції та події – інформацію, яку компанії можуть використовувати в дизайні продуктів, рекламних кампаніях тощо.

5. Підсумування тексту. Підсумування тексту використовує методи НЛП для переварювання величезних обсягів цифрового тексту та створення резюме та конспектів для покажчиків, дослідницьких баз даних або зайнятих читачів, які не мають часу читати повний текст. Найкращі програми для узагальнення тексту використовують семантичне міркування та генерацію природної мови (NLG), щоб додати корисний контекст та висновки до резюме.

1.2. Існуючі програмні засоби для написання природномовних текстів

Синтаксис і семантичний аналіз є двома основними методами, які використовуються при обробці природної мови. На рис. 1.1 можна побачити, наскільки інтерес до тематики ріс з кожним роком [21].

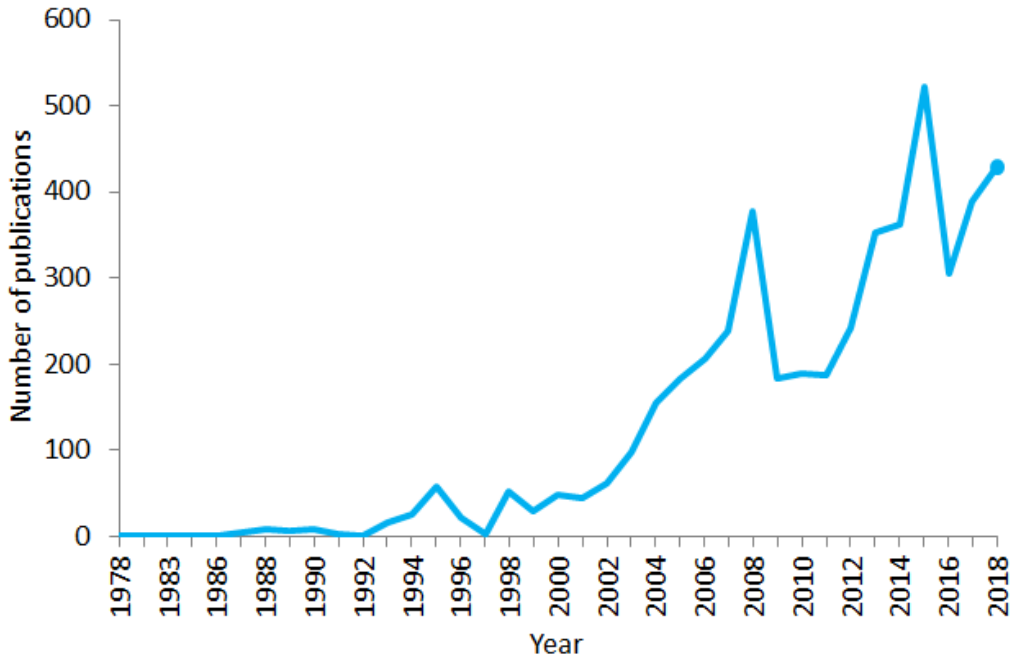


Рис. 1.1. Динаміка вивчення природномовних текстів кожного року

Синтаксис – це розташування слів у реченні для набуття граматичного сенсу. НЛП використовує синтаксис для оцінки значення мови на основі граматичних правил. Синтаксичні методи включають [16-21]:

1. Розбір. Це граматичний аналіз речення. Приклад: Алгоритм обробки природної мови подається реченням «Собака гавкав». Синтаксичний розбір передбачає розбиття цього речення на частини мови, наприклад, собака = іменник, гавкав = дієслово. Це корисно для складніших завдань обробки нижче по ходу.
2. Сегментація слів. Це процес отримання рядка тексту та виведення з нього словоформ. Приклад: людина сканує рукописний документ у комп'ютер. Алгоритм зможе проаналізувати сторінку і розпізнати, що слова розділені пробілами.
3. Порушення речення. Це встановлює межі речень у великих текстах. Приклад: Алгоритм обробки природної мови отримує текст «Собака гавкав. Я прокинувся». Алгоритм може розпізнати крапку, яка розділяє речення, використовуючи розрив речення.
4. Морфологічна сегментація. Це поділяє слова на менші частини, які називаються морфемами. Наприклад: слово untestably розбивається на [[un[[test]able]]ly], де алгоритм розпізнає «un», «test», «able» і «ly» як морфеми. Це особливо корисно для машинного перекладу та розпізнавання мовлення.
5. Стовбурові. Це поділяє слова з флексією в них на кореневі форми. Приклад: у реченні «Собака гавкав» алгоритм зможе розпізнати корінь слова «гавкнув» – «гавкнути». Це було б корисно, якщо користувач аналізував текст для всіх екземплярів слова кора, а також усіх його сполучень. Алгоритм може побачити, що це по суті одне й те саме слово, навіть якщо літери різні.

Семантика передбачає використання та значення за словами [46-48]. Обробка природної мови застосовує алгоритми для розуміння значення та структури речень. Семантичні прийоми включають:

1. Розшифровка сенсу слова. Це виводить значення слова на основі контексту. Приклад: Розгляньте речення «Свиня в загоні». Слово ручка має різні значення. Алгоритм, який використовує цей метод, може зрозуміти, що використання слова ручка тут відноситься до огородженої території, а не до письмового приладдя.
2. Розпізнавання іменованих об’єктів. Це визначає слова, які можна розділити на групи. Приклад. Алгоритм, що використовує цей метод, може проаналізувати статтю новин і визначити всі згадки певної компанії чи продукту. Використовуючи семантику тексту, він зможе розрізняти сутності, які візуально однакові. Наприклад, у реченні «Син Деніела Макдональда пішов у McDonald's і замовив Happy Meal» алгоритм міг розпізнати два екземпляри «McDonald's» як дві окремі об’єкти – один ресторан, а другий – особу.
3. Породження природної мови. Це використовує базу даних для визначення семантики за словами та створення нового тексту. Приклад. Алгоритм може автоматично написати підсумок результатів платформи бізнес-аналітики, зіставляючи певні слова та фрази з характеристиками даних у платформі BI. Іншим прикладом може бути автоматичне створення статей новин або твітів на основі певного тексту, який використовується для навчання.

Сучасні підходи до обробки природної мови засновані на глибокому навчанні, типі штучного інтелекту, який вивчає та використовує шаблони в даних для покращення розуміння програми. Моделі глибокого навчання вимагають величезної кількості мічених даних для алгоритму обробки

природної мови, щоб тренуватися та ідентифікувати відповідні кореляції, а збирання такого набору великих даних є однією з основних перешкод для обробки природної мови [16-18].

Попередні підходи до обробки природної мови передбачали підхід, більш заснований на правилах, коли простіші алгоритми машинного навчання вказували, які слова та фрази шукати в тексті, і давали конкретні відповіді, коли ці фрази з'явилися. Але глибоке навчання – це більш гнучкий, інтуїтивно зрозумілий підхід, у якому алгоритми вчаться визначати наміри мовців на багатьох прикладах – майже так, як дитина вивчає людську мову.

Три інструменти, які зазвичай використовуються для обробки природної мови, включають Natural Language Toolkit (NLTK), Gensim і Intel Natural Language Processing Architect. NLTK – це модуль Python з відкритим вихідним кодом із наборами даних та навчальними посібниками. Gensim – це бібліотека Python для тематичного моделювання та індексування документів. Intel NLP Architect – це ще одна бібліотека Python для топологій і методів глибокого навчання [16-21].

Для розробки програмного продукту можна використовувати відкриті інстурменти Visual Studio Code. Вони встановлюються з магазину Visual Studio Code. Це необхідне розширення, яке дозволяє створювати цифрові людські читачі. У подальшому дані наробки можна використовувати для аналізу та розуміння тексту, а також спрощення проведеного дослідження. Незалежно від того, чи то для вилучення чи розмітки «безладного» тексту чи повного розуміння НЛП, ця мова дозволяє «все, що можна мислити».

Це розширення мови VSCode для NLP++, яке відтворює функціональність VisualText, яка працювала в Microsoft Windows протягом останніх двох десятиліть. NLP++ – це комп'ютерна мова з відкритим вихідним кодом, спеціально призначена для створення аналізаторів тексту, які імітують людей, які читають, і включає в себе мову NLP++ і систему знань, яка називається «концептуальна граматика». NLP++

використовується для будь-якого типу обробки тексту, від простого тегування або вилучення до повного розбору мови. Існує повний англійський аналізатор, який є безкоштовним і доступним для використання. При цьому використовувати розширення можна у власних проектах для розробки нових програмних продуктів з мінімальними часовими витрами. Відсутня необхідність встановлювати окреме програмне забезпечення або використовувати інші сторонні засоби.

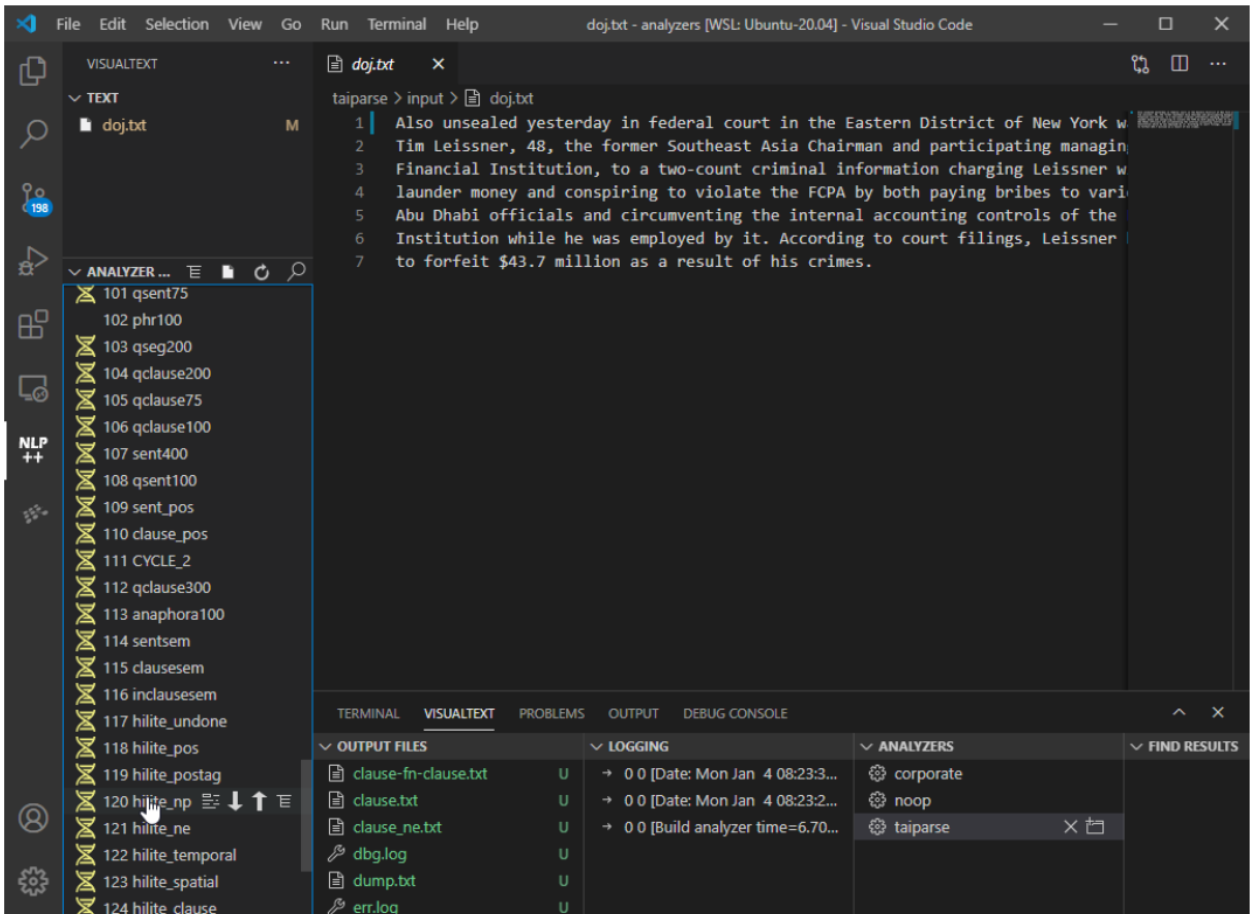


Рис. 1.2. Вигляд вікна розширення для аналізу природномовних текстів

Розширення мови та необхідний NLP-ENGINE працюють у Linux, Windows та MacOS. На рис. 1.2 можна побачити зображення використовуваного розширення.

Існує багато типів аналізаторів, написаних програмістами NLP++, включаючи:

1. Тегування тексту.
2. Аналіз електронних листів, дат, адрес, тощо з неструктурованого тексту.
3. Вилучення освної суті природномовних текстів.
4. Повний розбір НЛП.
5. Аналіз настроїв у текстах користувачів.
6. Очищення OCR.
7. Вилучення даних із неправильного тексту.
8. Автоматична генерація фрагментів з документації або повних документів.

Перераховані засоби будуть використані для розробки власного програмного продукту у Visal Studio Code.

1.3. Аналіз інформаційних потреб користувачів

Деякі з основних функцій, які виконують алгоритми обробки природної мови:

1. Класифікація тексту. Це передбачає призначення тегів до текстів, щоб помістити їх у категорії. Це може бути корисно для аналізу настроїв, який допомагає алгоритму обробки природної мови визначити настрої або емоції за текстом. Наприклад, коли бренд А згадується в кількості Х текстів, алгоритм може визначити, скільки з цих згадок було позитивним, а скільки негативним. Це також може бути корисним для виявлення намірів, що допомагає передбачити, що оратор або автор може зробити на основі тексту, який вони створюють.
2. Вилучення тексту. Це передбачає автоматичне підсумовування тексту та пошук важливих фрагментів даних. Одним із

прикладів цього є виділення ключових слів, яке витягує найважливіші слова з тексту, що може бути корисно для пошукової оптимізації. Щоб зробити це за допомогою обробки природної мови, потрібне певне програмування – воно не повністю автоматизовано. Однак існує багато простих інструментів вилучення ключових слів, які автоматизують більшу частину процесу – користувачеві потрібно лише встановити параметри в програмі. Наприклад, інструмент може витягти найбільш часто вживані слова в тексті. Іншим прикладом є розпізнавання іменованих об’єктів, яке витягує з тексту імена людей, місць та інших об’єктів.

3. Машинний переклад. Це процес, за допомогою якого комп’ютер перекладає текст з однієї мови, наприклад з англійської, на іншу мову, наприклад французьку, без участі людини.
4. Породження природної мови. Це передбачає використання алгоритмів обробки природної мови для аналізу неструктурованих даних і автоматичного створення вмісту на основі цих даних. Одним із прикладів цього є мовні моделі, такі як GPT3, які здатні аналізувати неструктурований текст, а потім генерувати правдоподібні статті на основі тексту.

Перераховані вище функції використовуються в різних реальних додатках, включаючи:

1) аналіз зворотного зв'язку з клієнтами, де ШІ аналізує огляди соціальних мереж;
2) автоматизація обслуговування клієнтів – коли голосові помічники на іншому кінці телефонної лінії обслуговування клієнтів можуть використовувати розпізнавання мови, щоб зрозуміти, що говорить клієнт, щоб він міг правильно спрямувати дзвінок;

3) автоматичний переклад – за допомогою таких інструментів, як Google Translate, Bing Translator і Translate Me;
4) академічні дослідження та аналіз – де ШІ може аналізувати величезні обсяги академічного матеріалу та дослідницьких робіт не лише на основі метаданих тексту, а й самого тексту;
5) аналіз і категоризація медичних записів – де ШІ використовує інсайти для прогнозування та в ідеалі запобігання захворюванням;
6) текстові процесори, що використовуються для плагіату та коректури - за допомогою таких інструментів, як Grammarly і Microsoft Word;
7) прогнозування акцій і розуміння фінансової торгівлі – використання ШІ для аналізу історії ринку та 10 тисяч документів, які містять вичерпні підсумки про фінансові результати компанії;
8) набір талантів у кадровому ділі;
9) автоматизація рутинних судових завдань – одним із прикладів є адвокат зі штучним інтелектом.

Дослідження, що проводяться в обробці природної мови, зосереджені на пошуку, особливо пошуку на підприємстві. Це означає, що користувачі запитують набори даних у формі запитання, яке вони можуть поставити іншій людині. Машина інтерпретує важливі елементи речення людської мови, які відповідають конкретним ознакам у наборі даних, і повертає відповідь.

НЛП можна використовувати для інтерпретації вільного, неструктурованого тексту та для його аналізу. Існує величезна кількість інформації, що зберігається у безкоштовних текстових файлах, таких як медичні картки пацієнтів. До моделей НЛП на основі глибокого навчання ця інформація була недоступною для комп’ютерного аналізу і не могла бути проаналізована будь-яким систематичним способом. За допомогою НЛП

аналітики можуть просіяти величезну кількість вільного тексту, щоб знайти відповідну інформацію.

Аналіз настроїв є ще одним основним випадком використання НЛП. Використовуючи аналіз настроїв, науковці з даних можуть оцінювати коментарі в соціальних мережах, щоб побачити, як працює бренд їхнього бізнесу, або переглядати нотатки команд обслуговування клієнтів, щоб визначити сфери, де люди хочуть, щоб бізнес працював краще [16-21].

1.4. Формулювання розширенної постановки задачі

Метою даного дослідження є вдосконалення інтелектуального асистування при написані природномовних текстів з точки зору збільшення швидкості надання мовної підтримки, зменшення витраченого часу при написанні групи лексем та зменшення обсягу повідомлення при взаємодії з мовним сервером через побудову підсистеми інтелектуального редактора природномовних текстів та створення мовного сервера. Для досягнення поставленної задачі було виділено кілька підзадач:

1) розробка архітектури програмного модуля – мовного клієнта, який буде використовуватись кінцевим користувачем;
2) розробка модуля інтелектуального аналізу текстового файлу для покращення та простоти аналізу;
3) діагностичне доповнення та розробка методів аналізу текстових документів;
4) робота над додатком для формування підказок при використанні природномовного клієнта.

Розробка архітектури потребує детального вивчення питання, а також уточнення функціональності та інформаційних потреб користувачів. Для створення клієнта було виділено наступні функції майбутнього програмного забезпечення:

1) встановлення необхідних робочих режимів у якості діагностичного модуля;
2) можливість обробляти велику кількість документів одночасно, що пришвидшує роботу для користувачів;
3) можливість використовувати різну локалізацію для оптимізації аналітичних можливостей програмного продукту;
4) збереження змін, що вносять користувачі у процесі використання програмного продукту з метою покращення продовження роботи у майбутньому.

Виявлені функції та характеристики допоможуть спростити функціональність майбутнього додатка та його основні властивості. Згідно з визначеними цілями та задачами розробки було створено UML-діаграму варіантів використання програмного продукту [25]. Результат можно побачити на рис. 1.3.

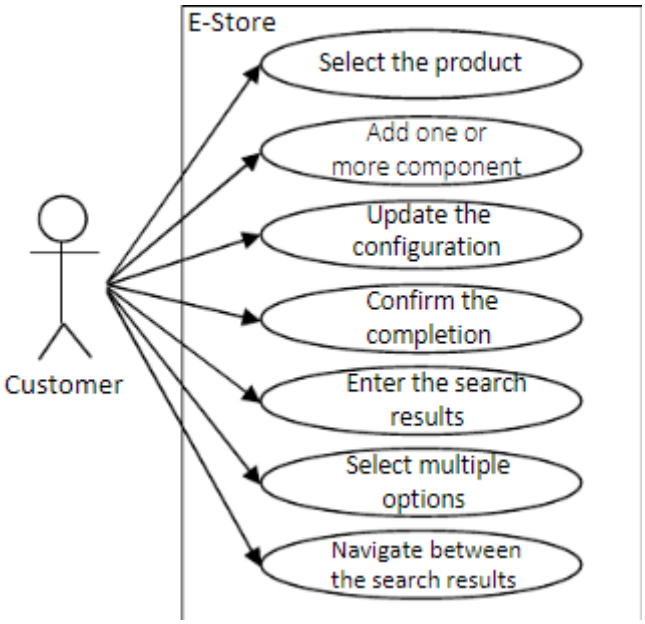


Рис. 1.3. Діаграма варіантів використання

1.5. Висновки до розділу 1

У першому розділі магістерської дисертації було розглянуто основні принципи та методи аналізу природномовних текстів. Розглянуто вже інсуючі засоби та методи, що використовуються на практиці. Виявлено приклади, на які необхідно звернути увагу при реалізації модулів для написання природномовних текстів. Використання Visuao Studio Code та його розширень для створення програмного продукту. Аналіз інформаційних потреб користувачів для розробки та спрощення аналізу. Формулювання теми та мети розробки майбутнього програмного продукту на основі отриманих з аналізу даних.

