МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача
кафедри
Ігор ШЕЛЕХОВ

(підпис)

18 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційна технологія автоматизації розробки мікросервісів та програмних бібліотек на основі штучного інтелекту OpenAI»
здобувача групи ІН.м - 23 Боженко Владислава Сергійовича

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

Боженко Владислав

(підпис)

Керівник,
доцент,
кандидат фізико-математичних
наук, доцент *Ігор Шеліхов*

(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ *Ігор ШЕЛЕХОВ*
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.М-23 Боженко Владислава Сергійовича

1. Тема роботи: «Інформаційна технологія автоматизації розробки мікросервісів та програмних бібліотек на основі штучного інтелекту OpenAI»

затверджую наказом по інституту від "6" грудня 2023 р. № 1412-VI
2. Термін здачі здобувачем кваліфікаційної роботи до 18 грудня 2023 року
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми. Огляд існуючих рішень. Огляд алгоритмів 2) Постановка задачі й формування завдань дослідження. 3) Створення архітектури системи 4) Практична реалізація. 5)Аналіз результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується *їх*

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	Аналіз проблеми. Огляд існуючих аналогів. Огляд алгоритмів	06.11-13.11	
2	Постановка завдання та вибір методів реалізації	14.11-19.11	
3	Створення архітектури системи	19.11-26.11	
4	Практична реалізація	27.11-11.12	
5	Оформлення пояснювальної записки до кваліфікаційної роботи	11.12-17.12	

Здобувач вищої освіти _____ *Керівник* _____

(підпис) *(підпис)*

АНОТАЦІЯ

Записка: 59 стр., 36 рис., 2 табл., 1 додаток, 19 використаних джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи актуальна, так як присвячена вирішенню практичної задачі з імплементації інформаційної технології автоматизації розробки мікросервісів та програмних бібліотек на основі штучного інтелекту OpenAI.

Об'єкт дослідження — автоматизація розробки інформаційних проектів

Мета роботи — створення та впровадження інформаційної технології, яка здатна автоматизувати розробку IT проектів, надаючи розробникам можливість делегувати рутинні задачі.

Методи дослідження — метод аналітичного огляду, метод порівняння та аналогій, метод моделювання, метод об’єктно-орієнтованого програмування для розробки продукту, метод емпіричного дослідження.

Результати — розроблено інформаційну технологію автоматизації розробки мікросервісів та програмних бібліотек. Комп’ютерна реалізація проекту виконана з використанням алгоритмічної мови Java та Kotlin, а в основі самої технології використано алгоритми NLP, а також існуючі провайдери AI функціоналу.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, АВТОМАТИЗАЦІЯ РОЗРОБКИ, NLP, ML, JAVA, KOTLIN

ЗМІСТ

ВСТУП

Сучасна індустрія програмного забезпечення переживає значну трансформацію від монолітних архітектур до більш гнучких і масштабованих мікросервісів. Хоча цей стрімкий розвиток приносить багато переваг у вигляді гнучкості, масштабованості та розподілу обов'язків, він також створив низку нових викликів, пов'язаних зі складністю розробки та підтримки таких систем.

Метою цієї роботи є вирішення цих проблем пов'язаних зі складністю розробки та підтримки таких систем шляхом впровадження інноваційних інструментів та підходів. Автоматизація процесу розробки мікросервісів та програмних бібліотек стала стратегічно важливою для компаній, які прагнуть не лише підвищити продуктивність, але й підтримувати високий рівень якості продукції[1].

Використання штучного інтелекту в цьому завданні покликане не тільки полегшити, але і поліпшити сам процес розробки, забезпечуючи автоматичне виявлення помилок, оптимізацію ресурсів і скорочення часу виробництва. З огляду на потужний потенціал технології, розробленої OpenAI, це дослідження є ключовим кроком на шляху до розробки стандарту. Воно спрямоване на створення інноваційних рішень, які виходять за рамки нестандартних підходів до розробки програмного забезпечення.

Завдання роботи - створити інструментарій, який не тільки спростить рутинну роботу розробників, але й забезпечує високий рівень якості у все більш складному програмному середовищі. Таким чином, робота має на меті зробити позитивний внесок у розвиток та вдосконалення індустрії інформаційних технологій шляхом поєднання архітектури мікросервісів та штучного інтелекту.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Дослідження предметної області

Штучний інтелект (ШІ) сьогодні є невід'ємною частиною всіх великих компаній електронної комерції. З розвитком інформаційної індустрії та широкими дослідженнями в галузі штучного інтелекту за останні два десятиліття компанії почали досліджувати способи автоматизації різних видів діяльності за допомогою найсучасніших алгоритмів машинного навчання та глибоких нейронних мереж. Багато IT-гігантів і стартапів вже зробили великий стрибок у цій галузі та мають спеціальні команди та ресурси для дослідження та розробки найсучасніших програм ШІ. Платформи онлайн-роздрібної торгівлі сьогодні значною мірою керуються алгоритмами та програмами на базі ШІ. Діяльність, починаючи від управління запасами та перевірки якості на складі до рекомендацій продукту та демографічних показників продажів на веб-сайті, використовує машинне навчання в різних масштабах [1].