2. СПОСІБ ІНТЕЛЕКТУАЛЬНОГО АСИСТУВАННЯ ДЛЯ ПРИРОДНОМОВНИХ ТЕКСТІВ

Visual Studio Code поєднує в собі простоту редактора вихідного коду з потужними інструментами розробника, такими як завершення та налагодження коду IntelliSense.

Перш за все, це редактор, який дозволяє виконувати багато основних задач. Основний цикл редагування-складання-налагодження дає менше часу на те, щоб возитися з вашим середовищем, і більше часу на виконання ваших ідей.

В основі Visual Studio Code є швидкий редактор вихідного коду, який підходить для повсякденного використання. Завдяки підтримці сотень мов VS Code допомагає миттєво працювати з підсвічуванням синтаксису, узгодженням дужок, автоматичним відступом, виділенням поля, фрагментами тощо. Інтуїтивно зрозумілі комбінації клавіш, просте налаштування та зіставлення комбінацій клавіш, створені спільнотою, дозволяють легко переміщатися по коду.

Для серйозного кодування надається важлива перевага від інструментів з більшим розумінням коду, ніж просто блоки тексту. Visual Studio Code включає вбудовану підтримку для завершення коду IntelliSense, розширене розуміння семантичного коду та навігацію, а також рефакторинг коду.

І коли кодування стає важким, важкі функціх піддаються більш простому налагодженню. Налаштування та відладка програмного коду часто є єдиною функцією, за якою розробники найбільше пропускають у більш компактному кодуванні, тому ми зробили це. Visual Studio Code включає інтерактивний налагоджувач, тож ви можете переходити через вихідний код, перевіряти змінні, переглядати стеки викликів та виконувати команди на консолі.

VS Code також інтегрується з інструментами збирання та написання сценаріїв для виконання звичайних завдань, що прискорює повсякденні робочі процеси. VS Code підтримує Git, тому ви можете працювати з системою керування кодом, не виходячи з редактора, включаючи перегляд відмінків змін, що очікують на розгляд.

Розширення – це надбудови, які настроюють та покращують Visual Studio. Вони включають додаткові параметри, функції або сценарії застосування для існуючих інструментів. Завдяки тисячам розширень у Marketplace є кілька варіантів для підвищення продуктивності та задоволення потреб робочого процесу, спрощення функціональності у відповідності зі стандартними вимогами.

2.1. Використання Visual Studio Code для розробки засобів аналізу природномовних текстів

Visual Studio Code у якості одного з інструментів розробки використовує мову TypeScript. TypeScript – це типізований наднабір JavaScript, який компілюється у звичайний JavaScript. Він пропонує класи, модулі та інтерфейси, які допоможуть вам створити надійні компоненти. Специфікація мови TypeScript містить повну інформацію про мову [25].

Visual Studio Code включає підтримку мови TypeScript, але не включає компілятор TypeScript, tsc. Щоб перенести вихідний код TypeScript на JavaScript (tsc HelloWorld.ts), вам потрібно буде встановити компілятор TypeScript глобально або у вашому робочому просторі.

Найпростіший спосіб інсталювати TypeScript – через npm, диспетчер пакетів Node.js. Якщо встановлено npm, можна встановити TypeScript глобально (-g) на комп’ютері [25]:

npm install -g typescript

Можна перевірити встановлення розширення, перевіривши версію.

tsc --version

Інший варіант – інсталювати компілятор TypeScript локально у будь-якому проекті (npm install --save-dev typescript), що має перевагу уникнення можливої взаємодії з іншими проектами TypeScript, які можуть бути на одному пристрої.

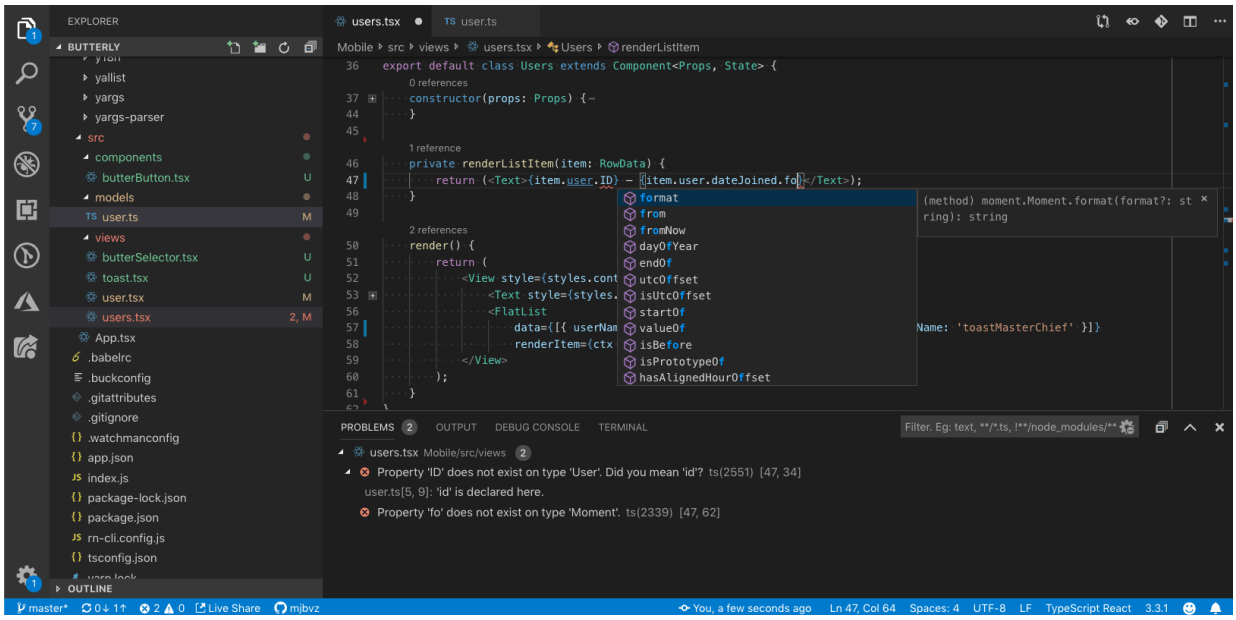


Рис. 2.1. Вигляд вікна TypeScript

Використання TypeScript можна побачити на рис. 2.1. Тут представлено основны структури даних та структуру проекту, яка є стандартною для кожного нового проекта.

Серед основних інструментів у Visual Studio Code варто виділити наступні [19]:

1) Git – VS Code має вбудовану підтримку для керування вихідним кодом за допомогою Git, але вимагає окремого встановлення Git.
2) Node.js (включає npm) – міжплатформне середовище виконання для створення та запуску програм JavaScript.
3) TypeScript – компілятор TypeScript, tsc, для транспіляції TypeScript у JavaScript.

Visual Studio Code інтегрується з існуючими інструментальними засобами для реалізації конкретних задач, що було використано у даному

проекті. Серед найбільш популярних інструментів серед розробників варто виділити наступні:

1. Yeoman – інструмент для створення програм, версія командного рядка Файл > Новий проект.
2. generator-hottowel – генератор Yeoman для швидкого створення додатків AngularJS.
3. Express – фреймворк для додатків Node.js із використанням механізму шаблонів Pug.
4. Gulp – система потокового виконання завдань, яка легко інтегрується із завданнями VS Code.
5. Mocha – тестова платформа JavaScript, яка працює на Node.js.
6. Yarn – менеджер залежностей і альтернатива npm.

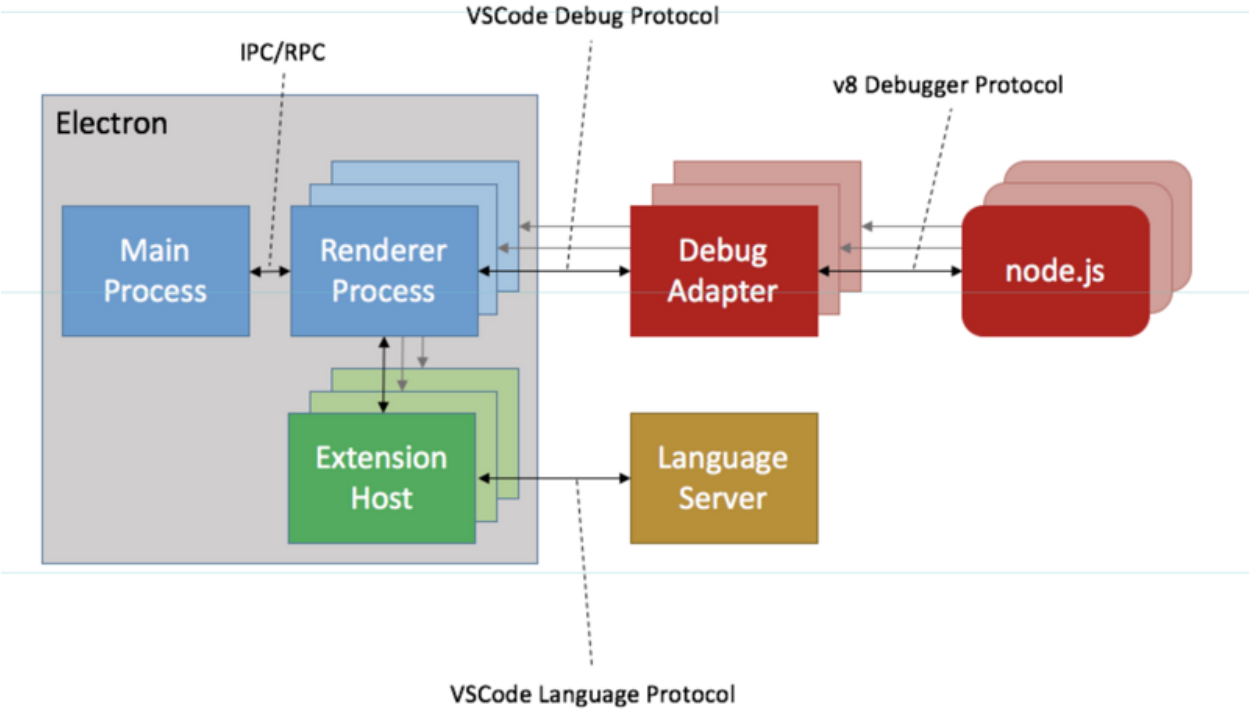


Рис. 2.2. Загальний вигляд мовного протоколу у Visual Studio Code

Більшу частину додаткових компонентів можна встановлювати окремо для розробки певних програмних продуктів та практичніх інструментів (рис. 2.2).

2.1.1. Серверна частина для розробки

Пакет розширення Visual Studio Code Remote Development дозволяє відкривати будь-яку папку в контейнері, на віддаленому комп'ютері (через SSH) або в підсистемі Windows для Linux і використовувати повний набір функцій VS Code. Це означає, що VS Code може надати локальний досвід розробки, включаючи повний IntelliSense (завершення), налагодження тощо – незалежно від того, де розташований або розміщений код [20-25].

Деякі переваги віддаленої розробки включають [20-25]:

1. Можливість редагувати, створювати або налагоджувати в ОС, відмінній від локальної.
2. Можливість розвиватися в середовищі, яке відповідає цільовому середовищу розгортання.
3. Використання більшого або більш спеціалізованого обладнання, ніж ваша локальна машина для розробки.
4. Можливість редагувати код, що зберігається в іншому місці, наприклад у хмарі або на сайті клієнта.
5. Розділення середовищ розробників, щоб уникнути конфліктів, покращити безпеку та прискорити вступ.

У порівнянні з використанням спільного доступу до мережі або синхронізації файлів, VS Code Remote Development забезпечує значно кращу продуктивність разом із кращим контролем над середовищем розробки та інструментами.

Віддалена розробка коду Visual Studio дозволяє вашій локальній інсталяції VS Code прозоро взаємодіяти з вихідним кодом і середовищем виконання на інших машинах (віртуальних чи фізичних), переміщуючи виконання певних команд на «віддалений сервер». VS Code Server швидко встановлюється за допомогою VS Code, коли ви підключаєтеся до віддаленої кінцевої точки, і може розміщувати розширення, які взаємодіють безпосередньо з віддаленим робочим середовищем, машиною та файловою

системою. На рис. 2.3 можна побачити схематичне відображення особливостей роботи серверу у Visual Studio Code [20-25].

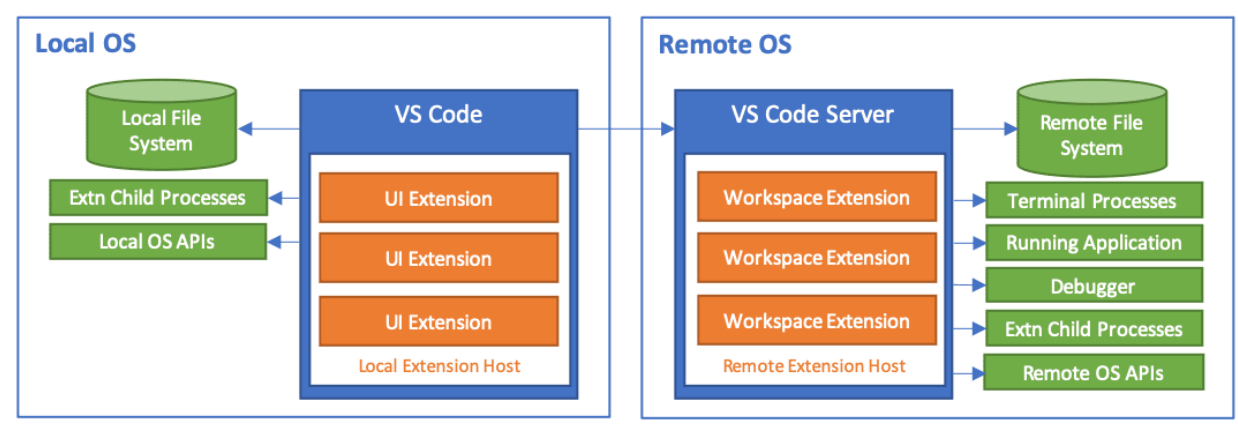


Рис. 2.3. Схема роботи серверу у Visual Studio Code

Віддалена розробка коду Visual Studio використовує існуючі добре відомі транспортні засоби, такі як безпечна оболонка, для автентифікації та захисту трафіку. Не потрібно публічно відкривати порти, крім тих, які використовуються цими відомими безпечними транспортними протоколами.

Сервер VS Code [15], що впроваджується, працює від того самого користувача, якого ви використовували для входу на комп'ютер, гарантуючи, що VS Code та його розширенням не надано неналежний підвищений доступ без дозволу. Сервер запускається та зупиняється за допомогою VS Code і не підключений до жодного користувача чи глобального входу чи сценаріїв запуску. VS Code керує життєвим циклом сервера, тому не потрібно турбуватися про те, чи працює він [20-25].

Для встановлення VS Code Server потрібно, щоб ваша локальна машина мала вихідне підключення HTTPS (порт 443) до:

1) update.code.visualstudio.com
2) *.vo.msecnd.net (Azure CDN)

За замовчуванням Remote – SSH намагатиметься завантажити на віддаленому хості, але якщо ви ввімкнете

remote.SSH.allowLocalServerDownload, розширення повернеться до завантаження VS Code Server локально та передавання його віддалено після встановлення з’єднання.

Розширення Remote – Containers завжди завантажується локально та передається в контейнер.

Можна встановити розширення вручну без підключення до Інтернету за допомогою команди Extensions: Install from VSIX..., але якщо використовується панель розширень або devcontainer.json для встановлення розширень, локальній машині та VS Code Server знадобиться вихідний HTTPS (порт 443) доступ до [11-15]:

1) marketplace.visualstudio.com
2) vscode.blob.core.windows.net
3) *.vo.msecnd.net (Azure CDN)
4) *.gallerycdn.vsassets.io (Azure CDN)

Отже, деякі розширення (наприклад, C#) завантажують вторинні залежності з download.microsoft.com або download.visualstudio.microsoft.com. Інші (наприклад, Visual Studio Live Share) можуть мати додаткові вимоги до підключення.

Весь інший зв’язок між сервером і клієнтом VS Code здійснюється через такі транспортні канали залежно від розширення:

3. SSH: автентифікований безпечний тунель SSH.
3. Контейнери: канал зв'язку, налаштований Docker.
3. WSL: випадковий локальний порт.

2.1.2. Підготовка файлу маніфесту

Кожне розширення Visual Studio Code потребує файлу маніфесту package.json у корені структури каталогу розширення.

Можна об’єднати окремі розширення разом у пакети розширень. Пакет розширень – це набір розширень, які будуть встановлені разом. Це дозволяє легко ділитися своїми розширеннями з іншими користувачами або

створювати набір розширень для певного сценарію, наприклад розробки PHP, щоб допомогти розробнику PHP швидко розпочати роботу з VS Code.

Пакет розширень об’єднує інші розширення за допомогою атрибута extensionPack у файлі package.json.

Щоб створити пакет розширень, ви можете скористатися генератором коду yo Yeoman і вибрати опцію «Новий пакет розширень». Існує можливість заповнити пакет набором розширень, які ви зараз інсталювали у вашому екземплярі VS Code. Таким чином, ви можете легко створити пакет розширень зі своїми улюбленими розширеннями, опублікувати його на Marketplace і поділитися ним з іншими [16-19].

Пакет розширень не повинен мати будь-яких функціональних залежностей зі своїми розширеннями в комплекті, а пакетні розширення повинні бути керованими незалежно від пакета. Якщо розширення має залежність від іншого розширення, цю залежність слід оголосити за допомогою атрибута extensionDependencies. На рис. 2.4 зображено додавання файлу маніфесту до існуючого проекту.

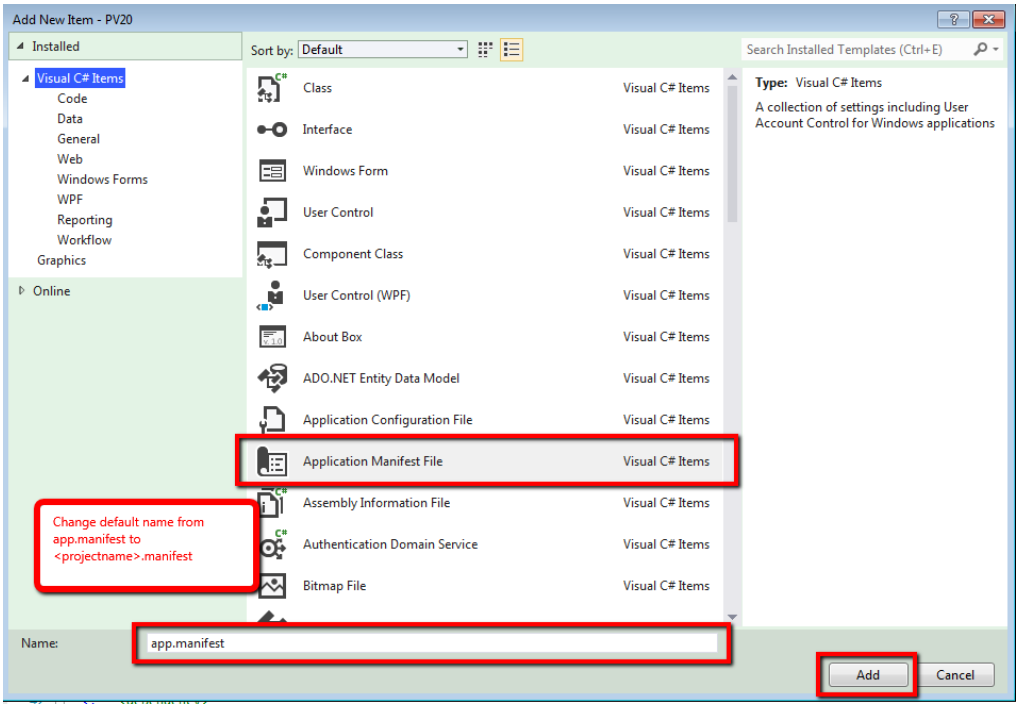


Рис. 2.4. Керування файлами маніфесту

Є кілька модулів Node.js, доступних на npmjs, щоб допомогти з написанням розширень VS Code. Можна включити їх у розділ залежностей вашого розширення [16-19]:

1) vscode-nls – підтримка екстерналізації та локалізації;
2) vscode-uri – реалізація URI, що використовується VS Code та його розширеннями;
3) jsonc-parser – сканер і відмовостійкий аналізатор для обробки JSON з коментарями або без них;
4) request-light – невелика бібліотека запитів Node.js з підтримкою проксі;
5) vscode-extension-telemetry – узгоджені телеметричні звіти для розширень VS Code;
6) vscode-languageclient – легко інтегрувати мовні сервери, які дотримуються протоколу мовного сервера.