Кількісні дані про придатність генеративних систем штучного інтелекту, для реальних бізнес-завдань демонструють ознаки покращення працездатності працівників різних галузей Їх продуктивність значно підвищується, при цьому найменш кваліфіковані користувачі отримали найбільші переваги. Деякі дослідження також показали поліпшення якості працюючих продуктів. Дослідження відбувалось під час одноразового використання інструментів штучного інтелекту - коли людина використовує штучний інтелект, завжди існує крива навчання, коли користувач вдосконалюється після повторного контакту з інтерфейсом користувача. [4]

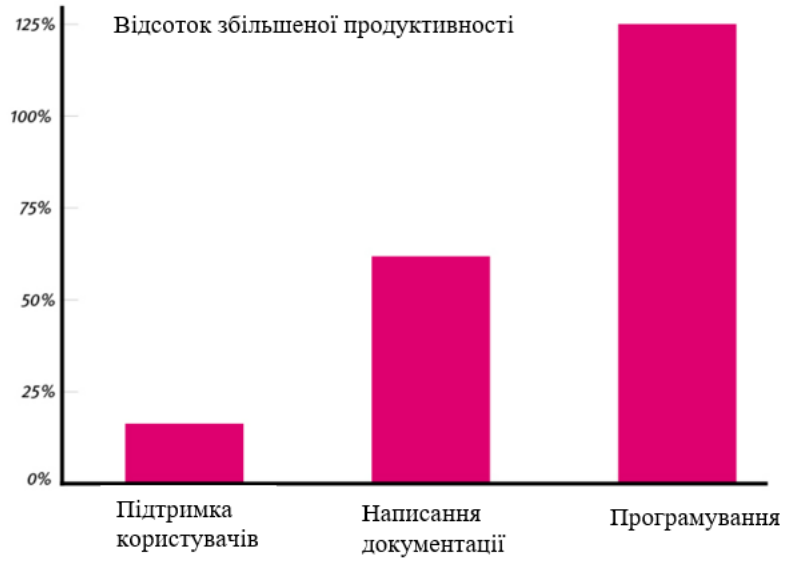


Рисунок 1.1.1 – Гістограма збільшення продуктивності в IT галузі після початку використання

Але підвищення продуктивності відбувається тільки тоді, коли співробітники виконують завдання, підтримувані ШІ. У деяких професіях, таких як UX-дизайн або інженер програмного забезпечення багато завдань можуть не підходити для підтримки штучного інтелекту, тому працівники в цих сферах отримують менше переваг. Створення технології автоматизації розробки мікросервісів та програмних бібліотек на основі штучного інтелекту OpenAI є доцільним, оскільки:

1. Подібна технологія автоматизації поліпшить продуктивність розробників на різних етапах проектів
2. Інструмент зменшить час витрачений на виконання рутинних задач, таких як: конфігурація проекту та його залежностей, конфігурація CI/CD, написання бізнес логіку та інше.
3. Дозволяє програмісту акцентувати увагу на виконанні ключових задач бізнесу, не змінюючи контекст роботи в рамках робочого дня.

1.2 Огляд існуючих рішень

GitHub Copilot - це інструмент розробки, створенний GitHub та OpenAI, який використовує мовну модель Codex від OpenAI для забезпечення автозавершення та підказок в інтегрованому середовищі розробки. Технологія інтегрується з різними редакторами коду, такими як Visual Studio Code та Atom, може генерувати код для різних завдань, таких як створення фрагментів коду, коментарів, тестів та інструкцій. OpenAI Codex заснований на мовній моделі, тому він розуміє контекстні завдання і може надавати як повні алгоритми, так і окремі рядки коду. Завдяки здатності адаптуватися до локальних і глобальних контекстів, GitHub Copilot забезпечує точне і контекстне автозавершення. Інструмент використовується для полегшення розробки, скорочення часу кодування і підвищення продуктивності розробників. [15]

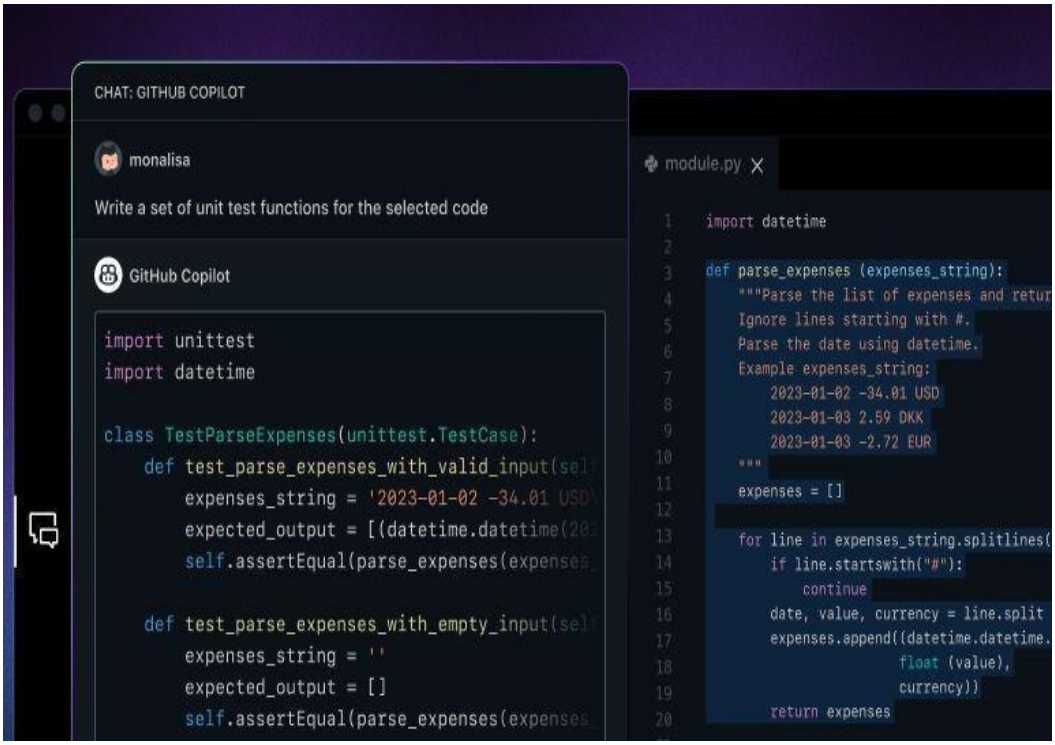


Рисунок 1.2.1 – Зовнішній вигляд GitHub Copilot

TabNine - це інтелектуальний інструмент для завершення коду, розроблений компанією TabNine. Заснований на технології машинного

навчання, він забезпечує швидке і точне автозавершення для полегшення створення коду.

Основними особливостями TabNine є використання моделей глибокого навчання для аналізу та генерації коду, підтримка різних мов програмування, адаптивне навчання для підвищення точності з часом, інтеграція з популярними редакторами коду, інтерактивне відображення підказок. TabNine є потужним інструментом для розробників, що допомагає підвищити продуктивність і зручність процесу написання коду.[16]

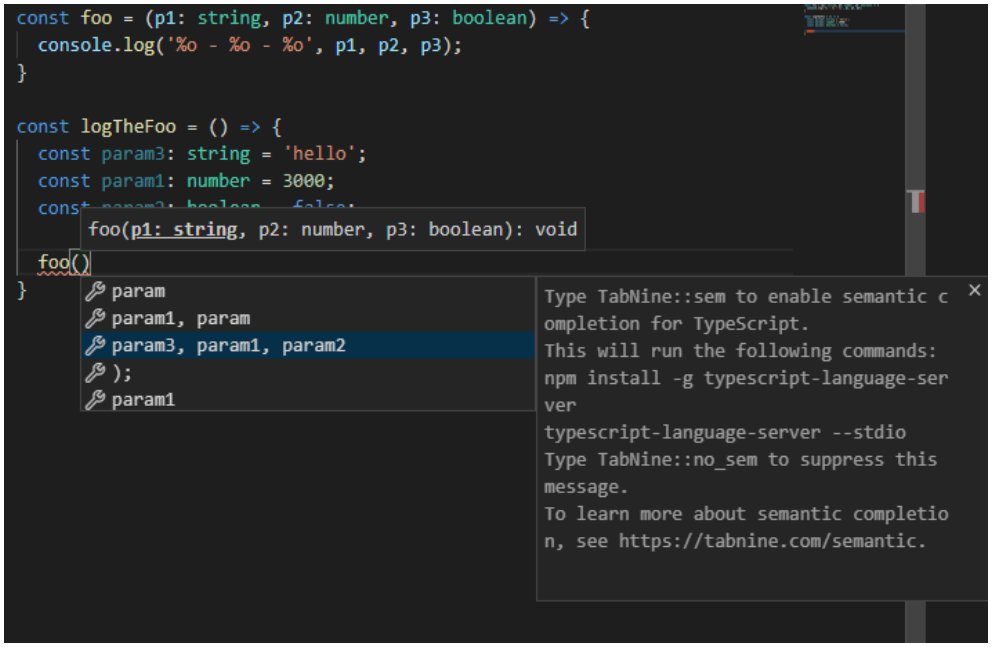


Рисунок 1.2.2 – Зовнішній вигляд TabNine

Під час оглядів інформаційних порталів було створена таблиця переваг та недоліків, що проаналізувати її та результаті отриманих даних зробити власний інформаційний портал.

Таблиця 1.2.1 – Результати порівняльного аналізу інформаційних порталів

Інструмент	**Переваги**	**Недоліки**
GitHub Copilot	1. GitHub Copilot значно прискорює процес написання коду, дозволяючи розробникам ефективно реалізовувати ідеї та функції. 2. Здатність інструменту розуміти статус розробки, враховувати раніше написаний код та забезпечувати контекстно-залежне автозавершення сприяє логічному та зручному робочому процесу. 3. Інтеграція з популярними редакторами коду, такими як Visual Studio Code, забезпечує розробникам зручну та безперешкодну роботу в їхньому улюбленому середовищі. 4. Пітримує генерацію різноманітних кодових конструкцій, включаючи функції, змінні та тестовий код. 5. Використання мовної моделі OpenAI Codex забезпечує високий рівень інтелектуальності та розуміння текстового опису завдань.	1. GitHub Copilot може генерувати код, який не завжди є точним або не відповідає найкращим практикам. Це може вимагати додаткових модифікацій від розробника. 2. Інструмент чутливий до контексту і введення. У деяких випадках автозавершення може залежати від контексту, що може вплинути на точність і різноманітність використання. 3. Велика кількість текстових даних, що використовуються для навчання, може викликати питання щодо конфіденційності та безпеки інформації. 4. Ліцензія на GitHub Copilot платна, що може бути обмеженням для окремих користувачів.