2.1.3. Архітектура Visual Studio Code API

Visual Studio Code створено з урахуванням можливості розширення. Від інтерфейсу користувача до досвіду редагування майже кожну частину коду VS можна налаштувати та покращити за допомогою API розширення. Насправді, багато основних функцій VS Code створені як розширення та використовують той самий API розширень [11-15].

Ось кілька прикладів того, чого можна досягти за допомогою Extension API:

1. Змінити зовнішній вигляд VS Code за допомогою кольору або теми піктограм файлу – Теми.
2. Додати користувацькі компоненти та подання в інтерфейс користувача – Розширення робочого середовища.
3. Створити веб-перегляд для відображення спеціальної веб-сторінки, створеної за допомогою HTML/CSS/JS.

4. Підтримка нової мови програмування за допомогою спеціальних мовних розширень.
5. Підтримка налагодження певного середовища виконання за допомогою інструментів для відлагодження програмного продукту.

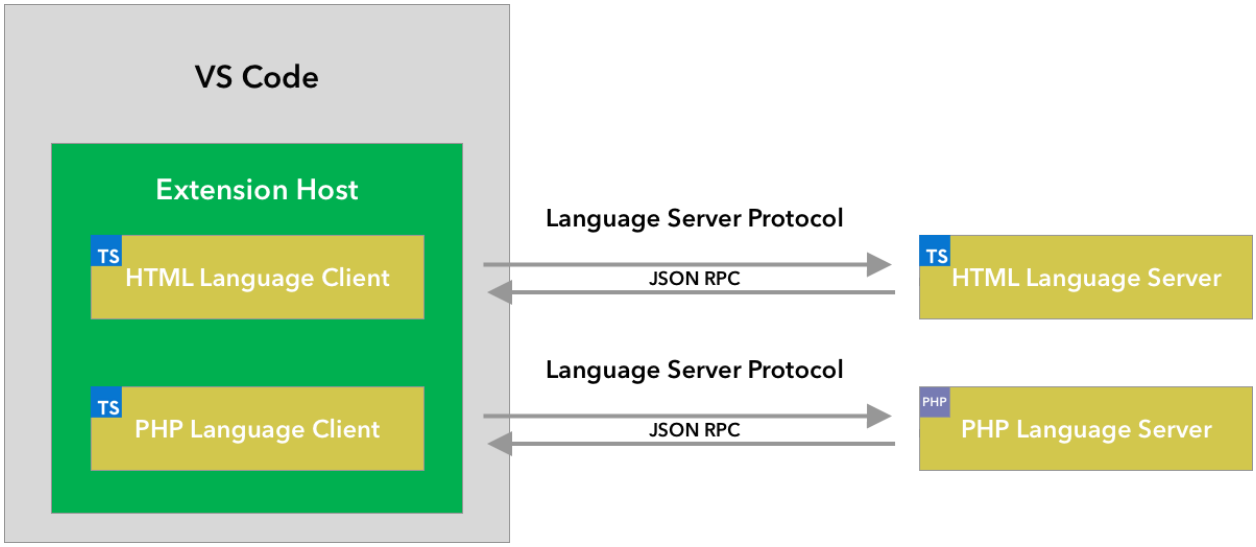


Рис. 2.5. Схема розширень у Visual Studio Code

Щоб створити гарну прибудову, потрібно докласти чимало зусиль. Ось що може допомогти вам кожен розділ документа API (рис. 2.5 – схема розширення VS Code) [11-15]:

1. Початок роботи навчає фундаментальних концепцій створення розширень за зразком Hello World.
2. Можливості розширення розбирають величезний API VS Code на менші категорії та вказують на більш детальні теми.
3. Посібники з розширень включають посібники та зразки коду, які пояснюють конкретні способи використання VS Code Extension API.
4. Розширення мови ілюструє, як додати підтримку мови програмування за допомогою посібників і зразків коду.

5. Testing and Publishing містить докладні посібники з різних тем розробки розширень, таких як тестування та публікація розширень.
6. Розширені теми пояснюють такі розширені концепції, як розширений хост, підтримка віддаленої розробки та кодові простори GitHub, а також запропонований API.
7. Посилання містить вичерпні довідки щодо API коду VS, балів внеску та багатьох інших тем.

2.1.4. Використання модуля для публікації

Публікатор – це особа, яка може публікувати розширення на Visual Studio Code Marketplace. Кожне розширення має включати ім'я видавця у файл package.json.

Розробники можуть створити нового видавця на сторінці керування видавцями Visual Studio Marketplace. Необхідно ввійти, використовуючи той самий обліковий запис Microsoft, який було використано для створення токена особистого доступу [11-15].

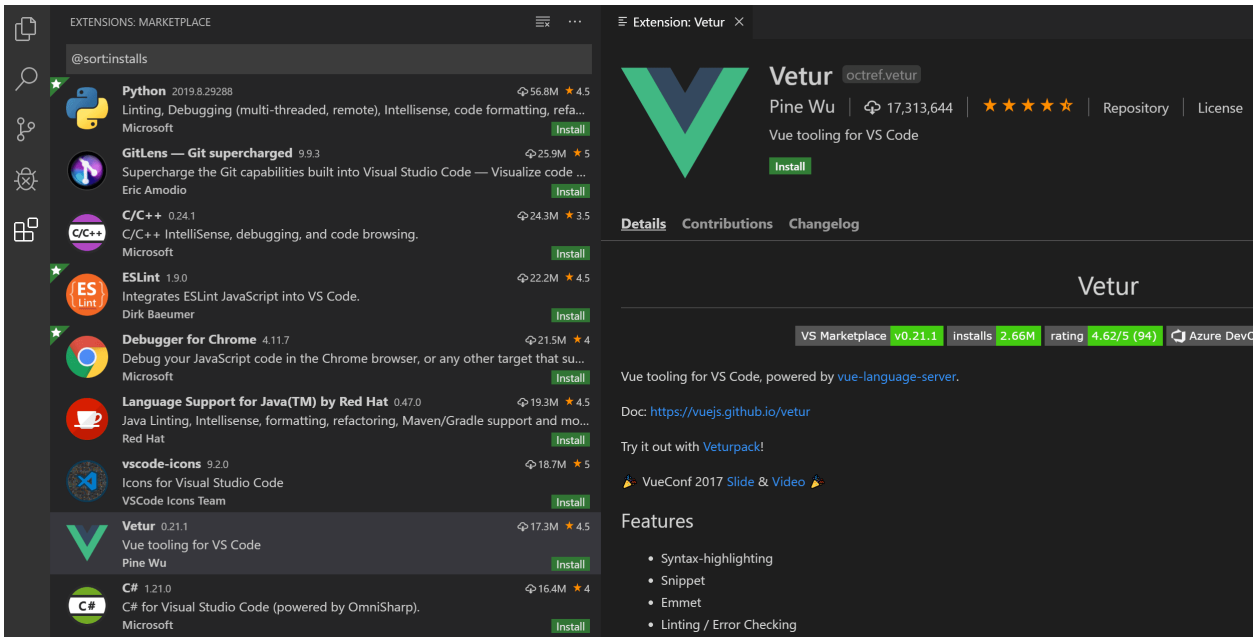


Рис. 2.6. Вікно управління розширеннями VS Code

Якщо потребується протестувати розширення на локальній установці VS Code або розповсюдити розширення, не публікуючи його на VS Code Marketplace, ви можете запакувати своє розширення. vsce може запакувати ваше розширення у файл VSIX, з якого користувачі можуть легко встановити. Деякі розширення публікують файли VSIX до кожного випуску GitHub (рис. 2.6).

Для авторів розширення вони можуть запустити пакет vsce у кореневій папці розширення, щоб створити такі файли VSIX.

Для користувачів, які отримують такий файл VSIX, вони можуть встановити розширення з кодом --install-extension my-extension-0.0.1.vsix.

Розробники можуть налаштувати вигляд розширення на Visual Studio Marketplace.

Для покращення вигляду розширення на Marketplace використовуються наступні методи:

1. Файл README.md у корені розширення використовуватиметься для заповнення вмісту сторінки Marketplace розширення. vsce змінить посилання README двома різними способами:
    1) Якщо додається поле репозиторію до пакета.json, і це загальнодоступне сховище GitHub, vsce автоматично виявить його і відповідно налаштує відносні посилання, використовуючи головну гілку за замовчуванням. Можна автоматично замінити гілку GitHub за допомогою прапора --githubBranch під час запуску пакету vsce або vsce publish.
    2) Для більш детального контролю розробник може встановити прапорці --baseContentUrl та --baseImagesUrl, щоб встановити базові URL-адреси для відносних посилань.
2. Файл ЛІЦЕНЗІЇ в корені розширення буде використовуватися як вміст ліцензії розширення.

3. Файл CHANGELOG.md у корені розширення буде використовуватися як вміст журналу змін розширення.
4. Розробник може встановити колір фону банера, встановивши galleryBanner.color на передбачуване шістнадцяткове значення в package.json.
5. Можна встановити піктограму, встановивши для значка відносний шлях до квадратного PNG-файлу розміром 128 пікселів, включеного у ваше розширення, у package.json.

Описані вище технологічні засоби та інструменти Visual Studio Code будуть використаны для реалізації програмного продукту.

2.2. Опис математичної моделі та оцінка складності розроблюваного програмного забезпечення

Архітектура використовуваних природномовних текстів виглядає як на рис. 2.7.

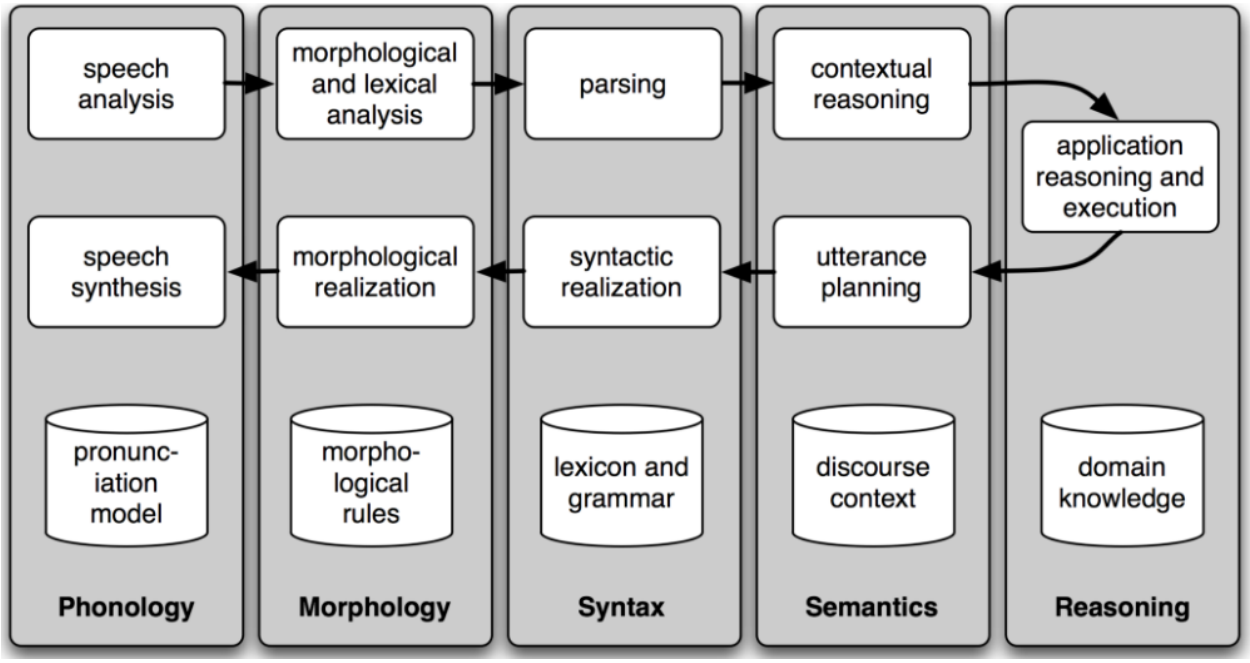


Рис. 2.7. Схема аналізу природномовних текстів

Для вимірювання характеристик програмних засобів використовують метрики. У дослідженні метрик ПО існує два основних напрямки [12-14]:

1) пошук метрик, що характеризують специфічні властивості програм, тобто метрик оцінки самого програмного забезпечення;
2) використання метрик для оцінки технічних характеристик і чинників розробки програм, тобто метрик оцінки умов розробки програм.

Основними напрямками досліджень, для яких застосовуються метрики, є:

1) оцінки топологічної та інформаційної складності програм;
2) оцінки надійності програмних систем;
3) оцінки продуктивності ПО;
4) оцінки рівня мовних засобів і їх застосування;
5) оцінки можливості розуміння програмних текстів, що необхідно для супроводу і модифікації програм;
6) оцінки продуктивності праці програмістів для складання планів і графіків робіт зі створення ПЗ.

Для тестування ПО з точки зору функціональності найбільш важливими характеристиками є характеристики першої групи, оскільки саме від цих характеристик безпосередньо залежить обсяг тестування. Крім того, ці метрики в більшості своїй досить прості, але цілком придатні для отримання хороших результатів [15].

Є досить велика кількість метрик, що дозволяють оцінити складність програмного забезпечення. Як правило, метрики складності ділять на три основні групи [12-14]:

1) метрики розміру ПО;
2) метрики складності потоку керування ПО;
3) метрики складності потоку даних ПО.

Метрики першої групи засновані на визначенні кількісних характеристик, пов'язаних з аналізом вихідного тексту програми і його розміру. До найбільш відомих метрик даної групи відносяться число операторів програми, кількість рядків вихідного тексту, набір метрик по Холстеду [15].

Найбільш простий характеристикою розміру програм є кількість рядків вихідного коду (LOC-оцінка – Lines Of Code). Дана характеристика хоча і не дає можливість зробити остаточний висновок про складність ПО, оскільки LOC істотно залежить від використовуваної мови програмування, проте дозволяє класифікувати програми за обсягом. В даний час частіше використовується термінологія «Source Lines of Code – SLOC». Ця метрика є досить простим способом оцінки складності ПО, в тому числі її застосовують для грубої оцінки обсягу робіт за проектом. Однак така метрика не враховує особливості сучасних мов програмування, а спирається на принцип, коли в одному рядку коду міститься одна команда мови.

Залежно від того, яким чином враховується вихідний код, виділяють два основних показника SLOC [22-24]:

1) кількість «фізичних» рядків коду (що використовуються абревіатури LOC, SLOC, KLOC, KSLOC, DSLOC) – визначається як загальне число рядків вихідного коду, включаючи коментарі і порожні рядки;
2) кількість «логічних» рядків коду SLOC (використовувані абревіатури LSI, DSI, KDSI, де «SI» – source instructions) – визначається як кількість команд і залежить від використовуваної мови програмування.

Для метрики SLOC існують похідні метрики, які дозволяють отримати різні характеристики створюваного ПО, наприклад, метрика зрозумілості програм – відсоток коментарів [25].

Для супроводу і модифікації ПО необхідно мати код з великою кількістю коментарів. Це пов'язано і з тим, що при внесенні змін до ПО до

роботи може бути притягнутий і інший програміст, і з тим, що з плином часу сам розробник не пам'ятає деталі реалізації. Для оцінки зрозумілості коментарі повинні бути розподілені рівномірно, оскільки досить часто зустрічається докладно документована шапка класу, а сам код не забезпечений коментарями. Для розрахунку зрозумілості ПО може бути використаний наступний принцип: код розбивається на N рівних по числу рядків ділянок і для кожного з них визначається функція F_i:

$$F_i = SGN\left(\frac{N_{\text{коммент}i}}{N_i} - 0{,}1\right) \tag{2.1}$$

де $N_{\text{коммент}i}$ – число рядків, забезпечених коментарями на і-ій ділянці; N_i – загальне число рядків на дільниці; функція

$$SGN = \begin{cases} 1, & x > 0; \\ 0, & x = 0; \\ -1, & x < 0. \end{cases} \tag{2.2}$$

Віднімання 0,1 з приватного призводить до того, що якщо менше 10% рядків забезпечені коментарями, то значення функції SGN

$$F = \sum_{i=1}^{n} F_i \tag{2.3}$$

буде негативним. Загальна оцінка показує, наскільки рівномірно розподілені коментарі. Якщо значення функції дорівнює п, то вважається, що є достатня кількість коментарів і вони розподілені рівномірно. Величина 0,1 може бути змінена у вихідній формулі залежно від вимог до щільності коментарів [25].

Підводячи підсумки, слід ще раз зазначити, що метод SLOC може допомогти в класифікації програм за розміром, але дня ПО аналогічного розміру для оцінки складності необхідно застосовувати інші метрики [12-14]. На рис. 2.8. можна побачити діаграму станів створюваного додатка.

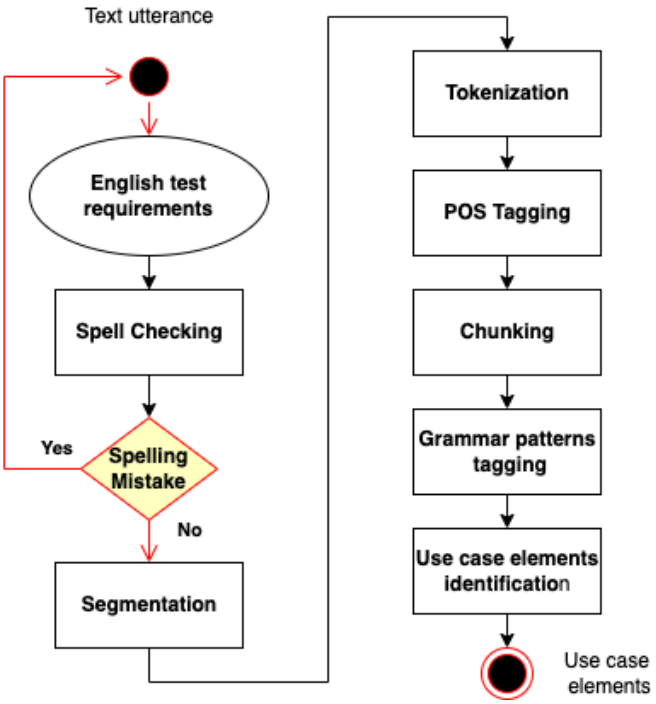


Рис. 2.8. UML діаграма станів створюваного додатка

2.3. Формулювання вимог до способу та програмного забезпечення

2.3.1. Вимоги до функціональних характеристик

Даний програмний продукт повинен надавати користувачам можливість коректної роботи з додатком. Мається на увазі, що програма повинна безперебійно та безпомилково працювати, давати можливість дітям грати просто та цікаво [24].

Програма повинна підтримувати масштабування, що передбачає впровадження в подальшому в програму нових модулів або функцій.

Крім того, програма повинна [24]:

1) бути надійною в роботі, тобто працювати стабільно та безперебійно, коректно відображати елементи графічної оболонки програми;
2) стійко працювати протягом тривалого часу;
3) працювати під операційною системою сімейства Windows;

4) зрозуміло та коректно відображати дані у зручному для користувача вигляді;
5) мати зрозумілий інтерфейс користувача;
6) мати довідку для того, щоб користувач міг дізнатись більше інформації про програмний продукт, а також про можливості програми та її функціональних елементів;
7) мати ергономічне кольорове оформлення;
8) мати написи, що вільно розрізняються (тобто відповідного розміру та кольору, що буде контрастним до фонового кольору);
9) бути здатною паралельно працювати з іншими додатками.

До функціональних характеристик системи можна віднести (тобто програма повинна забезпечувати можливість виконання нижче перерахованих функцій):

1) запуск користувачем програмного продукту за допомогою використання інтегрованого середовища розробки;
2) введення тестових даних для отримання результату;
3) запуск програми для проведення розрахунків;
4) перегляд статистичних даних;
5) перегляд таблиць, графіків та діаграм з результатами;
6) повторний запуск програми.

Створювана система повинна мати простий та зрозумілий інтерфейс для всіх користувачів. Також необхідно забезпечити розподілення прав доступу кожного з користувачів до даних системи. Система розроблюється для того, щоб покращити аналіз природномовних текстів за допомогою програмних засобів.

2.3.2. Організація вхідних і вихідних даних

Вхідні дані програми повинні бути організовані таким чином, щоб програма могла обробити дані та надати кінцевий результат. Дані, що

вводяться вручну, перевіряються на коректність після спроби збереження; це надає можливість користувачу ввести всі дані максимально коректно, віпдовідно до його потреб з метою аналізу природномовних текстів [25].

Вихідні дані програми організовані у вигляді результатів пошуку, з якими можно ознайомитись у відповідному вікні. Дані вихідні дані вирізняються відповідністю до стандартних вимог та потреб користувачів.