Продовження Табл. 1.2.1

TabNine	1. Забезпечує швидке і точне автозавершення, що полегшує написання коду і підвищує продуктивність розробників. 2. Підтримка різних мов програмування та їх комбінацій в одному проекті робить його універсальним інструментом для розробників у різних галузях. 3. Точність автозавершення погіршується з часом завдяки його здатності самоорганізовуватися відповідно до стилю коду розробника. 4. Розширені плагіни для популярних редакторів, таких як Visual Studio Code, полегшують використання TabNine у робочому середовищі.	– Деякі функції можуть вимагати підключення до Інтернету, що може бути незручним, якщо умови роботи обмежені. – Може генеруватися неповний або не оптимізований код, що може вимагати подальшої модифікації розробником. – Безкоштовна версія має обмежену функціональність і продуктивність в деяких областях, що може вплинути на повноту функціональності.

Отримавши результат після аналізу наведений вище прикладів, було вирішено, що розробити власну технологіяю автоматизації розробки мікросервісів та програмних бібліотек є доцільним.

1.3 Постановка задачі

Метою роботи є розроблення інструменту, що використовує штучний інтелект для автоматизації розробки мікросервісів та програмних бібліотек, створення зручного та ефективного інструментарію, який спростить та

прискорить процес імплементації програмного забезпечення. Таким чином, основними задачами є:

- Провести детальний аналіз можливостей та обмежень OpenAI для визначення його використання в автоматизації розробки.
- Розробити архітектуру інструменту, враховуючи потреби розробників у створенні мікросервісів та програмних бібліотек.
- Забезпечити можливість інтеграції інструменту з популярними редакторами коду
- Розробити функціонал для автоматичної генерації коду на основі текстових описів завдань.
- Провести широкий спектр тестів для перевірки якості та надійності інструменту, а також здійснити валідацію результатів генерації.
- Надати засоби підтримки для користувачів та підготувати докладну документацію з використання та налаштування інструменту.

Розроблена технологія має підтримувати наступний функціонал:

1. Система аналізує і розпізнає ключові компоненти потрібних технологій і додаткові бізнес-вимоги.
2. Система генерує структурований запит до OpenAI, що містить інформацію про технології, бізнес-логіку та вимоги до генерації коду. Запит має містити важливі деталі щодо необхідних класів, методів і логіки, які мають бути включені в код, що генерується.
3. Система надсилає сформований запит до провайдеру OpenAI, отримуючи у відповідь згенерований код, що містити класи, методи, коментарі та інші деталі.
4. Код, отриманий від AI, зберігається локально у вигляді програмних компонентів, таких як класи/модулі. Система автоматично оптимізує код.

5. Система інформує користувача про успішно згенерований та оптимізований код. Користувач може переглянути згенерований код, внести зміни та інтегрувати його в проект.

2 ВИБІР ПРОГРАМНИХ ЗАСОБІВ. ПРОЕКТУВАННЯ ТЕХНОЛОГІЇ

2.1 Вибір технології для обробки природної мови

Обробка природної мови(NLP) - це галузь комп'ютерних наук і, зокрема, штучного інтелекту або ШІ, що поєднує обчислювальну лінгвістику (моделювання людської мови на основі правил) зі статистикою, машинним навчанням і моделями глибокого навчання. Поєднуючи ці методи, комп'ютери можуть обробляти людську мову у вигляді тексту і мовних даних і розуміти її повне значення. Обробка природної мови лежить в основі комп'ютерних програм, які перекладають текст з однієї мови на іншу, реагують на голосові команди і швидко узагальнюють великі обсяги тексту, навіть у голосових системах GPS у режимі реального часу, цифрових помічниках і програмному забезпеченні для перетворення голосу в текст та інші форми сервісів для зручності споживачів. Однак NLP також відіграє все більш важливу роль у корпоративних рішеннях, які допомагають впорядкувати бізнес, підвищити продуктивність співробітників і спростити критичні бізнес-процеси.[5]

Людська мова сповнена неточностей, що робить дуже складним написання програмного забезпечення, яке точно визначає передбачуване значення текстових або усних даних. Омоніми, омофони, іронія, ідіоми, метафори, винятки з граматики та вживання, а також варіації у структурі речень – ускладнюють процес розуміння контексту тексту.

Для кращого індексування та пошуку даних з великих обсягів тексту необхідно відфільтрувати непотрібні слова з даних і індексувати тільки логічні слова, щоб отримати кращу продуктивність пошуку. Обробка природної мови часто включає в себе зв'язки між словами в одному реченні та синтаксис речення. Тому, якщо заздалегідь створити індексний файл, можна швидко і легко знайти потрібне слово, і підвищити здатність працювати з великомасштабними запитами. Два основні підходи, що використовуються,

поєднують статистичну інформацію про розподіл слів із загальними і специфічними для тексту термінами. Алгоритми, які дотримуються цих підходів, включають систему, незалежну від частоти термінів (TF-IDF), яка базується виключно на частоті термінів, і метод генерації резюме, який використовує комбінацію частоти термінів та інших ознак для обчислення ваги речень, метод Cue, метод Title, "Trainable Document Summary", який використовує комбінацію інших ознак, таких як метод розташування, обмеження довжини речення, фіксовані фрази, ознаки абзаців, слова-теми, слова з великої літери та інші ознаки, а також частоти термінів. [7]

Рисунок 2.1.1 – Приклад аналізу вхідного тексту

Узагальнення тексту за допомогою нейронних мереж стало важливим досягненням у галузі обробки природної мови. Нейронна мережа навчається на корпусі статей і модифікується за допомогою об'єднання ознак для створення резюме з реченнями з найвищим рейтингом у статті. У процесі навчання нейронна мережа вивчає типи речень, які повинні бути включені в резюме. Під час злиття ознак нейромережа зменшує активність прихованих шарів і згортає їх в окремі значення з частотою. Потім вона підсумовує важливі ознаки, які повинні бути присутніми в реченнях, що складають частину резюме. Нарешті, модифікована нейронна мережа відбирає речення, які стануть частиною резюме, шляхом їх ранжування.

NLP дозволяє комп’ютерам обробляти тексти та слова подібно до людей. Він поєднує комп’ютерну лінгвістику зі статистикою, глибоким навчанням і машинним навчанням. Люди щодня взаємодіють один з одним в Інтернеті за допомогою різних медіа. При цьому вони обмінюються різними типами даних, такими як текст, мова, зображення тощо.[10] Ці дані є важливими для розуміння людської поведінки та звичок. Тому їх використовують для навчання комп’ютерів імітації людського інтелекту. NLP використовує дані, щоб навчити машини імітувати людську мовну поведінку

Apache OpenNLP — це набір інструментів, який використовує машинне навчання для обробки тексту природною мовою. Він забезпечує підтримку поширених завдань NLP, таких як токенізація, сегментація, тегування мови тощо. Основною метою Apache OpenNLP є забезпечення підтримки завдань NLP і надання великої кількості готових моделей для різноманітних мов. Крім того, він забезпечує інтерфейс командного рядка (CLI) для легких експериментів і навчання:

CoreNLP — це набір програм, написаних на Java Стенфордською групою NLP, які можуть виконувати різноманітні завдання NLP, як-от токенізація, тегування частин мови, лемматизация тощо. Його можна використовувати через командний рядок, у коді Java або за допомогою викликів на сервер.

В рамках даної роботи нами буде використано обидва інструменти, оскільки вони чудово доповнює один одного.

2.2 Вибір системи штучного інтелекту

У цьому розділі ми ретельно розглянемо ряд існуючих систем штучного інтелекту, спрямованих на визначення оптимального вибору для конкретних вимог та завдань. Наш аналіз зосереджений на ключових гравців у цій галузі, які визначають стандарти та напрями розвитку.

OpenAI - сучасна дослідницька організація, яка визначає стандарти для штучного інтелекту та машинного навчання. Одне з її найважливіших

досягнень - генеративний трансформатор GPT, що є потужною мовною моделлю, доступною на сьогоднішній день. GPT базується на трансформаторній архітектурі, яка дозволяє ефективно обробляти та генерувати текст. OpenAI має відкритий доступ до GPT API. OpenAI API - це потужний інструмент, який дозволяє розробникам інтегрувати лінгвістичний інтелект у свої продукти, відкриває надаючи широкий спектр можливостей, від автоматичної генерації контенту до вирішення завдань обробки природної мови (NLP)[12].

Google AI відіграє важливу роль у сфері штучного інтелекту, завдяки платформі TensorFlow. TensorFlow - це платформа машинного навчання з відкритим вихідним кодом, розроблена в Google. Вона надає інструменти для створення та навчання моделей, включаючи нейронні мережі. TensorFlow базується на графових обчисленнях і є ефективним для паралельного виконання. Він також має широкий спектр інструментів і бібліотек для обробки даних і розробки моделей. Це дає можливість використовувати інтелект Google у найрізноманітніших сферах - від хмарних сервісів до пошуку інформації. З огляду на динаміку розвитку, очікується, що Google AI і надалі сприятиме розширенню функціональності TensorFlow та впровадженню нових рішень у сфері штучного інтелекту. TensorFlow від Google AI є синергетичною платформою для машинного та глибокого навчання. Його висока продуктивність та гнучкість роблять його важливим інструментом для створення та тренування моделей. Наш аналіз охопить взаємодію TensorFlow з іншими продуктами Google, зокрема, визначаючи можливості інтеграції [9].

Microsoft AI - це комплексний набір інструментів і служб, спрямованих на підтримку розробки та впровадження рішень штучного інтелекту. Azure Machine Learning - це хмарна платформа Microsoft для створення, навчання та розгортання моделей машинного навчання. Вона надає інфраструктуру та інструменти для підтримки всього життєвого циклу моделі. Розгортання та

моніторинг є простими, що полегшує розгортання та керування моделями в режимі реального часу.

Cognitive Services - це пакет готових служб, які надають можливості штучного інтелекту розпізнавання мови та аналіз тексту. Ці сервіси не вимагають глибоких знань про машинне навчання. Когнітивні сервіси дозволяють розробникам легко додавати інтелект у додатки, плагіни та веб-сайти. API для цих сервісів забезпечують швидку інтеграцію з різними платформами. З огляду на активний розвиток Azure Machine Learning і постійне вдосконалення Cognitive Services, в майбутньому можна очікувати подальшого розвитку і розширення сфери застосування можливостей Microsoft AI [9].