Файли зазначеного формату повинні зберігатися на локальних або знімних носіях, відформатованих відповідно до вимог операційної системи. Звіти формуються як реального часу і передаються користувачеві. Звіти, що є тимчасовими і стираються після завершення роботи програми, можуть бути сформовані заново при наступному запуску комп'ютера. За бажанням, будь-який звіт при роботі з програмним застосунком можна зберегти окремо.

2.3.3. Тимчасові характеристики і розмір пам'яті

Час реакції програми повинен відповідати встановленим стандартним виомгам та потребам. Він не повинен перевищувати встановлені значення та параметри. Наприкла, натискання однієї клавіші та очікування відповіді не повинні первищувати значення 0,25 с. Реакція на запуск та вихід з програмного продукту не повинна перевищувати 2 с, що дозволить досягти поставлених значень з мінімальними ризиками. Реакція на натискання пунктів налаштування ігрового поля не повинна перевищувати 0,2 с. Обсяг займаної оперативної пам’яті не повинен перевищувати 50,4 Мбайт для забезпечення оптимальної обробки інформації та результативності.

2.3.4. Критерії ефективності та якості програми

Якість ПЗ – комплекс параметрів програмного продукту, визначальних здатність виконувати покладені нею функції [22].

На даний момент цей показник регулюється міжнародним стандартом ISO/IEC 25010:2011. Цей стандарт встановлює багаторівневу систему оцінки якості ПЗ, що базується на восьми базових характеристиках.

Основні характеристики якості програмного забезпечення відповідно до стандарту ISO/IEC 25010:2011:

1. Функціональність. ПЗ визнається функціональним, якщо виконує покладені нею завдання, відповідає заданим потребам користувачів. Цей аспект передбачає правильну і точну роботу, сумісність всіх компонентів, що входять до складу.
2. Надійність. Під надійністю ПЗ розуміють безперебійне виконання завдань, що покладаються на нього на заданих умовах протягом встановленого часу.
3. Юзабіліті (зручність використання). Цей параметр характеризує рівень зручності ПЗ для користувачів, його наочність, легкість експлуатації та вивчення.
4. Ефективність. Параметр відповідає ступінь забезпечення продуктом необхідної продуктивності при заданих умовах.
5. Зручність супроводу. Цей показник характеризує простоту аналізу, тестування, корекції компонентів, його обслуговування, а також ступінь адаптації до нових умов.
6. Портативність. Ступінь легкості його перенесення іншу платформу. Забезпечення якості ПЗ передбачає його перевірку по кожному з перерахованих параметрів, виявлення слабких сторін та усунення несправностей.
7. Сумісність. Здатність програмних компонентів взаємодіяти друг з одним.
8. Захищеність, тобто мінімізація загроз, пов'язаних з несанкціонованим читанням, зміною інформації і т. д. Загрози можуть бути пов'язані з некоректним використанням ПЗ,

зовнішнім впливом з боку сторонніх осіб, виходом з ладу технічних засобів.

Моделі мають різну кількість рівнів і повністю або частково збігаються щодо набору характеристик якості. Наприклад, модель якості МакКолла на найвищому рівні має три характеристики: функціональність, модифікованість і переносність, а на нижчих рівнях моделі – 11 підхарактеристик якості і 18 критеріїв (атрибутів) якості. Стандарт ISO 9126 пропонує використовувати для опису внутрішнього та зовнішнього якості ПЗ багаторівневу модель. На верхньому рівні виділено 6 основних характеристик якості ПЗ. Кожна характеристика описується за допомогою кількох вхідних у неї атрибутів. Для кожного атрибута визначається набір метрик, що дозволяють його оцінити. Множина характеристик і атрибутів якості згідно з ISO 9126.

2.4. Розроблені методи для оброблення природномовних текстів

Сьогодні одним з найпопулярніших завдань у Data Science є обробка інформації, представленої у текстовій формі. Саме це подання тексту у вигляді математичних рівнянь, формул, парадигм, шаблонів для розуміння семантики (змісту) тексту для його подальшої обробки: класифікації, фрагментації тощо. Загальна область, яка вирішує описані проблеми, називається природною мовою. Обробка (НЛП) [16-21].

Серед завдань, які вирішує Processing Natural Language Processing, найважливішими є:

1) машинний переклад – перше класичне завдання, яке поставлено перед розробниками NLP-технологій (варто зазначити, що воно ще не вирішене на необхідному на сьогодні рівні якості);
2) перевірка граматики та орфографії – як висновок першого завдання;

3) класифікація тексту – визначення семантики тексту для подальшої обробки (одне з найпопулярніших завдань на сьогоднішній день);
4) розпізнавання іменованих сутностей (NER) – визначення та вибір сутностей із заздалегідь визначеним значенням (використовується для фільтрації текстової інформації та розуміння загальної семантики);
5) резюме – узагальнення тексту до спрощеної форми (переосмислення змісту текстів);
6) генерація тексту – одне із завдань, які використовуються для побудови AI-систем;
7) моделювання теми – техніка виділення прихованих тем із великих обсягів тексту.

Важливо відзначити, що всі ці завдання в сучасній і актуальній обробці природної мови часто об’єднуються в одне, створюючи інтерактивні AI-системи: чат-боти. Це середовище (система), яка допомагає поєднувати людські запити з програмним забезпеченням.

Обробка природної мови зазвичай означає обробку тексту або текстової інформації (аудіо, відео). Важливим кроком у цьому процесі є перетворення різних слів і словоформ в одну мовленнєву форму.

Однією з простих і водночас широко використовуваних метрик є відстань редагування (іноді відома як відстань Левенштейна) – алгоритм для оцінки подібності двох рядкових значень (слова, форми слів, складу слів) шляхом порівняння мінімальної кількості операції для перетворення одного значення в інше. Приклад виконання редагування відстані показано нижче (рис. 2.9).

```
One operation - Edit distance: 1
str1 = "string", str2 = "strong"

Several operations - Edit distance: 3
str1 = "sunday", str2 = "saturday"
```

Рис. 2.9. Приклад використання простої метрики аналізу природномовного тексту

Отже, цей алгоритм включає такі текстові операції:

1) вставлення символу в рядок;
2) видалити (або замінити) символ із рядка іншим символом;
3) заміни символів.

Косинусна подібність – це показник, який використовується для вимірювання подібності тексту в різних документах. Розрахунки для цієї метрики базуються на вимірі подібності вектора за формулою косинусних векторів.

Ви можете використовувати різні текстові елементи або характеристики як вектори, що описують цей текст, наприклад, використовуючи методи векторизації тексту. Наприклад, косинусна подібність обчислює відмінності між такими векторами, які показані нижче на моделі векторного простору для трьох доданків.

Результат розрахунку косинусної подібності описує подібність тексту і може бути представлений як значення косинуса або кута.

Результати обчислення відстані косинуса для трьох текстів у порівнянні з першим текстом (див. зображення вище) показують, що значення косинуса прагне досягти одиниці, а кут до нуля при збігу текстів.

Найбільш інтуїтивно зрозумілий і простий спосіб векторізації текстової інформації включає наступне [16-18]:

1) призначити кожному слову унікальний цілочисельний індекс для побудови словника слів із цілочисельними індексами;

2) підрахувати кількість зустрічей кожного слова та зберегти його (кількість) із відповідним індексом.

В результаті отримується вектор з унікальним значенням індексу та частотами повторення кожного зі слів у тексті.

Методи нормалізації тексту (або нормалізації слів) в обробці природної мови використовуються для попередньої обробки текстів, слів і документів. Такі процедури зазвичай використовуються для правильної інтерпретації тексту (слів або мовлення), щоб отримати більш точні моделі НЛП. Серед них можна виділити [16-21]:

1. Незалежна від контексту нормалізація: видалення небуквенно-цифрових текстових символів.
2. Каноніалізація: перетворення даних у «стандартну», «звичайну» або канонічну форму.
3. Основа: витягує корінь слова.
4. Лематизація: перетворює слово на його лему.

2.5. Висновки до розділу 2

У другому розділі магістерської дисертації було розглянуто особливості використання Visual Studio Code та засобів інтегрованого середовища розробки. Оглянуто використовувані програмні засоби: серверну частину, особливості та характеристики файлу маніфесту, архітектуру Visual Studio Code, використання модуля публікації для публікації створюваного розширення в магазині. Описано математичну модель програмного продукту. Розглянуто та розроблено основні вимоги до створюваного програмного продукту. Зроблено огляд основних засобів архітектури розробрюваних засобів природномовних текстів.

3. РОЗРОБЛЕНІ ПРОГРАМНІ ЗАСОБИ ДЛЯ ПІДТРИМКИ ОБРОБЛЕННЯ ПРИРОДНОМНОВНИХ ТЕКСТІВ

Для розробки програмного продукту було використано програмне середовище Visual Studio Code. У якості мови програмування використано TypeScript [20-21].

TypeScript – це типізований наднабір JavaScript, який компілюється у звичайний JavaScript. Це означає [20]:

1) typed – Ви можете визначити типи змінних, параметрів і повертаних даних.
2) superset – TypeScript додає деякі додаткові функції на додаток до JavaScript. Весь дійсний JavaScript є дійсним TypeScript, але не навпаки.
3) компілюється у звичайний JavaScript – TypeScript не може бути запущений браузером. Таким чином, доступні інструменти піклуються про компіляцію вашого TypeScript у JavaScript, щоб браузер зрозумів.

Visual Studio (VS) Code IDE є однією з найбільш використовуваних платформ для розробників JavaScript, C# і Python і швидко стає одним із трьох найкращих середовищ інструментів у Red Hat. VS Code легко налаштовується і пропонує здоровий і зростаючий ринок для розширень усіх типів і технологій, включаючи розширення для Yeoman.

Yeoman описує себе як «...загальну систему риштувань, що дозволяє створювати будь-які програми. Це дозволяє швидко розпочати роботу над новими проектами та спрощує обслуговування існуючих проектів». У агностичному ключі Yeoman дозволяє користувачам об’єднувати цілі проекти або лише частини.

Yeoman можна використовувати для створення нових розширень VS Code. Розробники Red Hat і Apache Camel нещодавно створили чимало

розширень, включаючи мовну підтримку для Java(™) від Red Hat, ініціалізатор проекту від Red Hat і мовну підтримку для Apache Camel.

Для створення каркасу мовного клієнта був використаний генератор Yeoman [20-21].

Можливості генератора Yeoman:

1) створення нового розширення (TypeScript);
2) створення нового розширення (JavaScript);
3) створення нового розширення з темою для IDE;
4) створення нового розширення з шаблонами;
5) створення підтримки нової мови з модулем підсвічування.

Генератором Yeoman була створена наступна структура VS Code розширення:

1) .gitignore файл з списком виключень для системи керування версіями;
2) .vscodeignore з списком виключень при розгортанні розширення;
3) README.md файл документації;
4) src папку з файлом extension.ts, який вміщає точку входу розширення;
5) out папку, яка вміщає скомпільований код.
6) node_modules папку з залежними модулями;
7) package.json файл, який є маніфестом розширення.

На жаль, існуюче розширення VS Code для Yeoman не оновлювалося з середини 2017 року і перестало належним чином працювати з останньою IDE десь у 2018 році. Щоб виправити цю ситуацію, було створено форк та оновлено його. Новий проект VS Code Yeoman тільки на початковій стадії, але він уже містить виправлення, тому тепер він працює в останніх версіях VS Code.

.NET Core забезпечує швидку і модульну платформу для створення серверних програм, які працюють у Windows, Linux і macOS.

Використовуйте код Visual Studio з розширеннями C# і F#, щоб отримати потужний досвід редагування за допомогою C# IntelliSense, F# IntelliSense (розумне завершення коду) і налагодження [20-21].

Для роботи проекту необхідні наступні програмні рішення у Visual Studio Code:

1. .NET Core SDK. Пакет SDK також включає в себе середовище виконання.
2. Розширення C# з VS Code Marketplace.
3. Розширення F# (Ionide) з VS Code Marketplace.

.NET SDK – це набір бібліотек та інструментів для розробки та запуску програм .NET.

Коли завантажується .NET, можна вибрати пакет SDK або середовище виконання, наприклад середовище виконання .NET або ASP.NET Core. Для роботи необхдіно встановити середовище виконання на машині, яку потрібно підготувати до запуску програм .NET. Встановіть SDK на машині, яку ви хочете використовувати для розробки. Коли завантажується SDK, автоматично з ним надається час виконання.

Завантаження SDK включає такі компоненти:

1. CLI .NET. Інструменти командного рядка, які можна використовувати для локальної розробки та сценаріїв безперервної інтеграції.
2. Драйвер dotnet. Команда CLI, яка запускає програми, що залежать від платформи.
3. Компілятори мови програмування Roslyn і F#.
4. Механізм збірки MSBuild.
5. Середовище виконання .NET. Забезпечує систему типів, завантаження збірки, збірник сміття, рідну взаємодію та інші базові послуги.
6. Бібліотеки часу виконання. Надає примітивні типи даних і основні утиліти.

7. Середовище виконання ASP.NET Core. Надає основні послуги для додатків, підключених до Інтернету, таких як веб-програми, програми Інтернету речей та мобільні серверні системи.
8. Час виконання робочого столу. Надає основні послуги для настільних програм Windows, включаючи Windows Forms і WPF.

Завантаження під час виконання включає такі компоненти [20-21]:

1. За бажанням, робочий стіл або середовище виконання ASP.NET Core.
2. Середовище виконання .NET. Забезпечує систему типів, завантаження збірки, збірник сміття, рідну взаємодію та інші базові послуги.
3. Бібліотеки часу виконання. Надає примітивні типи даних і основні утиліти.
4. Драйвер dotnet. Команда CLI, яка запускає програми, що залежать від платформи.

В якості мови програмування для реалізації мовного сервера було вирішено використати мову C# 7 та платформу .NET 4.7 [21].

CLR використовує транслятор вихідного коду в двійковий код на мові IL, під час запуску програми відбувається JIT (Just-in-time) компіляція, де інструкції IL конвертуються в машинні інструкції. Процес компіляції відбувається кожного разу та займає певний час, тому була надана можливість створення машинного коду завчасно, за допомогою утиліти ngen.exe, проте слід враховувати що це має ряд недоліків. Усі використовувані бібліотеки ядра .NET для прискорення роботи проходять попередню компіляцію до початку роботи документу та зберігаються у глобальному кеші збірок (GAC) [20].

Використання проміжної мови дозволяє розгортати програмний продукт маючи лише одну конфігурацію AnyCPU для різних архітектур

процесора (x86, x64), оскільки в даному випадку JIT компіляція буде виконуватись з врахуванням поточної архітектури процесора.

Компілятор .NET-мови створює метадані і зберігає їх в збірку, яка містить CIL. Збіркою є виконуючий файл “.exe ”або бібліотека “.dll”. Метадані описують всі класи та члени класів, а також класи та члени класів, які поточна збірка викликає з іншої збірки, використовуються при використанні рефлексії.

Microsoft .NET підтримує автоматичний збір сміття за допомогою використання лічильників, управління потоками (як на рівні фізичного потоку так і на рівні асинхронної задачі), обробку виключних ситуацій.

Microsoft .NET надає велику кількість бібліотек, документацію та оновлюється на постійній основі.

Мова C++ CLI, яка підтримує CLI, дозволяє зв'язувати збірки написані на .NET з мовою C++, за рахунок створення обгорток на C++ CLI.

3.1. Мовний клієнт

Клієнт насправді є звичайним розширенням VS Code. Він містить файл package.json в корені папки робочої області. У цьому файлі є три цікаві розділи.

Метою формату мовного сервера індексу (LSIF, вимовляється як «else if») є підтримка розширеної навігації по коду в інструментах розробки або веб-інтерфейсі без необхідності локальної копії вихідного коду. Формат за духом схожий на протокол мовного сервера (LSP), який спрощує інтеграцію можливостей багатого редагування коду в інструмент розробки.

LSP надає функції розширеного розробки коду, такі як автозавершення, форматування за типом і навігація з розширеним кодом. Для ефективного забезпечення цих функцій мовний сервер вимагає, щоб усі файли вихідного коду були доступні на локальному диску. Сервери мови LSP також можуть читати частини або всі файли в пам'ять і обчислювати

абстрактні синтаксичні дерева для забезпечення цих функцій. Мета формату мовного сервера індексу – розширити протокол LSP для підтримки функцій навігації з розширеним кодом без цих вимог. LSIF визначає стандартний формат для мовних серверів або інших інструментів програмування для передачі своїх знань про робочу область коду. Цю збережену інформацію пізніше можна використовувати для відповіді на запити LSP для тієї ж робочої області без запуску мовного сервера [17].

LSIF заснований на LSP і використовує ті ж типи даних, що визначені в LSP. На високому рівні LSIF моделює дані, що повертаються із запитів мовного сервера. Так само, як і LSP, LSIF не містить жодної інформації про символи програми, а також не визначає семантику символу (наприклад, що робить визначення символу або чи метод замінює інший метод). Тому LSIF не визначає базу даних символів, що узгоджується з підходом LSP.

Використання існуючих типів даних LSP як основи для LSIF має ще одну перевагу, оскільки LSIF можна легко інтегрувати в інструменти або сервери, які вже розуміють LSP.

3.1.1. Розробка діагностичного модуля

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 [17].

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. В даній роботі в якості мовного клієнта вибраний VS Code.

Мовний сервер працює як окремий процес, а засоби розробки спілкуються з сервером за допомогою мовного протоколу через JSON-RPC. Нижче наведено приклад того, як клієнт і мовний сервер взаємодіють під час звичайного сеансу редагування:

1. Користувач відкриває файл (іменується документом) у засобі: інструмент сповіщає мовний сервер, що документ відкритий («textDocument/didOpen»). Відтепер правда про вміст документа більше не зберігається у файловій системі, а зберігається інструментом у пам’яті. Тепер вміст має бути синхронізовано між інструментом і мовним сервером.
2. Користувач вносить зміни: інструмент сповіщає сервер про зміну документа («textDocument/didChange»), а мовне представлення документа оновлюється мовним сервером. Коли це відбувається, мовний сервер аналізує цю інформацію та повідомляє інструмент про виявлені помилки та попередження («textDocument/publishDiagnostics»).
3. Користувач виконує «Перейти до визначення» для символу відкритого документа: інструмент надсилає запит «textDocument/definition» з двома параметрами: (1) URI документа та (2) позиція тексту, звідки «Перейти до визначення» ' був ініційований запит до сервера. Сервер відповідає URI документа та положення визначення символу всередині документа.
4. Користувач закриває документ (файл): від інструмента надсилається сповіщення «textDocument/didClose», яке інформує мовний сервер, що документ більше не знаходиться в пам’яті. Поточний вміст тепер оновлений у файловій системі.

Цей приклад ілюструє, як протокол взаємодіє з мовним сервером на рівні посилань на документи (URI) та позицій документа. Ці типи даних нейтральні до мови програмування і застосовуються до всіх мов

програмування. Типи даних не знаходяться на рівні моделі предметної області мови програмування, яка зазвичай забезпечує абстрактні синтаксичні дерева та символи компілятора (наприклад, дозволені типи, простори імен, …). Той факт, що типи даних прості та нейтральні до мови програмування, значно спрощує протокол. Набагато простіше стандартизувати URI текстового документа або позицію курсора в порівнянні зі стандартизацією дерева абстрактного синтаксису та символів компілятора в різних мовах програмування (рис. 3.1).

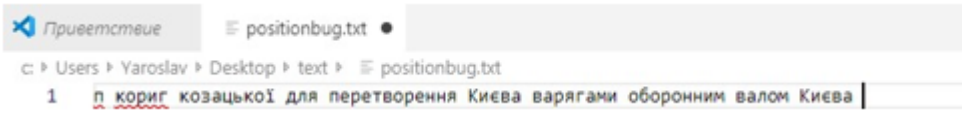


Рис. 3.1. Розробка діагностичного мовного модуля в проекті

Засіб діагностики використовується виключно в певному контексті, але не виконує функціонального значення. Кількість типві підтримуваних мовних одиниць та діпазон залежить від багатьох параметрів.

3.1.2. Використання розумного доповнення для розробки програмного продукту

Інтерфейс постачальника елемента завершення визначає контракт між розширеннями та IntelliSense.

Постачальники можуть відкласти обчислення властивостей {@linkcode CompletionItem.detail} і {@linkcode CompletionItem.documentation documentation}, реалізувавши функцію {@linkcode CompletionItemProvider.resolveCompletionItem resolveCompletionItem}. Однак властивості, необхідні для початкового сортування та фільтрації, як-от sortText, filterText, insertText і діапазон, не повинні змінюватися під час розв’язання.

Провайдери запитують завершення або явно за допомогою жесту користувача, або – залежно від конфігурації – неявно під час введення слів або символів тригера [17].