Таблиця 2.2.1 – Аналіз існуючих систем штучного інтелекту

Інструмент	***Переваги***	***Недоліки***
Open AI	1. Має здатність до розуміння та генерації природних текстів, що робить його потужним інструментом для завдань обробки мови і перетворення її в програмний код. 2. Відкритий доступ до API GPT дозволяє розробникам використовувати мовні можливості у своїх проектах. 3. Використовується в генерації контенту, розв'язанні завдань NLP та розробці чат-ботів.	1. Вільний доступ до них обмежений, а висока вартість широкого використання може бути економічним бар'єром. 2. GPT можуть створювати контент, який відповідає на запити, але якість відповіді може залежати від якості вхідних даних.

Продовження Табл. 2.2.1

Google AI	1. TensorFlow - це потужна і гнучка платформа машинного навчання, яка пропонує ефективність і багату функціональність. 2. інтеграція з іншими продуктами Google дозволяє використовувати інтелект у різних сферах. 3. використовується в різних галузях, в тому числі в медицині, автомобілебудуванні та комп'ютерному зорі.	1. Деякі функції можуть вимагати підключення до Інтернету, що може бути незручним, якщо умови роботи обмежені. 2. Може генеруватися неповний або не оптимізований код, що може вимагати подальшої модифікації розробником. 3. Безкоштовна версія має обмежену функціональність і продуктивність в деяких областях, що може вплинути на повноту функціональності.
Microsoft AI	1. Надає широкий спектр інструментів і готових служб штучного інтелекту. 2. Інтеграція з іншими службами Microsoft у хмарному середовищі забезпечує безперебійну роботу системи.	1. Використання Azure передбачає залежність від хмарних сервісів, що може бути обмеженням для розробників. 2. Використання сервісів Microsoft AI, зокрема великих моделей для комплексного аналізу даних, може призводити до значного споживання ресурсів

В контексті завдань роботи і вимог до майбутнього проекту використання OpenAI, зокрема мовної моделі GPT виглядає найбільш доцільним, бо має важливі переваги, що роблять її потужним інструментом для створення програмного коду.

2.3 Вибір алгоритму роботи технології

Використання методів машинного навчання в контексті обробки природної мови представляє собою велике поле досліджень, яке не може бути вичерпно охоплене в рамках однієї магістерської роботи. Слід зазначити, що вибір конкретних пакетів та моделей є суб’єктивним, спрямованим на покращення аналізу зворотного зв'язку. Однак важливо враховувати, що існують інші ефективні інструменти та підходи. Цифрова трансформація у сфері безпеки набирає обертів, оскільки все більше процесів переходять в онлайн. Збираючи значно більше даних, ніж раніше, ми стикаємося з проблемою ефективного їх використання[1]. Багато з цих даних є у вигляді сирих текстів, які ми не завжди можемо встигти обробити та проаналізувати, що може вести до упущених загроз або невизначених причин подій. Однак останні досягнення у сфері Обробки Природної Мови (Natural Language Processing - NLP) вносять рішення в ці проблеми. Однак останні досягнення у сфері Обробки природної мови (Natural Language Processing - NLP) вносять рішення в ці проблеми:

1. Обробка природної мови полегшує збір даних, конвертуючи потоки інформації у машинно читабельний текст. До прикладів NLP-застосувань для збору даних входять оптичне розпізнавання символів, конвертація мови у текст та переклад мов. Хоча ці застосування NLP є зрілі, їх впровадження у рішення для забезпечення безпеки відбувається повільно.

2. Після збору даних алгоритми класифікації NLP допомагають визначити пріоритети для подальшої обробки. Наприклад, аналіз настрою

може розрізняти емоційні стани в тексті, такі як гнів, радість чи сум. Сентимент-аналіз використовується в продуктах моніторингу соціальних мереж для фільтрації шумової інформації та виявлення потенційних загроз.

3. Допомагаючи збільшити швидкість обробки, алгоритми визначення імен (NER) автоматизують вилучення важливих даних з наративних звітів і розвідувальних документів. Це включає визначення осіб, місць та організацій, що значно економить час та підвищує якість даних.

4. Одним із найвідоміших проектів у сфері NLP є GPT-3 від Open AI. Це загальний штучний інтелект, який приймає введені дані та генерує наративний текст. Текстова генерація – це швидкорозвиваюча галузь NLP, інновації в якій відбуваються практично постійно. Хоча наразі продуктовані програмні засоби для безпеки, які використовують генерацію тексту, ще відсутні, ймовірно, це швидко зміниться, з першим застосуванням алгоритмів, як у випадку наведеного вище, для узагальнення даних про інциденти та загрози у чіткий конспект[8].