3.1.3. Розробка модуля підказок мовного клієнта

Модуль підказки призначений для спрощення пошуку клієнта. Він надає можливість виправити помилку при введенні інформації. Для цього необхідно навести курсор на вказану ділянку поля введення.

Повідомлення про помилку виводиться автоматично. Щоб достягти поставленої задачі, було використано автоматизовані модулі та засоби середовища розробки. Для того, щоб отримати результат, необхідно вказати діапазон для аналізу представленої інформації. У результаті користувач має змогу отримати потрібне повідомлення про виконану задачу.

Інше повідомлення формується за допомогою аналізу токенів. Цей масив включає в себе індекси, які формуються в місцях розташування курсору. Побачити результат виконання можна у класі HoverProvider, що реалізує потрібні програмні методи та можливості.

3.1.4. Взаємодія програмного додатка та користувача

Людино-машинна взаємодія використовується для оцінки ефективності взаємодії машини та користувача. Дане питання розглядається на рівні користувальницького інтерфейсу. Серед ключових параметрів оцінки варто звернути увагу на такі:

1) оперативність виконання оперцій користувачем;
2) інтенсивність роботи програми;
3) кількість виявлених помилок, що впливають на робочу ефективність;
4) швидкість та ефективність навчання;
5) продуктивність.

Швидкість обробки операцій залежить від витраченого часу на виконання запитів та виведення результатів. Тривалість дій залежить також від визначення команд, формування мети та задач. Якщо програма допомогає реалізувати поставлені цілі перед користувачем, вона має гарний інтерфейс, що дозволяє виконувати поставлені задачі намного швидше.

У своїй основі Visual Studio Code є редактором коду. Як і багато інших редакторів коду, VS Code використовує звичайний користувальницький інтерфейс і макет провідника зліва, показуючи всі файли і папки, до яких ви маєте доступ, і редактор праворуч, показуючи вміст файлів, які ви відкрили.

VS Code має простий та інтуїтивно зрозумілий макет, який максимально збільшує простір, наданий редактору, залишаючи достатньо місця для перегляду та доступу до повного контексту вашої папки чи проекту. Інтерфейс користувача розділений на п’ять областей:

1. Редактор – основна область редагування файлів. Можна відкривати скільки завгодно редакторів поруч по вертикалі та горизонталі.
2. Бічна панель – містить різні види, як-от Explorer, щоб допомогти під час роботи над проектом.
3. Рядок стану – інформація про відкритий проект і файли, які редагуються.
4. Панель активності. Розташована в дальній лівій частині, вона дає змогу перемикатися між представленнями та надає додаткові індикатори, що залежать від контексту, як-от кількість вихідних змін, коли увімкнено Git.
5. Панелі – можна відображати різні панелі під областю редактора для виведення або налагоджувальної інформації, помилок і попереджень або вбудованого терміналу. Панель також можна перемістити вправо, щоб збільшити простір по вертикалі.

Кожного разу, коли запускаэться VS Code, він відкривається в тому ж стані, в якому був, коли ви востаннє його закривали. Папка, макет і відкриті файли зберігаються.

Відкриті файли в кожному редакторі відображаються із заголовками вкладок (Tabs) у верхній частині області редактора.

У верхній частині Провідника є подання з позначкою для відкриття редакторів. Це список активних файлів або попередніх переглядів. Це файли, які ви раніше відкривали в VS Code, над якими ви працювали. Наприклад, файл буде відображений у поданні відкриття редакторів, якщо:

1. Внеесено зміни до файлу.
2. Обрано заголовок файлу.
3. Вибрано файл у провіднику.
4. Відкрито файл, який не є частиною поточної папки.

Виконавши завдання, ви можете видалити файли окремо з перегляду для відкриття редакторів або видалити всі файли за допомогою дій «Перегляд: закрити всі редактори» або «Перегляд: закрити всі редактори в групі».

При роботі з програмою можливі також помилки, викликані за рахунок людсього фактору. Людські помилки бувают наступного характеру:

1) помилки через недостатнє знання програмного забезпечення та предметної області;
2) помилки через неуважність;
3) некоректна обробка інформації за допомогою модулів системи;
4) помилки, викликані неправильною моторикою.

Кожен вид помилок може привести до некоректного відображення інформації. Також варто звернути увагу на особливості людино-машиної взаємодії при поточних налаштуваннях, щоб оптимізувати основні дії користувача, допомогти йому виконувати поставлені цілі з мінімальними помилками та похибками.

3.2. Мовний сервер та протокол

Реалізація підтримки таких функцій, як автозаповнення, визначення переходу або документація при наведенні курсора, для мови програмування – це значні зусилля. Традиційно цю роботу потрібно повторити для кожного інструменту розробки, оскільки кожен надає різні API для реалізації одних і тих же функцій (рис. 3.2).

Рис. 3.2. Використання мовного сервера

Ідея мовного сервера полягає в тому, щоб надати специфічні для мови інструменти всередині сервера, які можуть спілкуватися з інструментами розробки за протоколом, який забезпечує взаємодію між процесами.

3.2.1. Реалізація мовного протоколу для роботи сервера

Мовний сервер використовує набір бібліотек для роботи з словником, які працюють платформі .NET, отже відсутня необхідність створювати проксі бібліотеки для роботи зі словником між різними платформами виконання.

В якості середовища розробки мовного сервера використовується Visual Studio 17.

Visual Studio, підтримує можливість використання доповнень, надає систему керування пакунками NuGet від Microsoft, яка в першу чергу використовується для .NET.

Language Server – це особливий тип розширення Visual Studio Code, яке забезпечує можливість редагування для багатьох мов програмування. За допомогою мовних серверів ви можете реалізувати автозаповнення, перевірку помилок (діагностика), перехід до визначення та багато інших мовних функцій, які підтримуються в VS Code.

Однак під час реалізації підтримки мовних функцій у VS Code ми виявили три поширені проблеми [4-10]:

По-перше, мовні сервери зазвичай реалізуються на рідних мовах програмування, і це створює труднощі для їх інтеграції з VS Code, який має середовище виконання Node.js.

Крім того, мовні функції можуть бути ресурсомісткими. Наприклад, щоб правильно перевірити файл, мовному серверу потрібно проаналізувати велику кількість файлів, створити для них абстрактні синтаксичні дерева та виконати статичний аналіз програми. Ці операції можуть спричинити значне використання процесора та пам’яті, і ми повинні переконатися, що продуктивність VS Code залишається незмінною.

Нарешті, інтеграція інструментів кількох мов із кількома редакторами коду може зажадати значних зусиль. З точки зору мовних інструментів, вони повинні адаптуватися до редакторів коду з різними API. З точки зору редакторів коду, вони не можуть очікувати жодного уніфікованого API від мовних інструментів. Це робить реалізацію мовної підтримки M мов у редакторах коду N роботою M * N.

Щоб вирішити ці проблеми, Microsoft визначила протокол мовного сервера, який стандартизує зв’язок між мовними інструментами та редактором коду. Таким чином, мовні сервери можуть бути реалізовані

будь-якою мовою та запущені в власному процесі, щоб уникнути витрат на продуктивність, оскільки вони спілкуються з редактором коду через протокол мовного сервера. Крім того, будь-які LSP-сумісні мовні інструменти можуть інтегруватися з кількома LSP-сумісними редакторами коду, а будь-які LSP-сумісні редактори коду можуть легко підібрати кілька LSP-сумісних мовних інструментів [4-10].

3.2.2. Функції та характеристики лексичного аналізатора

У VS Code мовний сервер складається з двох частин:

1. Мовний клієнт: звичайне розширення VS Code, написане на JavaScript / TypeScript. Це розширення має доступ до всього API простору імен VS Code.
2. Мовний сервер: інструмент мовного аналізу, який працює в окремому процесі.

Як коротко зазначено вище, запуск мовного сервера в окремому процесі має дві переваги:

1. Інструмент аналізу може бути реалізований будь-якими мовами, якщо він може спілкуватися з мовним клієнтом відповідно до протоколу мовного сервера.
2. Оскільки інструменти мовного аналізу часто завантажують процесор і пам’ять, їх використання в окремому процесі дозволяє уникнути витрат на продуктивність.

Ось ілюстрація VS Code з двома розширеннями мовного сервера. Клієнт мови HTML і мовний клієнт PHP є звичайними розширеннями VS Code, написаними на TypeScript. Кожен з них створює відповідний мовний сервер і спілкується з ним через LSP. Хоча мовний сервер PHP написаний на PHP, він все ще може спілкуватися з мовним клієнтом PHP через LSP [4-10].

3.2.3. Запити користувача на виконання мовних протоколів

Щоб оброблювати запити кілької клієнтів, необхідно розглянути можливість програмного продукту для обрабоки цих запитів. Відповідно для кожного TCP-клієнта підбирається оптимальний варіант для комунікації із сервером, що дозволяє отримати необхідний результат.

Для підтримки з кількома документами мовний сервер підтримує одночасну роботу з множиною різних процесорних документів. Цей процесор может одночасно оброблювати кілька різних операцій. Це реалізовано за допомогою використання розробленої черги з операцій.

Серед основних типів виконуваних операцій зі сторони мовного сервера виділяють наступні [22]:

1) абстрактний базовий клас для всіх операцій;
2) клас, що відповідає для формування підказок – він наслідує функціональність абстрактного базового класу;
3) операція для інтелектуального доповнення, що наслідує функціональність абстрактного класу;
4) клас для виконання операцій аналізу;
5) операція поточного аналізу операцій, що наслідує існуючий абстрактний клас;
6) операція для первинної обробки та аналізу тексту;
7) операція для поточного аналізу текстового документу.

Кожен клас та операцію мають можливість безперервного виконання або відміни за такої необхідності.

3.2.4. Підтримка працездатності мовних протоколів

Мовний сервер має всю необхідну функціональність для виконання аналізу та пошуку в текстових документах. Серед основних дій та параметрів варто звернути увагу на такі:

1) доповнення поточного тексту, що проходить аналіз, з використанням флексій;

2) перевірка тексту на наявність орфографічних помилок;
3) врахування флексій при проведенні аналізу тексту.

Щоб реалізувати перераховані вище задачі, було використано відкритий словник mphdict. Даний словник має доволі багатий реєстр існуючих слів для аналізу. Це один з найбільших словників даного типу, який міст великий реєстр даних для проведения аналізу та пошуку в документах. Даний словник доступний по відкритим стандартам, що надає можливість кожному використовувати даний програмний продукт для проведення аналізу незалежно від користувацьких потреб користувачів [4-10].

Проект було використано і розроблено за допомогою вже існуючего словника. Дані були впорядковані у форматі баз даних, що надає можливість кожному проводити аналіз на основі отриманих даних. Серед основних типів словників, що входять до структури онлайн-словника виділяють наступні:

1) словник граматичної мови;
2) синонімичний словник української мови;
3) словник про дослідження походження слів.

Також граматичний словник української мови включає в собі всі необхідні таблиці, що потрібні для проведення аналізу.

На рис. 3.3 наведено схематичне відображення основних сутностей та взаємодію між ними. Граматична категорія та інші характеристики значно спрощують пошук необхідної інформації та використання розширення. Унікальний ідентифікатор надає можливість використувати ці дані для проведення пошуку та аналізу.

Для оптимізації створюваної пошукової системи використовуються методи, які створються під час запуску словника. За допомогою такого підходу вдається сформувати спеціальний словник зі словоформами. Це надає можливість оптимізувати та кваліфікувати дані за існуючими параметрами.

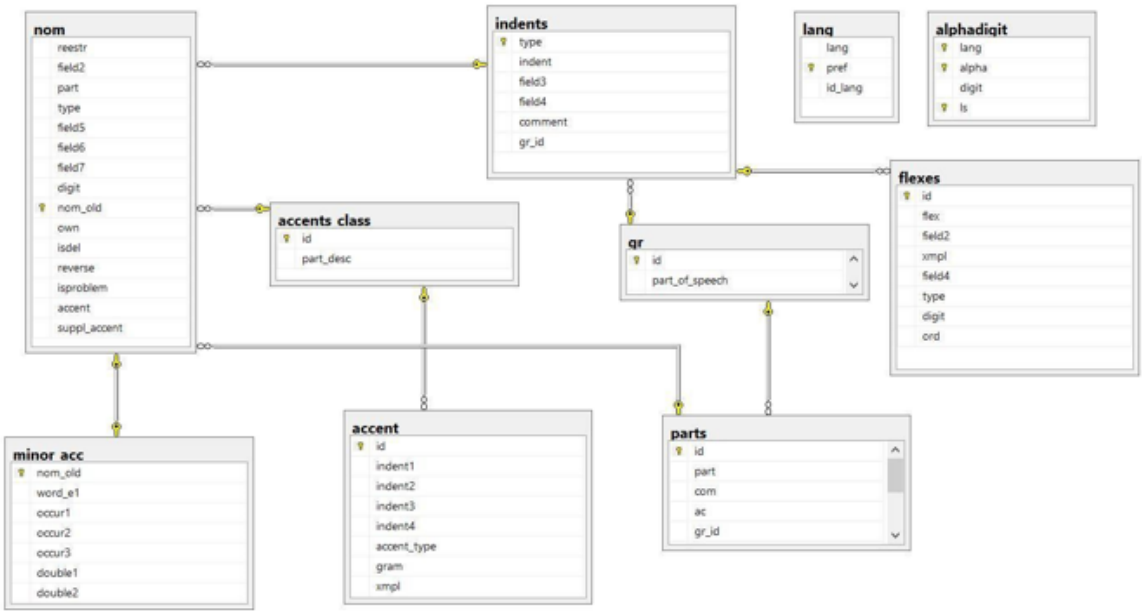


Рис. 3.3. Основні сутності мовного сервера

Запити від клієнта відправляються до користувача всього до 50 раз, що значно прискорює пошукову ефективність. Для формування словника використовуються наступні значення [4-10]:

1) виділення основної частини словоформи;
2) виділення наголосу для визначення частини мови;
3) встановлення відмітки в таблиці словника для того, щоб однозначно ідентифікувати певний клас.

У даній роботі було використано вже існуючий словник, а також було виконано завантаження даних з інших словників для оптимізації функціональності програмного продукту. Для початку було створено адаптер, який довзоляє отримати доступ до інформації з бази даних. Далі було використано сортування за допомогою списку наявних слів, що надає можливість прискорити проведення текстового аналізу.

Для перевірки правильності орфографічного написання тексту в документі було використану наступні методи:

1) запис вихідної інформації;

2) проведення пошуку тексту та порівняння з інформацією у словнику;
3) проведення корекції згідно з невідомими словами для словника;
4) внесення додаткових змін за необхідності – при цьому враховується морфологічне походження мовних одиниць.

При проведенні аналізу вся інформація зберігається у кеші зі списку. Пошук починається з початку документа з пешого слову. Після цього починається обробка та поріняння частин тексту. Для кожного слова враховуються всі основні параметри, що залежить від типу, частини мови та інших даних, які зберігаються у масиві. Додаткова інформація може потребувати використання корекцій. Серед них виділяють аналіз кандидатів, селекцію та складання моделі мовної одиниці та можливих помилок. Механізм відбору полягає у знаходженні найбільш ідентичних мовних одиниць. При цьому враховуються можливі зменні невідомого слова, що впливають на ефективність подальшого аналізу.

Якщо при проведенні аналізу було досягнуто невідоме слово, то не враховуються використовувані діагностичні модулі. При цьому видається сповіщення про помилку, що надає можливість провести класифікацію відповідно до потреб користувача.

Після отримання повідомлення клієнт має право виконати одну із дій. При цьому отримується словникова одиниця, що переміщається до місця розташування курсору у тексті. При пустому результати утворються додаткові суфікси, а також запитуються необхідні дані. Необхідно також обов’язкове використання іменника для формування запитів з інформацією. Вихідні дані сортуються, після чого відправляються до кінцевого користувача. Він має право зберегти дані або використати їх для виконання іншої пошукової інформції [4-10].

3.3. Аналіз результатів

Всі основні характеристики та параметри ефективності зібрані в програмному забезпеченні Visual Studio Code, що надає можливість користувачу побачити перелік основних дій та команд, які доступні для обробки.

Тривалість виконання механічних дій не займає багато часу, що надає можливість швидко обрати необхідну команду. При цьому можливе прискорення швидкості обробки інформаціх за допомогою клавіатури у порівнянні з мишею. Проблемі з роботою розширення можуть виникнути за рахунок додаткового завантаження з серверу, що надає певне системне навантаження на програмний модуль. Повне завантаження може зайняти набагато більше часу – до двадцяти секунд. Продуктивність залежить від особливостей конкретного пристрою та швидкості обробки даних за допомогою процесора (рис. 3.4).

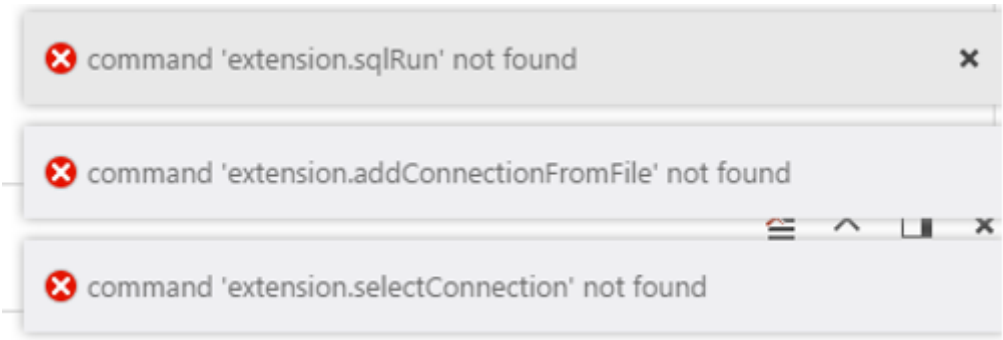


Рис. 3.4. Помилка при виборі певної команди

Модуль розумного доповнення кожен раз повертає масив після проведення аналізу мовної одиниці. Модуль розумного доповнення необхідний для того, чтоб спростити пошук та введення інформції клієнтом у встановлене поле. Чим більша лексична одиниця використовується для аналізу, тим скоріше буде виконана підстановка (рис. 3.5).

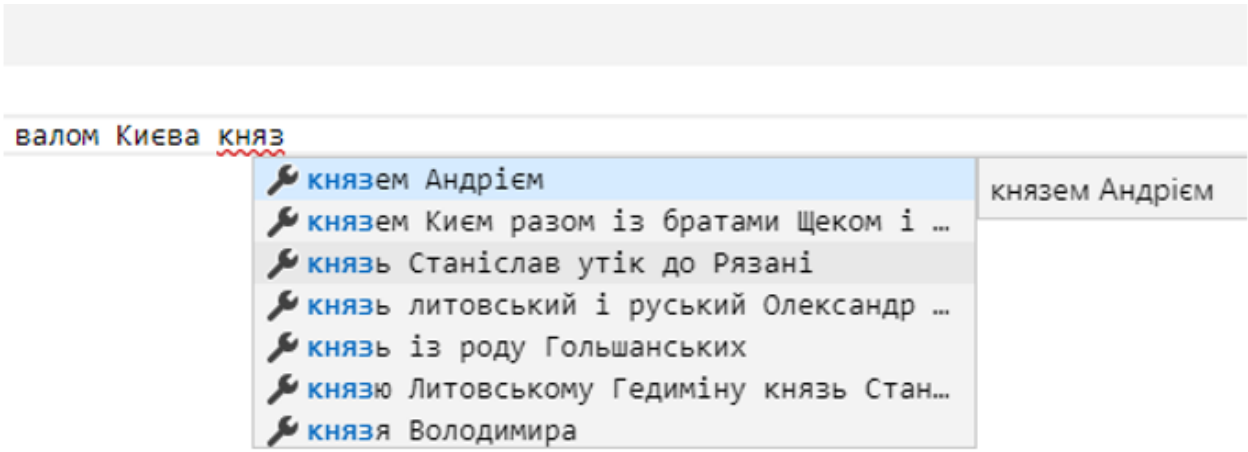


Рис. 3.5. Робота модуля доповнення

Результат роботи показує ефективність роботи мовного клієнта та модуля доповнення у даному випадку.

Ідея протоколу мовного сервера (LSP) полягає в стандартизації протоколу для взаємодії інструментів і серверів, щоб один мовний сервер можна було повторно використовувати в кількох інструментах розробки, а інструменти могли підтримувати мови з мінімальними зусиллями.

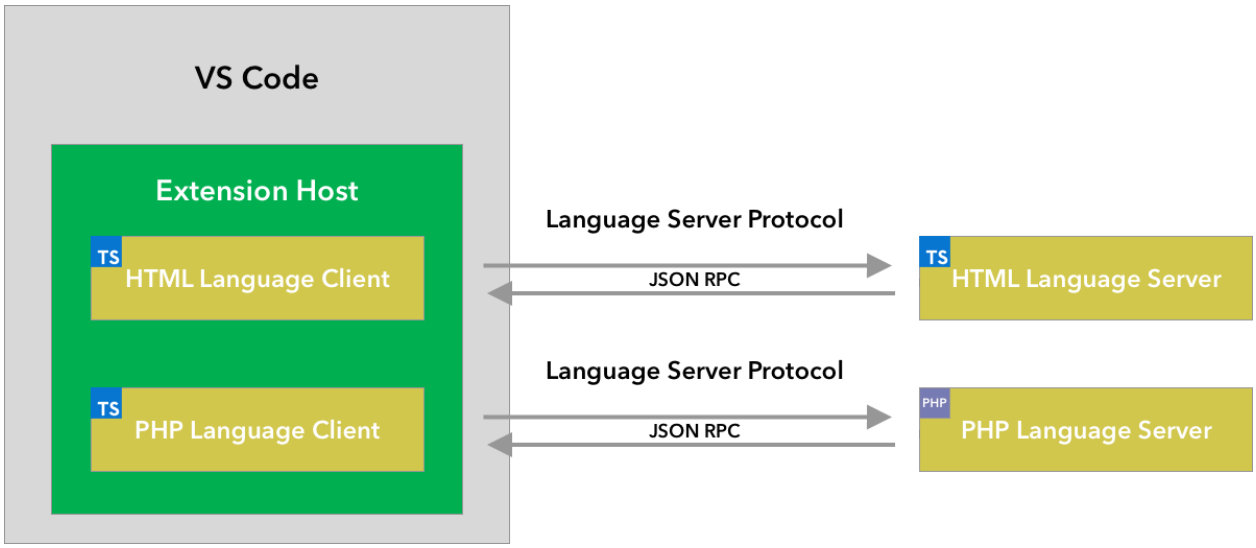


Рис. 3.6. Протокол мовного сервера

Мовний сервер працює як окремий процес, а засоби розробки спілкуються з сервером за допомогою мовного протоколу через JSON-RPC.

3.4. Висновки до розділу 3

У третьому розділі магістерської дисертації було розглянуто та надано приклади розробки мовного клієнта та сервера. Описано основні функції та характеристики програмної реалізації. Описано методи підтримки працездатності системи. Розглянуто основні способи обробки запитів користувача. Описано взаємодію програмних модулів. Надано повний опис розробки діагностичного модуля та використання розумного розширення для розробки програмного продукту.

4. СТАРТАП-ПРОЄКТ ЗА ТЕМАТИКОЮ ДОСЛІДЖЕННЯ

Для створення програмного продукту було використано середовище розробки Visual Studio Code. У якості мови програмування було вирішено використовувати TypeScript. Таке рішення було обгрунтовано одразу кількома параметрами.

TypeScript – це типізований наднабір JavaScript, який компілюється у звичайний JavaScript. Він пропонує класи, модулі та інтерфейси, які допоможуть вам створити надійні компоненти.

Visual Studio Code включає підтримку мови TypeScript, але не включає компілятор TypeScript, tsc. Щоб перенести вихідний код TypeScript на JavaScript (tsc HelloWorld.ts), потрібно встановити компілятор TypeScript глобально або у робочому просторі.

Найпростіший спосіб інсталювати TypeScript – через npm, диспетчер пакетів Node.js.

У VS Code можна побачити, що надаються певні мовні функції, такі як підсвічування синтаксису та відповідність дужок. Коли було введено текст у редакторі, можна помітити IntelliSense, інтелектуальні завершення коду та пропозиції, надані VS Code та мовним сервером TypeScript.

Є можливість змінити параметри компілятора TypeScript, додавши файл tsconfig.json, який визначає параметри проекту TypeScript, такі як параметри компілятора та файли, які мають бути включені.

Щоб використовувати tsconfig.json для решти цього підручника, викличте tsc без вхідних файлів. Компілятор TypeScript знає, як переглянути параметри проекту та параметри компілятора у вашому tsconfig.json.

Необхідно додати файл tsconfig.json, який встановлює параметри компіляції в ES5 та використовує модулі CommonJS.

Налагодження TypeScript підтримує карти джерел JavaScript. Щоб створити вихідні карти для файлів TypeScript, необхідно скомпілювати з

параметром --sourcemap або встановити для властивості sourceMap у файлі tsconfig.json значення true.

Вбудовані вихідні карти (мапа джерела, де вміст зберігається як URL-адреса даних замість окремого файлу) також підтримуються, хоча вбудовані джерела ще не підтримуються.

4.1. Суть розробки програмного проєкту за тематикою дослідження

Обробка природної мови та штучний інтелект – технологія штучного інтелекту для бізнесу стає все більш популярною темою і майже неминучою для більшості компаній. Він здатний автоматизувати підтримку, покращувати роботу клієнтів і аналізувати відгуки. Хоча впровадження технології AI може здатися страхітливим, це не повинно бути.

Обробка природної мови (NLP) – це форма ШІ, яку легко зрозуміти та почати використовувати. Це також може зробити багато для просування вашого бізнесу вперед.

Обробка природної мови (NLP) описує взаємодію між людською мовою та комп’ютером. Це технологія, якою багато людей користуються щодня і яка існує роками, але часто сприймається як належне.

Кілька прикладів НЛП, які люди використовують щодня:

1. Перевірка орфографії.
2. Автозавершення.
3. Голосові текстові повідомлення.
4. Спам-фільтри.
5. Пов’язані ключові слова в пошукових системах.
6. Siri, Alexa або Google Assistant.

У будь-якому випадку, комп’ютер може визначити відповідне слово, фразу або відповідь за допомогою контекстних підказок, так само, як і будь-яка людина. Концептуально це досить проста технологія.

НЛП перевершує людей у кількості мов і даних, які він може обробити. Тому його потенційне використання виходить за межі наведених вище прикладів і робить можливими завдання, на виконання яких працівникам знадобилися місяці чи роки.

4.2. Необхідність розробки проєкту

Обробка природної мови включає багато різних методів інтерпретації людської мови, починаючи від статистичних методів і методів машинного навчання до підходів на основі правил і алгоритмів. На практиці необхідно використовувати широкий спектр підходів, оскільки текстові та голосові дані дуже різняться, як і практичні застосування.

Основні завдання НЛП включають токенізацію та синтаксичний аналіз, лемматизацію/стеммінг, тегування частини мови, виявлення мови та ідентифікацію семантичних зв’язків.

Загалом, завдання НЛП розбивають мову на коротші елементарні частини, намагаються зрозуміти взаємозв’язки між частинами та досліджувати, як частини працюють разом, щоб створити зрозумілу мовну одиницю.

Ці основні завдання часто використовуються в можливостях НЛП вищого рівня, наприклад:

1. Категоризація вмісту. Підсумок документа на основі лінгвістики, включаючи пошук та індексування, сповіщення про вміст і виявлення дублювання.
2. Відкриття теми та моделювання. Точне встановлення значення й теми в текстових колекціях та застосовуйте до тексту розширені аналітичні засоби, як-от оптимізація та прогнозування.
3. Контекстне вилучення. Автоматично витягувати структуровану інформацію з текстових джерел.

4. Аналіз настроїв. Визначення настрою чи суб’єктивних думок у великих обсягах тексту, включаючи середні настрої та аналіз думок.
5. Перетворення мовлення в текст і перетворення тексту в мовлення. Перетворення голосових команд у письмовий текст, і навпаки.
6. Узагальнення документа. Автоматичне створення конспектів великих об’єктів тексту.
7. Машинний переклад. Автоматичний переклад тексту або мовлення з однієї мови на іншу.

У всіх цих випадках головною метою є введення необроблених мов і використання лінгвістики та алгоритмів для перетворення або збагачення тексту таким чином, щоб він приніс більшу цінність.

Обробка природної мови йде рука об руку з аналітикою тексту, яка підраховує, групує та класифікує слова, щоб витягти структуру та значення з великих обсягів вмісту. Аналітика тексту використовується для дослідження текстового вмісту та отримання нових змінних із необробленого тексту, які можна візуалізувати, фільтрувати або використовувати як вхідні дані для прогнозних моделей або інших статистичних методів.

НЛП і текстова аналітика використовуються разом для багатьох додатків, зокрема:

1. Слідча знахідка. Можна визначити закономірності та підказки в електронних листах або письмових звітах, щоб допомогти виявляти та розкривати злочини.
2. Предметна експертиза. Класифікація вмісту за значущими темами, щоб можна було вжити заходів і виявити тенденції.
3. Аналітика соціальних мереж. Можливість відстежувати обізнаність і настрої щодо конкретних тем і визначте ключових осіб, які впливають на них.

4.3. Аналіз можливостей запуску проєкту для використання користувачами

Найбільшою перевагою НЛП для бізнесу є здатність технології виявляти та обробляти величезні обсяги текстових даних у цифровому світі, в тому числі; платформи соціальних мереж, онлайн-огляди, новини тощо.

Крім того, збираючи та аналізуючи бізнес-дані, НЛП може запропонувати підприємствам цінну інформацію про ефективність бренду. Крім того, моделі НЛП можуть виявляти будь-які проблеми, що зберігаються, та вживати необхідних заходів для пом'якшення для покращення продуктивності.

Google мовлення до тексту може досягти всього цього, навчаючи машини розуміти людську мову швидшим, точнішим і послідовним способом, ніж людські агенти. Технологія здатна послідовно контролювати та обробляти дані. Це допомагає брендам залишатися в курсі своєї присутності в Інтернеті та не зазнавати невідповідностей.

Застосування обробки природної мови для бізнесу має кілька основних напрімків, тому розроблювані програмні розширення за допомогою Visual Studio Code та TypeScript мають практичну цінність.

Навіть бренди, які не чули про обробку природної мови, повинні знати про чат-ботів та їх застосування в службі підтримки клієнтів. Чат-боти – це інструменти на основі штучного інтелекту, які можуть вести розмови з людьми на веб-сайтах або в мобільних додатках. Технологія штучного інтелекту, що забезпечує роботу чат-ботів, – це НЛП, яка дає змогу цим інструментам розуміти, аналізувати та відповідати на запити клієнтів миттєво цілодобово, без вихідних.

Сьогодні чат-боти відіграють роль менеджерів з обслуговування клієнтів, відповідаючи на запити клієнтів щодо конкретних тем із заздалегідь визначеними відповідями. Щоб додати до своєї корисності, чат-боти також можуть запропонувати конкретну підтримку клієнтів, як-от

бронювання послуги, надання посилання на детальні інструкції або пошук потрібних продуктів. По суті, застосування чат-ботів нескінченні і залежать від унікальних потреб бізнесу.

Насправді, завдяки прогресу в алгоритмах машинного навчання, чат-боти сьогодні можуть виконувати аналіз настроїв, виявляти наміри користувачів і належним чином реагувати.

Підприємствам потрібно отримувати прибуток, а скорочення витрат є найкращим способом оптимізації доходів. Обробка природної мови (NLP) виявилася високоефективною технологією для компаній, яка дозволяє заощадити час і гроші при оптимізації бізнес-процесів.

Згідно зі статистикою, компанії можуть заощадити до 30% на підтримці клієнтів, впровадивши автоматичних чат-ботів як агентів підтримки. Використовуючи чат-ботів на основі НЛП, компанії звільняють час для співробітників. Це дозволяє їм залишатися зосередженими на таких творчих завданнях, як залучення клієнтів для залучення потенційних клієнтів і продажів.

Крім того, моделі НЛП здатні вдосконалюватися та ставати точнішими з часом без будь-яких додаткових інвестицій. Це досягається, коли моделі НЛП можуть працювати з більшою кількістю даних, що автоматично покращує продуктивність і точність моделей НЛП.

Сьогодні користувачі генерують величезний обсяг даних на цифрових носіях. Дані генеруються на різних платформах, тому бренди не можуть контролювати та активно реагувати.

Інструменти обробки природної мови (NLP) надають підприємствам ефективний інструмент для послідовного моніторингу величезної кількості створеного користувачами вмісту на всіх цифрових платформах. Використовуючи інструменти НЛП, компанії можуть активно залучати клієнтів, відповідаючи на їхні запити чи занепокоєння. Крім того, інструменти НЛП допомагають компаніям застосовувати різні фільтри

вмісту, як-от блокування небажаного вмісту або впровадження фільтрів спаму на веб-сайтах чи сторінках у соціальних мережах.

Моделі НЛП також використовуються підприємствами для підтримки якості контенту на форумах. Це гарантує, що немає шкідливих зворотних посилань або небажаної реклами. Він також може виявляти «заборонені» слова або вульгаризм, щоб автоматично фільтрувати створений користувачами вміст, який можна класифікувати як ворожі висловлювання.

Підвищення коефіцієнта конверсії часто є найактуальнішою проблемою для керівників маркетингу, які використовують усілякі стратегії для перетворення відвідувачів у клієнтів. Вищі коефіцієнти конверсії не тільки призводять до підвищення доходів для бізнесу, але й знижують вартість залучення клієнтів.

Моделі НЛП довели свою ефективність для оптимізації конверсій. Інструменти на основі НЛП, такі як чат-боти, автозавершення тексту та функція попереднього пошуку; значно покращує загальний досвід клієнтів. Це, у свою чергу, дозволяє компаніям завоювати довіру клієнтів і покращувати коефіцієнт конверсії.

4.4. Розробка стратегічної ефективності проєкту

Здатність розуміти, аналізувати й маніпулювати мовою робить бізнес-додатки не тільки крутими, але, що найголовніше, значно ефективнішими. У нашій статті ви дізнаєтеся, чому штучний інтелект настільки важливий для маркетингу і як технологію НЛП можна використовувати в маркетинговому програмному забезпеченні, щоб покращити:

1) ефективність обслуговування клієнтів;
2) вести кваліфікаційні процеси;
3) аналіз настроїв коментарів;
4) оптимізація інструментів SEO;
5) планування контент-стратегії.

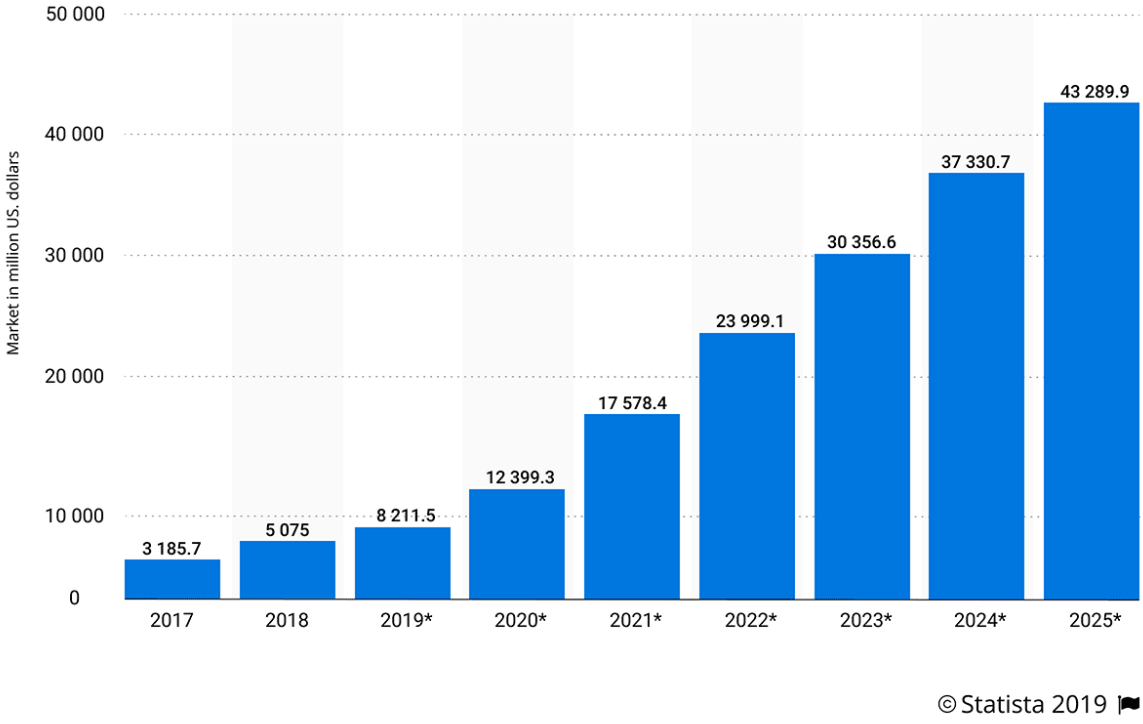


Рис. 4.1. Збільшення ринку природномовних текстів

Згідно зі звітом McKinsey, найбільше застосування штучного інтелекту є найвищим у розробці продуктів і послуг (24% – нові вдосконалення продуктів на основі штучного інтелекту, 21% оптимізація наявних функцій), а також функції обслуговування (24% – оптимізація роботи сервісу, 19% – прогностична послуга та втручання). Штучний інтелект також використовується в управлінні, фінансах, маркетингу, HR та багатьох інших відділах, щоб приймати більше рішень на основі даних і автоматизувати різні процеси (рис. 4.1).

MartTech – це галузь, яка поєднує маркетинг і технології, щоб забезпечити маркетинговий успіх своїх клієнтів, використовуючи потужні маркетингові інструменти та методи для підвищення ефективності їхніх маркетингових кампаній. Це все про використання даних і новітніх технологій для підвищення впізнаваності бренду.

Згідно зі звітом Martech: 2020 and Beyond, понад 75% брендів використовують рішення MarTech для маркетингу електронної пошти та

соціальних мереж. Бізнес-інструменти цього типу також використовуються для CRM, управління контентом та аналітики більш ніж 50% компаній, які входять до списку. Насправді, компанії інвестують у рішення MarTech не лише для того, щоб зробити маркетингові кампанії більш ефективними, але й для забезпечення найкращого досвіду користувачів.

У 2021 році більшість опитаних маркетологів (52%) стверджують, що рішення AI дуже важливі для успіху маркетингу. Відповідно до звіту про стан маркетингу AI за 2021 рік, найважливішими результатами впровадження AI в маркетингові системи є покращення продуктивності (41%), отримання кращої аналітики на основі даних (40%), можливість персоналізації роботи користувачів (38%), економія часу на повторювані завдання (35%) і підвищення рентабельності інвестицій (34%).

4.5. Маркетингова програма проєкту

Ось кілька прикладів того, як НЛП можна використовувати, щоб зробити маркетинговий інструмент кращим за інші. Розроблюваний програмний продукт можна буде використовувати для того, щоб покращити робочу ефективність рішень.

4.5.1. Обслуговування клієнтів

Використовуючи аналітику даних, уповноважену обробкою природних мов, маркетингові системи здатні збирати корисні дані більш ефективно. Завдяки додаткової бізнес-інформації маркетингове програмне забезпечення дає користувачам кращі шанси охопити клієнтів за допомогою нового вмісту та оголошень. Доступ до конкретних даних дає змогу створити більш точні персони покупців, підвищуючи тим самим точність націлювання реклами.

НЛП підтримує аналітику, надаючи важливу інформацію. Це допомагає компаніям вимірювати досвід клієнтів і покращувати системи

рекомендацій. Ця технологія виявилася дуже корисною в обслуговуванні клієнтів, оскільки її можна використовувати для аналізу вхідних заявок на підтримку, щоб їх можна було визначити пріоритетами та надіслати потрібним фахівцям, тим самим швидше вирішуючи проблеми.

4.5.2. Ведуча кваліфікація

Досягнення в НЛП змінили бізнес, покращивши можливості чат-ботів. Чат-боти щодня обслуговують мільйони запитів щодо обслуговування клієнтів у різних галузях – згідно з опитуванням Hubspot, 47% клієнтів зацікавлені в купівлі продуктів через ботів. Вони здатні швидко відповідати на запитання клієнтів і залучати їх протягом усього процесу покупки товару чи послуги. Як у тому, що? Завдяки НЛП боти здатні не тільки обробляти запити, а й збирати важливу інформацію про відвідувачів, аналізувати настрій і потреби клієнтів.

Чат-боти NLP гарантують, що клієнти отримують негайну увагу та кваліфікують потенційних клієнтів, тому експерти з продажу можуть втрутитися лише тоді, коли зусилля з продажу найімовірніше будуть успішними.

4.5.3. Аналіз настроїв

Вимірювання поінформованості про бренд має вирішальне значення, коли справа доходить до планування вашого контенту та комунікаційної стратегії. Людська мова складна. Рішенням, заснованим на старіших технологіях, важко дізнатися, що саме думають клієнти про компанію та її продукти чи послуги без НЛП.

Маркетингові інструменти, що використовуються для соціального прослуховування та соціального відстеження, можуть допомогти маркетологам визначити, що люди пишуть у соціальних мережах і як вони ставляться до брендів, але саме НЛП дає змогу програмному забезпеченню

аналізувати письмові думки та розуміти наміри користувачів, часто приховані. за словами самих коментарів.

4.5.4. Інструменти SEO оптимізації

За допомогою правильних інструментів SEO маркетологи, розробники контенту та спеціалісти з SEO можуть дізнатися, чи є веб-сайт добре оптимізований та видимий у браузері потенційним клієнтам. Існує кілька способів надіслати одне й те саме повідомлення. Технологія НЛП може бути реалізована для визначення альтернативних або споріднених ключових слів, які можна використовувати для оптимізації вмісту на основі фактичних пошуків користувачів Інтернету.

Існують інструменти для оптимізації SEO, які не тільки допомагають користувачам дізнатися, які фрази мають найбільший потенціал, але й надають їм додаткові поради щодо конкурентів та способів їх випередження.

4.5.5. Стратегія планування контенту

SEO оптимізація – це лише одна частина контент-маркетингу. Бути видимим для користувачів Інтернету – це одне, а фактично вміти залучати їх – це зовсім інше. Окрім визначення найпопулярніших ключових слів, пов’язаних із кожною галуззю, НЛП можна використовувати для визначення найактуальніших тем, про які даний бренд повинен писати, щоб зацікавити клієнтів у своїх продуктах та послугах.

Найсучасніші маркетингові інструменти використовують обробку природних мов, щоб надати бізнес-користувачам поради щодо того, про що писати та говорити, щоб привернути найбільшу увагу.

4.6. Висновки до розділу 4

В четвертому розділі магістерської дисертації було розглянуто суть та особливості розробки програмного продукту. Зроблено огляд технологічних характеристик та необхідності розробки з метою практичного використання. Було проаналізовано основні сфери діяльності, де можливо розглянути дану розробку та реалізувати її на практиці. Було розроблено стратегічну ефективність та маркетингову програму. Наведено основні приклади та цифри, що говорять про раціональність програмної розробки та її практичну ефективність.

ВИСНОВКИ

Метою даної магістерської дисертації було дослідити та проаналізувати методи аналізу природномовних текстів. Для розробки було обрано розширення, що мають значення для інструментального забезпечення аналізу текстової інформації.

У першому розділі магістерської дисертації було розглянуто основні принципи та методи аналізу природномовних текстів. Розглянуто вже інсуючі засоби та методи, що використовуються на практиці. Виявлено приклади, на які необхідно звернути увагу при реалізації модулів для написання природномовних текстів. Використання Visuao Studio Code та його розширень для створення програмного продукту. Аналіз інформаційних потреб користувачів для розробки та спрощення аналізу. Формулювання теми та мети розробки майбутнього програмного продукту на основі отриманих з аналізу даних.

У другому розділі магістерської дисертації було розглянуто особливості використання Visual Studio Code та засобів інтегрованого середовища розробки. Оглянуто використовувані програмні засоби: серверну частину, особливості та характеристики файлу маніфесту, архітектуру Visual Studio Code, використання модуля публікації для публікації створюваного розширення в магазині. Описано математичну модель програмного продукту. Розглянуто та розроблено основні вимоги до створюваного програмного продукту. Зроблено огляд основних засобів архітектури розробрюваних засобів природномовних текстів.

У третьому розділі магістерської дисертації було розглянуто та надано приклади розробки мовного клієнта та сервера. Описано основні функції та характеристики програмної реалізації. Описано методи підтримки працездатності системи. Розглянуто основні способи обробки запитів користувача. Описано взаємодію програмних модулів. Надано повний опис

розробки діагностичного модуля та використання розумного розширення для розробки програмного продукту.

У четвертому розділі магістерської дисертації було розглянуто суть та особливості розробки програмного продукту. Зроблено огляд технологічних характеристик та необхідності розробки з метою практичного використання. Було проаналізовано основні сфери діяльності, де можливо розглянути дану розробку та реалізувати її на практиці. Було розроблено стратегічну ефективність та маркетингову програму. Наведено основні приклади та цифри, що говорять про раціональність програмної розробки та її практичну ефективність.

Отже, у ході розробки магістерської дисертації було розроблино мовний клієнт та мовний сервер, а також програмні розширення, що можуть використовуватись у якості надбудови різних проектів на практиці.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Анисимов А. В. Система обработки текстов на естественном языке [Текст] / Анисимов А. В., Марченко А. А., «Штучний інтелект». — 2002. — № 4.
2. Глібовець М.М. Штучний інтелект: Підруч. для студ. вищ. навч. Закладів [Текст] / М.М. Глібовець, О.В. Олецький — Київ: Видавничий дім «КМ Академія», 2002. — 336 с.
3. Глинський, Я.М. Штучний інтелект. Інтелектуальні роботи / Я.М.Глинський, В.А. Ряжська В.А. [Текст] — Львів: Деол, 2002. - 168 с.
4. Громов Ю.Ю Интеллектуальные информационные системы и технологии: учебное пособие/Ю.Ю. Громов, О.Г. Иванова, В.В. Алексеев и др. [Текст] — Тамбов: Изд-во ФГБОУ ВПО «ТГТУ», 2013 — 244 с.
5. Кришна Г. Хороший интерфейс – невидимый интерфейс. [Текст] — изд. Питер – 2016. – 256 с.
6. Купер А. Интерфейс. Интерфейс. Основы проектирования взаимодействия. 4-е изд. [Текст] — изд. Питер – 2016. – 720 с.
7. Леоненков А. Самоучитель UML, 2-е издание. [Текст] — БХВ-Петербург – 2007. – 569 с.
8. Малайчук В.П., Мозговой А.В. Математическая дефектоскопия: Монография. [Текст] — Днепропетровск: Системные технологии. – 2005.
9. Марка Д.Д., МакГоуэн К. Методология структурного анализа и проектирования SADT. [Текст] — М.: МетаТехнология – 1993 – 240 с.
10. Прохоренок Н.А. Самое необходимое [Текст] — СПб.: БХВ-Петербург, 2011. – 416 с.
11. Сорока К.О. Основи теорії систем і системного аналізу [Текст] — 2004. — 291 с.

12. Тельнов Ю.Ф. Интеллектуальные информационные системы [Текст] / Ю.Ф. Тельнов. — М.: Московский международный институт эконометрики, информатики, финансов и права, 2004. — 82 с.
13. Тидвелл Дж. Разработка пользовательских интерфейсов. [Текст] — изд. Питер — 2008. — 416 с.
14. Ту Дж., Гонсалес Р. Принципы распознавания образов. [Текст] — М.: Издательство "Мир". – 1978.
15. LinguisticAndInformationSystems [Електронний ресурс]. — Режим доступу до ресурсу: https://github.com/LinguisticAndInformationSystems — Дата доступу: 15.09.2021.
16. How to sell your dataset? [Електронний ресурс]. — Режим доступу до ресурсу: https://towardsdatascience.com/how-to-sell-your-dataset — Дата доступу: 15.09.2021.
17. ISO/IEC 9126-1:2001//Software engineering//Product quality//Part 1: Quality model. [Електронний ресурс]. — Режим доступу до ресурсу: https://www.iso.org/standard/22749.html — Дата доступу: 15.10.2021.
18. ISO/IEC 25010:2011//Systems and software engineering//Systems and software Quality Requirements and Evaluation (SQuaRE)//System and software quality models. [Електронний ресурс]. — Режим доступу до ресурсу: https://www.iso.org/standard/35733.html — Дата доступу: 11.11.2021.
19. L. Goasduff, 2 Megatrends Dominate the Gartner Hype Cycle for Artificial Intelligence, Gartner, Inc., Stanford, CT, USA, 2020. [Електронний ресурс]. — Режим доступу до ресурсу: https://www.gartner.com/smarterwithgartner/2-megatrends-dominate-the-gartner-hype-cycle-for-artificial-intelligence-2020/ — Дата доступу: 11.11.2021.

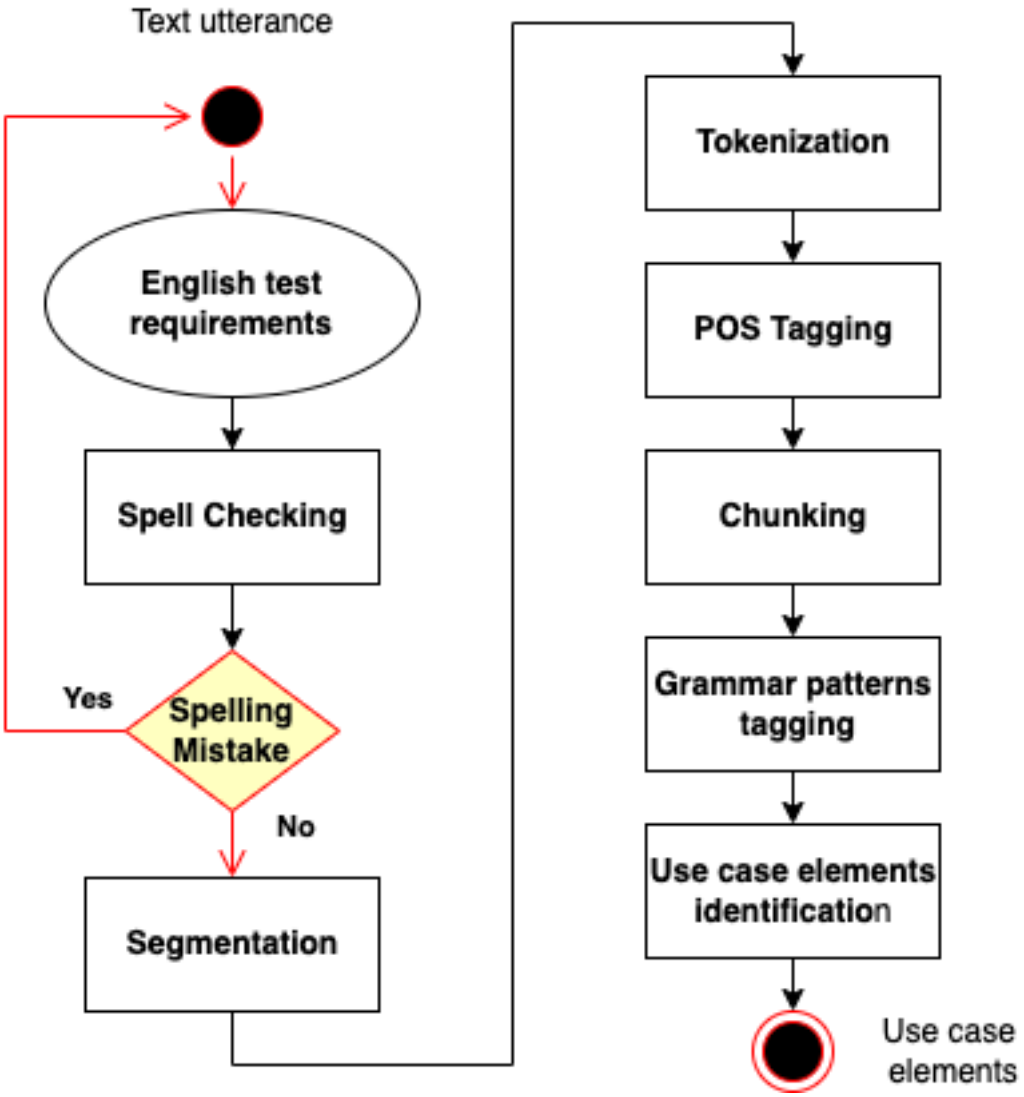
20. L. Wood, Global Chatbot Market Anticipated to Reach $9.4 Billion by 2024 - Robust Opportunities to Arise in Retail & eCommerce, Insider Inc., New York, NY, USA, 2019. [Електронний ресурс]. — Режим доступу до ресурсу: https://markets.businessinsider.com/news/stocks/global-chatbot-market-anticipated-to-reach-9-4-billion-by-2024-robust-opportunities-to-arise-in-retail-ecommerce-1028759508 — Дата доступу: 01.12.2021.
21. Natural Language Processing (NLP) Examples. [Електронний ресурс]. — Режим доступу до ресурсу: https://www.tableau.com/learn/articles/natural-language-processing-examples — Дата доступу: 15.10.2021.
22. UML basics//The class diagram//An introduction to structure diagrams in UML 2 [Електронний ресурс]. — Режим доступу до ресурсу: https://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/index.html — Дата доступу: 28.10.2021.
23. Visual Studio Code. [Електронний ресурс]. — Режим доступу до ресурсу: https://visualstudio.microsoft.com/ru/ — Дата доступу: 10.10.2021.
24. Visual Studio Code//Getting started. [Електронний ресурс]. — Режим доступу до ресурсу: https://code.visualstudio.com/docs — Дата доступу: 10.10.2021.
25. Інтерфейс користувача. [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/ Інтерфейс_користувача — Дата доступу: 28.09.2021.

ДОДАТКИ

Додаток 1

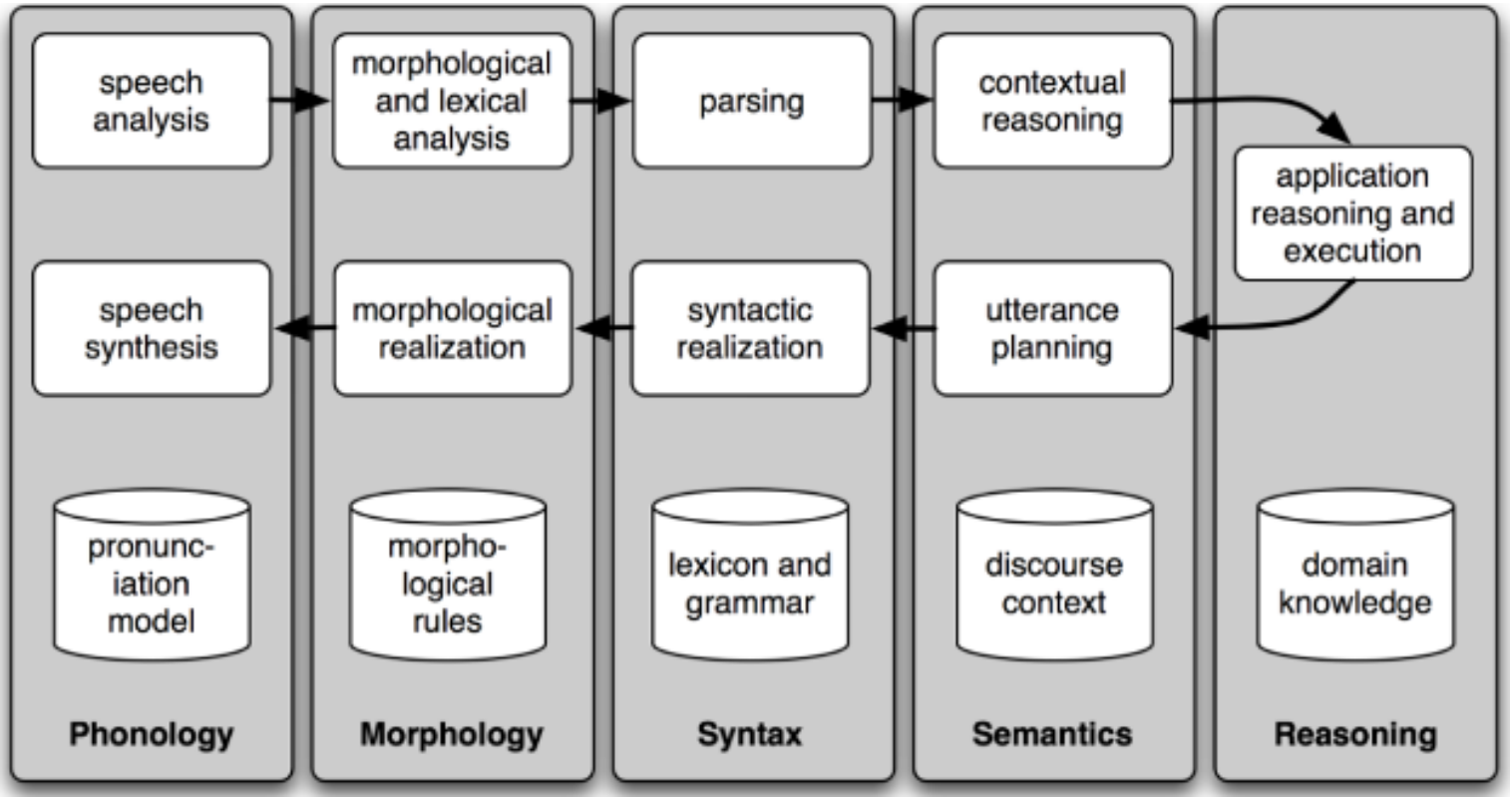
Копії графічних матеріалів

UML діаграма станів системи перевірки орфографії



Мединський Микола, група КП-01мп

Схема аналізу природномовних текстів



Мединський Микола, група КП-01мп

Додаток 2

Лістинг програми

Лістинг 1. client/extension.ts

```
import * as path from 'path';
import {
	workspace as Workspace, window as Window, ExtensionContext, TextDocument, OutputChannel, WorkspaceFolder, Uri
} from 'vscode';

import {
	LanguageClient, LanguageClientOptions, TransportKind
} from 'vscode-languageclient/node';

let defaultClient: LanguageClient;
const clients: Map<string, LanguageClient> = new Map();

let _sortedWorkspaceFolders: string[] | undefined;
function sortedWorkspaceFolders(): string[] {
	if (_sortedWorkspaceFolders === void 0) {
		_sortedWorkspaceFolders = Workspace.workspaceFolders ? Workspace.workspaceFolders.map(folder => {
			let result = folder.uri.toString();
			if (result.charAt(result.length - 1) !== '/') {
				result = result + '/';
			}
			return result;
		}).sort(
			(a, b) => {
				return a.length - b.length;
			}
		) : [];
	}
	return _sortedWorkspaceFolders;
}
Workspace.onDidChangeWorkspaceFolders(() => _sortedWorkspaceFolders = undefined);

function getOuterMostWorkspaceFolder(folder: WorkspaceFolder): WorkspaceFolder {
	const sorted = sortedWorkspaceFolders();
	for (const element of sorted) {
		let uri = folder.uri.toString();
		if (uri.charAt(uri.length - 1) !== '/') {
```

```
                        uri = uri + '/';
                }
                if (uri.startsWith(element)) {
                        return
Workspace.getWorkspaceFolder(Uri.parse(element))!;
                }
            }
            return folder;
      }

      export function activate(context: ExtensionContext) {

            const module = context.asAbsolutePath(path.join('server', 'out',
'server.js'));
            const          outputChannel:          OutputChannel          =
Window.createOutputChannel('lsp-multi-server-example');

            function didOpenTextDocument(document: TextDocument): void {
                // We are only interested in language mode text
                if     (document.languageId     !==     'plaintext'     ||
(document.uri.scheme !== 'file' && document.uri.scheme !== 'untitled')) {
                        return;
                }

                const uri = document.uri;
                // Untitled files go to a default client.
                if (uri.scheme === 'untitled' && !defaultClient) {
                        const  debugOptions  =  {  execArgv:  ["--nolazy",  "--
inspect=6010"] };
                        const serverOptions = {
                                run: { module, transport: TransportKind.ipc },
                                debug: { module, transport: TransportKind.ipc,
options: debugOptions}
                        };
                        const clientOptions: LanguageClientOptions = {
                                documentSelector: [
                                        {    scheme:    'untitled',    language:
'plaintext' }
                                ],
                                diagnosticCollectionName:   'lsp-multi-server-
example',
                                outputChannel: outputChannel
```

```
                        };
                        defaultClient = new LanguageClient('lsp-multi-server-example', 'LSP Multi Server Example', serverOptions, clientOptions);
                        defaultClient.start();
                        return;
                  }
                  let folder = Workspace.getWorkspaceFolder(uri);
                  // Files outside a folder can't be handled. This might depend on the language.
                  // Single file languages like JSON might handle files outside the workspace folders.
                  if (!folder) {
                        return;
                  }
                  // If we have nested workspace folders we only start a server on the outer most workspace folder.
                  folder = getOuterMostWorkspaceFolder(folder);

                  if (!clients.has(folder.uri.toString())) {
                        const debugOptions = { execArgv: ["--nolazy", `--inspect=${6011 + clients.size}`] };
                        const serverOptions = {
                              run: { module, transport: TransportKind.ipc },
                              debug: { module, transport: TransportKind.ipc, options: debugOptions}
                        };
                        const clientOptions: LanguageClientOptions = {
                              documentSelector: [
                                    { scheme: 'file', language: 'plaintext', pattern: `${folder.uri.fsPath}/**/*` }
                              ],
                              diagnosticCollectionName: 'lsp-multi-server-example',
                              workspaceFolder: folder,
                              outputChannel: outputChannel
                        };
                        const client = new LanguageClient('lsp-multi-server-example', 'LSP Multi Server Example', serverOptions, clientOptions);
                        client.start();
                        clients.set(folder.uri.toString(), client);
                  }
            }
```

```
            Workspace.onDidOpenTextDocument(didOpenTextDocument);
            Workspace.textDocuments.forEach(didOpenTextDocument);
            Workspace.onDidChangeWorkspaceFolders((event) => {
                  for (const folder  of event.removed) {
                        const client = clients.get(folder.uri.toString());
                        if (client) {
                              clients.delete(folder.uri.toString());
                              client.stop();
                        }
                  }
            });
      }

      export function deactivate(): Thenable<void> {
            const promises: Thenable<void>[] = [];
            if (defaultClient) {
                  promises.push(defaultClient.stop());
            }
            for (const client of clients.values()) {
                  promises.push(client.stop());
            }
            return Promise.all(promises).then(() => undefined);
      }
```

Лістинг 2. server/index.ts

```
import {
      CompletionList,
      createConnection,
      Diagnostic,
      InitializeParams,
      ProposedFeatures,
      TextDocuments,
      TextDocumentSyncKind
} from 'vscode-languageserver';
import { getLanguageModes, LanguageModes } from './languageModes';
import { TextDocument } from 'vscode-languageserver-textdocument';

// Create a connection for the server. The connection uses Node's IPC as a
transport.
// Also include all preview / proposed LSP features.
const connection = createConnection(ProposedFeatures.all);

// Create a simple text document manager. The text document manager
// supports full document sync only
const documents: TextDocuments<TextDocument> = new
TextDocuments(TextDocument);

let languageModes: LanguageModes;
```

```
connection.onInitialize((_params: InitializeParams) => {
	languageModes = getLanguageModes();

	documents.onDidClose(e => {
		languageModes.onDocumentRemoved(e.document);
	});
	connection.onShutdown(() => {
		languageModes.dispose();
	});

	return {
		capabilities: {
			textDocumentSync: TextDocumentSyncKind.Full,
			// Tell the client that the server supports code
completion
			completionProvider: {
				resolveProvider: false
			}
		}
	};
});

connection.onDidChangeConfiguration(_change => {
	// Revalidate all open text documents
	documents.all().forEach(validateTextDocument);
});

// The content of a text document has changed. This event is emitted
// when the text document first opened or when its content has changed.
documents.onDidChangeContent(change => {
	validateTextDocument(change.document);
});

async function validateTextDocument(textDocument: TextDocument) {
	try {
		const version = textDocument.version;
		const diagnostics: Diagnostic[] = [];
		if (textDocument.languageId === 'html1') {
			const modes =
languageModes.getAllModesInDocument(textDocument);
			const latestTextDocument =
documents.get(textDocument.uri);
			if (latestTextDocument && latestTextDocument.version ===
version) {
				// check no new version has come in after in after
the async op
				modes.forEach(mode => {
					if (mode.doValidation) {

	mode.doValidation(latestTextDocument).forEach(d => {
								diagnostics.push(d);
							});
					}
				});
				connection.sendDiagnostics({ uri:
latestTextDocument.uri, diagnostics });
			}
		}
	} catch (e) {
		connection.console.error(`Error while validating
${textDocument.uri}`);
		connection.console.error(String(e));
```

```
	}
}

connection.onCompletion(async (textDocumentPosition, token) => {
	const document =
documents.get(textDocumentPosition.textDocument.uri);
	if (!document) {
		return null;
	}

	const mode = languageModes.getModeAtPosition(document,
textDocumentPosition.position);
	if (!mode || !mode.doComplete) {
		return CompletionList.create();
	}
	const doComplete = mode.doComplete!;

	return doComplete(document, textDocumentPosition.position);
});

// Make the text document manager listen on the connection
// for open, change and close text document events
documents.listen(connection);

// Listen on the connection
connection.listen();
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

Спосіб та програмне забезпечення інтелектуального асистування при написанні природномовних текстів

Доповідач: Мединський Микола Сергійович
Науковий керівник: к.т.н., доцент каф. ПЗКС ФПМ, Заболотня Т. М.

Київ – 2021

Актуальність роботи

Тема ”Спосіб та програмне забезпечення інтелектуального асистування при написанні природомовних текстів” є актуальною адже:

Проблема асистування створенню природномовних 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 природномовних текстів.

Об’єкт та предмет дослідження

Об’єкт дослідження: Автоматизоване оброблення природомовних текстів

Предмет дослідження: Способи та програмні засоби для інтелектуального редагування природомовних текстів.

Мета дослідження

Метою даного дослідження є вдосконалення інтелектуального асистування при написані природомовних текстів з точки зору збільшення швидкості надання мовної підтримки, зменшення витраченого часу при написанні групи лексем та зменшення обсягу повідомлення при взаємодії з мовним сервером через побудову підсистеми інтелектуального редактора природномовних текстів та створення мовного сервера.

Спосіб інтелектуального асистування при написанні природномовних текстів

1. Мовний клієнт встановлює з’єднання з мовним сервером за допомогою розробленого розширення, створюється процесор документу, що оновлюється при подальших діях користувача.

2. Лексичний аналізатор мовного сервера розбиває вхідну послідовність на масив лексем для подальшого дослідження.

3. Синтаксичний аналізатор мовного сервера аналізує вхідну послідовність символів з метою розбору граматичної структури з перевіркою відповідності граматичним правилам для подальшого дослідження.

4. В залежності від результатів аналізаторів відбувається пошук через адаптер для роботи зі словником, формування варіантів написання слова (генерація підказок), доповнення речення.

5. Результати своєї роботи мовний сервер надсилає через процесор документу до мовного клієнту і відображає користувачу різні можливості за допомогою відповідних модулей підказок, доповнення та діагностики.

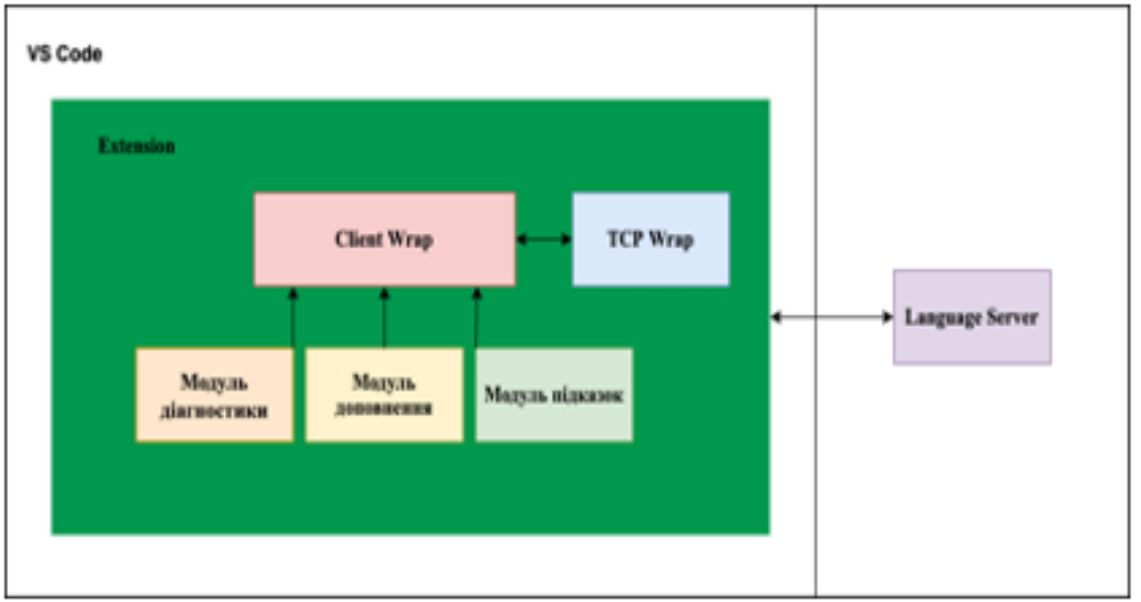
Структура підсистеми інтелектуального редактора природномовних текстів

Мовний клієнт

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

Структура мовного клієнта

1. Модуль доповнення тексту
2. Модуль діагностики тексту
3. Модуль генерування підказок

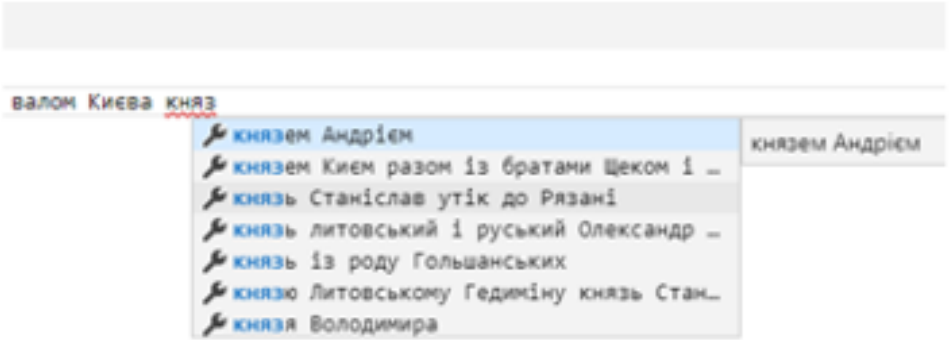


Модуль діагностики тексту

Модуль діагностики повинен надавати користувачу інформацію про помилки вводу тексту запиту, наприклад, невірний символ.

Модуль доповнення тексту

Модуль доповнення тексту повинен надавати список можливих варіантів продовження тексту, який був введеним користувачем, що забезпечить прискорення написання тексту.



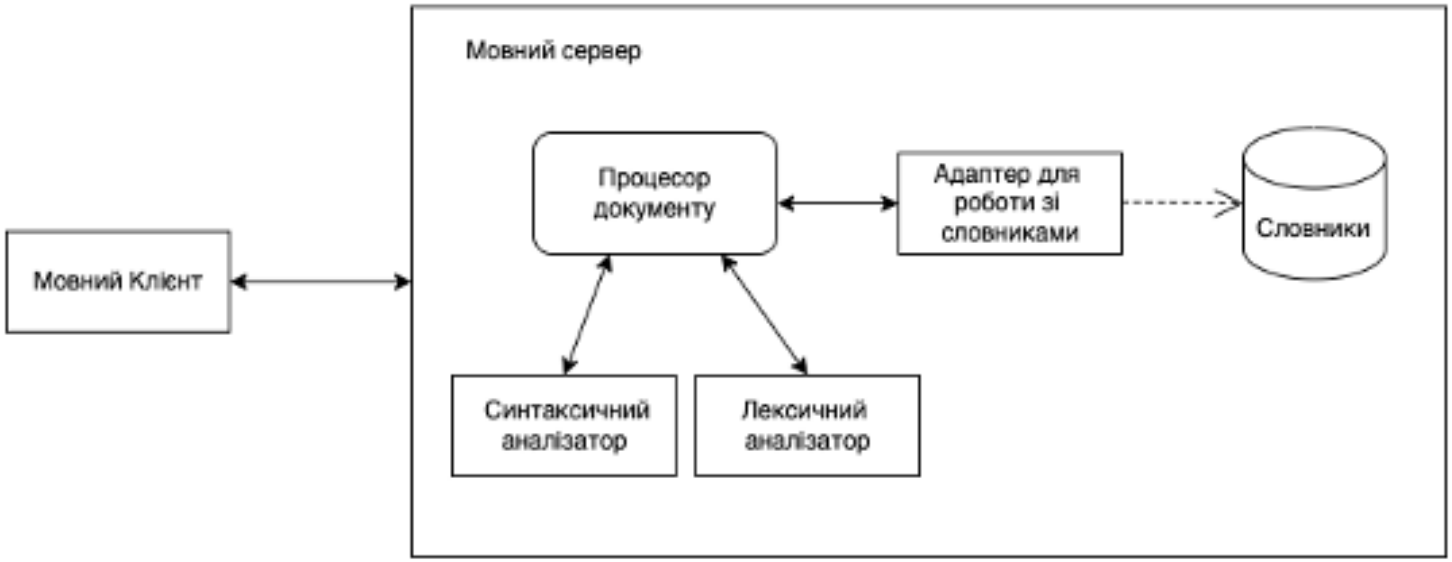
Модуль генерування підказок

Модуль підказки надає додаткову інформацію про лексему або помилку при наведенні курсору на відповідну лексему.

Мовний сервер

Мовний сервер надає функціональну підтримку підсистеми обробки текстів, що забезпечує зменшення навантаження на мовні клієнти та створення окремого модуля, що надає однаковий функціонал різним мовним клієнтам.

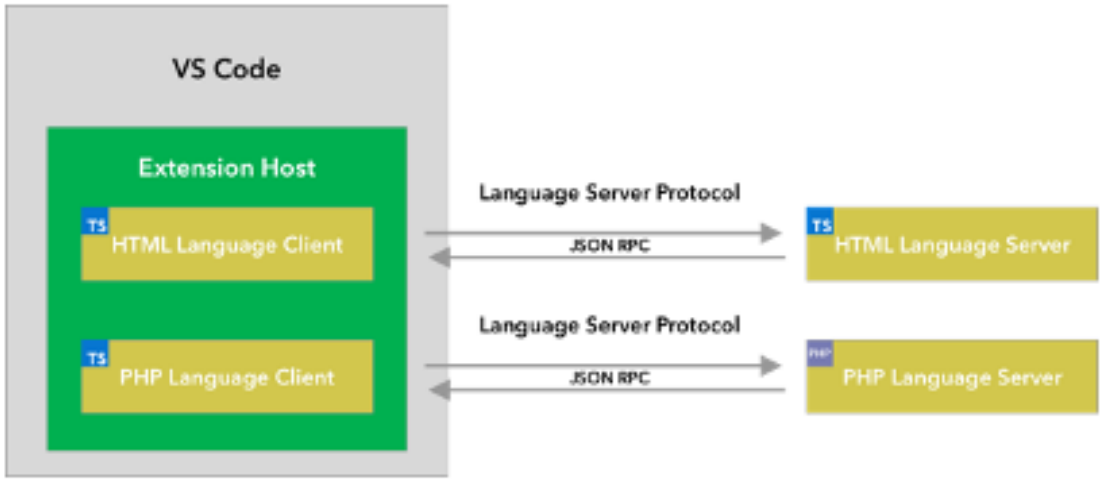
Архітектура мовного сервера



Базові модулі для мовного сервера

1. Лексичний аналізатор
2. Синтаксичний аналізатор
3. Адаптер для роботи з словником
4. Процесор документу

Протокол мовного сервера

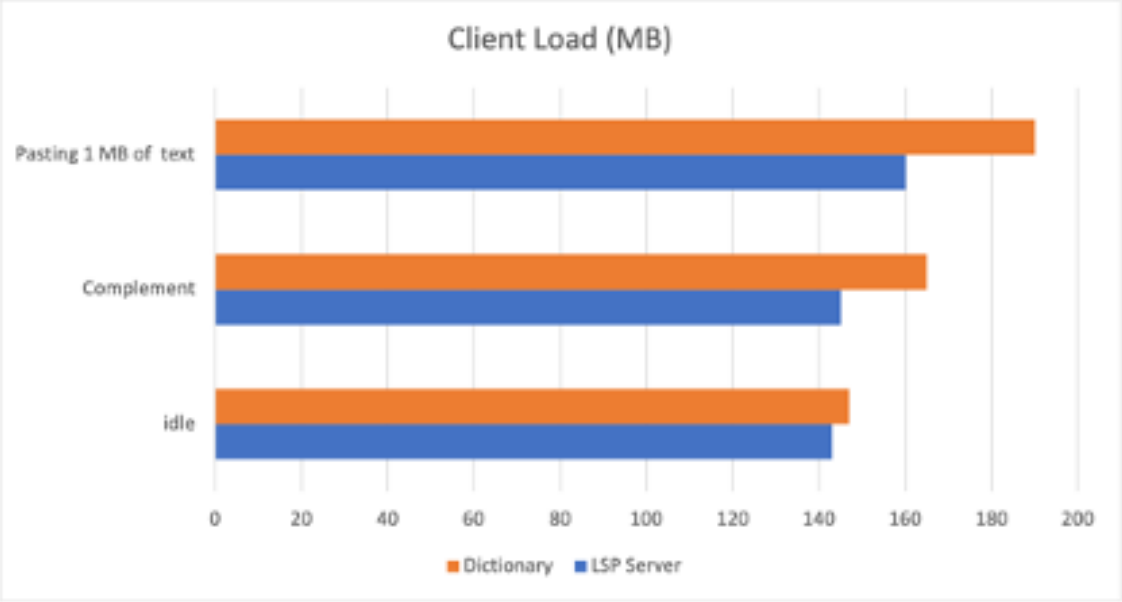


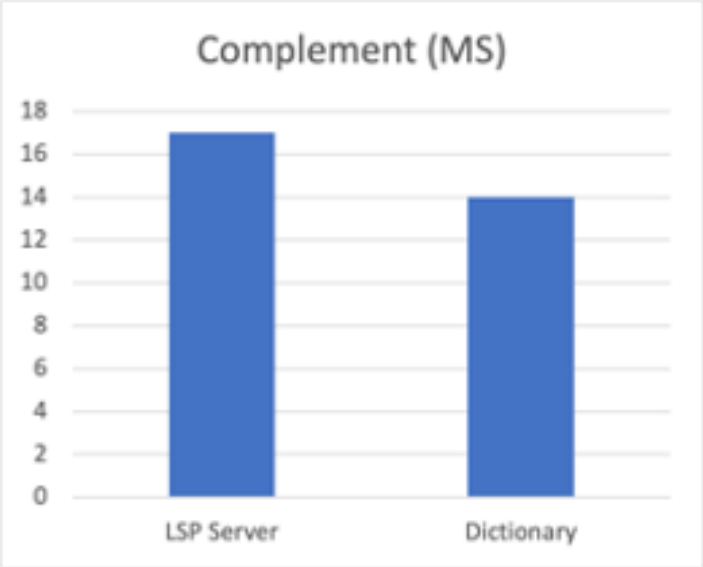
— це контракт, набір правил, який підтримується мовним сервером і використовується клієнтами для користування послугами сервера.

Переваги LSP

Використані технології

Порівняння ефективності способу з існуючими аналогами





Практичне значення

Практичне значення одержаних результатів визначається тим, що запропонований спосіб побудови підсистеми інтелектуального редактора природномовних текстів та створення мовного сервера дозволили підвищити точність та швидкість надання мовної підтримки та розробити відповідні програмні засоби, придатні до використання в межах будь-якого мовного клієнта, що має підтримку всіх розроблених можливостей мовного сервера.

Наукова новизна

Вперше запропоновано модифікований спосіб інтелектуального асистування при написанні природномовних текстів, який відрізняється від існуючих способів автоматизованого оброблення природномовних текстів та інтелектуального асистування тим, що:

1. Вперше було запропоновано спосіб удосконалення протокола мовного сервера за рахунок розробки мовного протоколу та обрання протоколу більш низького рівня в якості транспортного, що забезпечує зменшення обсягу повідомлення.
2. Вперше було розроблено синтаксичні аналізатори з врахуванням афіксів для програмного забезпечення інтелектуального редагування природномовних текстів, що забезпечує більш точне обчислення релевантності лексеми шляхом врахування контексту, за рахунок чого було удосконаленно провайдери розумного доповнення, що забезпечує зменшення витраченого часу при написанні групи лексем.
3. Було створено виділений мовний сервер, що забезпечує зменшення навантаження на мовний клієнт, можливість оновлення мовних провайдерів без необхідності оновлення мовного клієнта та збільшення швидкості надання мовної підтримки.

Висновки

1. Розглянуто особливості оброблення природномовних текстів.
2. Запропоновано спосіб створення інтелектуальної підсистеми для вдосконалення асистування при написанні текстів.
3. Обрано технології для програмної реалізації запропонованого способа.
4. Розроблено програмне забезпечення, що реалізує запропонований спосіб.
5. Проаналізовано ефективність запропонованого способу та визначено переваги над існуючими аналогами.

Апробація роботи

Основні положення і результати роботи були представлені та обговорювались на науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг ПМК-2021» (Київ, 17-19 листопада 2021 р.)

Unicheck
89.1%

UNICHECK

Ім'я користувача:
Заболотня Тетяна Миколаївна

ID перевірки:
1009688411

Дата перевірки:
15.12.2021 18:38:14 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
15.12.2021 18:43:55 EET

ID користувача:
83364

Назва документа: Мединський_МД_Unicheck

Кількість сторінок: 73 **Кількість слів:** 14604 **Кількість символів:** 110298 **Розмір файлу:** 136.91 KB **ID файлу:** 1009687481

10.9%
Схожість

Дякую за увагу