Аналіз настрою є дуже популярним застосуванням сили машинного навчання. Аналізуючи вміст кожного тексту, ми можемо оцінити, наскільки позитивний чи негативний вага речення або весь текст. Це має величезну цінність, оскільки нам потрібно валідувати негативний текст. Розглянемо приклад такого аналізу: нам знадобляться наступні рядки коду для виконання базового аналізу настрою

```
// створюємо екземпляр SentimentAnalyzer:
SentimentAnalyzer sentimentAnalyzer = new SentimentAnalyzer(
    new SentimentModel(new File("path/to/sentiment-model")));

// проводимо аналіз:
String text = "проект виглядає дивовижно, чудова робота";
double[] outcomes = sentimentAnalyzer.predictProbabilities(text);

// результат:
double positiveProbability = outcomes[0];
double negativeProbability = outcomes[1];
```

Рисунок 2.3.1 — Приклад сентематичного аналізу

Вихідним значенням є кортеж, де перше значення - це "полярність" (наскільки позитивне речення на шкалі від -1 до 1). Друге значення - це суб'єктивність, яка показує, наскільки алгоритм впевнений у оцінці свого першого значення; на цей раз шкала починається з 0 і закінчується на 1. Давайте розглянемо кілька додаткових прикладів:

```
public static void main(String[] args) {
    try {
        // Навчальні дані для тренування моделі (потрібно надати свої дані)
        InputStream trainingData = new FileInputStream("навчальних-данні.txt");

        // Тренування моделі настрою
        SentimentModel sentimentModel = trainSentimentModel(trainingData);

        // Ініціалізація SentimentME з тренованою моделлю
        SentimentME sentimentAnalyzer = new SentimentME(sentimentModel);

        // Приклади речень пов'язаних з айті проектами
        String projectDescription1 = "Розробка веб-додатку для управління завданнями в агільному проекті.";
        String projectDescription2 = "Оптимізація бази даних для підвищення продуктивності системи.";

        // Аналіз настрою
        double[] result1 = sentimentAnalyzer.calculateSentenceScores(projectDescription1);
        double[] result2 = sentimentAnalyzer.calculateSentenceScores(projectDescription2);

        // Виведення результатів
        System.out.println("Результат для projectDescription1: (" + result1[0] + ", " + result1[1] + ")");
        System.out.println("Результат для projectDescription2: (" + result2[0] + ", " + result2[1] + ")");
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private static SentimentModel trainSentimentModel(InputStream trainingData) throws IOException {
    // Тренування моделі настрою з використанням OpenNLP
    SentimentSample[] samples = SentimentSample.readSamples(trainingData);
    SentimentModel model = new SentimentTrainer().train(samples);
    return model;
}
```

Рисунок 2.3.2 — Приклад вхідних даних для сентематичного аналізу

У цьому прикладі виводу перше число в кожному кортежі вказує на полярніст" речення (наскільки воно позитивне чи негативне), а друге число - на "суб'єктивність" (наскільки впевнений алгоритм у своєму висновку). Обидва числа знаходяться в діапазоні від 0 до 1. У цьому випадку, більше значення полярності вказує на більш позитивне речення, а менше значення суб'єктивності вказує на більш впевнений алгоритм.

```
Результат для projectDescription1: (0.365, 0.635)
Результат для projectDescription2: (0.598, 0.402)
```

Рисунок 2.3.3 — Приклад вихідних даних для сентематичного аналізу

Вилучення ключових слів з даного рядка - це ще одна потужна техніка, яка може поліпшити аналіз тексту. Пакет OpenNLP надає можливості для токенізації тексту та аналізу частин мови, що робить його ефективним для визначення важливих слів у тексті. Після токенізації тексту можна використовувати інструменти OpenNLP для визначення ключових слів, враховуючи частини мови та контекст тексту.

Токенізація є першим етапом у будь-якому NLP-процесі. Її важливий вплив охоплює усі наступні етапи аналізу. Токенізатор розбиває неструктуровані дані та текст природної мови на частини інформації, які можна розглядати як відокремлені елементи. Випадки токенів у документі можуть бути використані безпосередньо як вектор, що представляє цей документ. Цей процес перетворює неструктурований рядок (текстовий документ) у числову структуру даних, що підходить для машинного навчання. Токени також можуть бути використані безпосередньо комп'ютером для виклику корисних дій та відповідей[5]. Чи вони можуть служити в машинному навчанні як ознаки, які викликають більш складні рішення чи поведінку.

Приведений нижче код на Java демонструє використання OpenNLP для токенізації тексту:

```java
public class OpenNLPTokenization {

    public static void main(String[] args) {
        try {
            // Завантаження моделі токенізатора OpenNLP
            InputStream modelIn = new FileInputStream("шлях/до/моделі/tokenizer-model.bin");
            TokenizerModel tokenizerModel = new TokenizerModel(modelIn);
            Tokenizer tokenizer = new SimpleTokenizer(tokenizerModel);

            // Приклад тексту для токенізації
            String textToTokenize = "Токенізація - це перший етап будь-якого NLP-процесу.";

            // Токенізація тексту
            String[] tokens = tokenizer.tokenize(textToTokenize);

            // Виведення результатів
            System.out.println("Токени: " + Arrays.toString(tokens));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Рисунок 2.3.4 — Приклад токенізації вхідних даних

Вивід у консолі для вказаного коду:

```
Токени: [Токенізація, -, це, перший, етап, будь-якого, NLP-процесу, .]
```

Рисунок 2.3.5 — Приклад вихідних даних в результаті токенізації

Приведений нижче код на Java ілюструє використання OpenNLP для визначення ключових слів. Цей код тонізує текст та виводить токени, а також визначає та виводить топ ключових слів. Для використання потрібно вказати шлях до моделі токенізатора OpenNLP та розширити список стоп-слів відповідно до контексту:

```java
public class OpenNLPKeywordExtraction {

    public static void main(String[] args) {
        try {
            InputStream modelIn = new FileInputStream("шлях/до/моделі/tokenizer-model.bin");
            TokenizerModel tokenizerModel = new TokenizerModel(modelIn);
            Tokenizer tokenizer = new SimpleTokenizer(tokenizerModel);

            String textToAnalyze = "Важливий фрагмент тексту для аналізу настрою та визначення ключових слів.";

            String[] tokens = tokenizer.tokenize(textToAnalyze);

            System.out.println("Токени: " + Arrays.toString(tokens));

            List<String> topKeywords = getTopKeywords(tokens);
            System.out.println("Топ ключових слів: " + topKeywords);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    private static List<String> getTopKeywords(String[] tokens) {
        List<String> topKeywords = Arrays.stream(tokens)
                .filter(token -> !isStopWord(token))
                .collect(Collectors.toList());

        System.out.println("Результат: " + topKeywords);
        return topKeywords;
    }
    private static boolean isStopWord(String word) {
        List<String> stopWords = Arrays.asList("це", "і", "та", "в", "з", "на", "від", "що");
        return stopWords.contains(word);
    }
}
```

Рисунок 2.3.6 — Приклад вихідних даних в результаті токенізації

Наведений Java-код використовує бібліотеку OpenNLP для визначення ключових слів у тексті, що є важливим етапом в обробці природної мови (NLP). Код розпочинається завантаженням моделі токенізатора OpenNLP. Потім задається текст для аналізу, який піддається токенізації – процесу розбиття тексту на окремі слова. Після токенізації визначаються та виводяться токени. Код також містить функцію для фільтрації та вибору топових ключових слів, прибираючи стоп-слова (наприклад, "це", "та"). Ключові слова виводяться для подальшого аналізу.

Цей код буде використано для вивчення та розуміння процесу визначення ключових слів в тексті за. За виправленням шляху до моделі та розширенням списку стоп-слів його можна використовувати для аналізу тексту та виділення важливої інформації.

Тематичне моделювання (Topic Modeling) є важливим інструментом у сфері обробки природної мови (NLP), і його застосування в магістерській роботі може значно покращити розуміння контенту тексту.

Тематичне моделювання дозволяє автоматично визначати теми, які присутні в текстових даних. Замість того, щоб ручно визначати ключові слова або категорії, модель аналізує вміст і самостійно визначає теми, що робить його ефективним для обробки великих обсягів інформації. Це особливо корисно для аналізу великих текстових корпусів, таких як наукові статті, новинні статті або соціальні мережі.

Процес тематичного моделювання включає в себе кілька кроків. По-перше, текст розбивається на токени (слова), які потім лематизуються для видалення форм слова. Наступним етапом є створення словника та матриці термінів документів, яка відображає, які слова зустрічаються у кожному документі. Далі використовується тематична модель, така як Latent Dirichlet Allocation (LDA), для визначення тем, які можуть бути присутні у тексті.

Застосування тематичного моделювання в магістерській роботі може мати різноманітні застосування. Наприклад, визначення головних тем у колекції документів або аналіз динаміки тематик з часом. Також можливе використання для кластеризації документів за подібністю тем[5]. Правильне налаштування параметрів тематичної моделі дозволяє отримати точніші та значущі результати. Важливо також враховувати специфічні особливості дослідження та контексту застосування. Тематичне моделювання є потужним інструментом для розкриття структури текстових даних і використання його може принести значний внесок у вивчення та розуміння текстової інформації в роботі.

```java
public class TopicModeling {
    public static void main(String[] args) {
        try {
            InputStream tokenizerModelIn = new FileInputStream("en-token.bin");
            TokenizerModel tokenizerModel = new TokenizerModel(tokenizerModelIn);
            Tokenizer tokenizer = new SimpleTokenizer();
            InputStream lemmatizerModelIn = new FileInputStream("en-lemmatizer.dict");
            LemmatizerModel lemmatizerModel = new LemmatizerModel(lemmatizerModelIn);
            LemmatizerME lemmatizer = new LemmatizerME(lemmatizerModel);
            String textToAnalyze = "Важливий фрагмент тексту для аналізу настрою та визначення ключових слів.";
            String[] lemmas = lemmatizer.lemmatize(tokens, new String[0]);
            System.out.println("Токени: " + Arrays.toString(tokens));
            System.out.println("Леми: " + Arrays.toString(lemmas));
            List<String> topKeywords = getTopKeywords(tokens);
            System.out.println("Топ ключових слів: " + topKeywords);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    private static List<String> getTopKeywords(String[] tokens) {
        List<String> topKeywords = Arrays.stream(tokens)
                .filter(token -> !isStopWord(token))
                .collect(Collectors.toList());
        System.out.println("Результат: " + topKeywords);
        return topKeywords;
    }
    private static boolean isStopWord(String word) {
        List<String> stopWords = Arrays.asList("це", "і", "та", "в", "з", "на", "від", "що");
        return stopWords.contains(word);
    }
}
```

Рисунок 2.3.7 — Приклад коду для Topping modeling

Наведений код реалізує приклад тематичного моделювання за допомогою бібліотеки OpenNLP у середовищі Java. Спочатку відбувається завантаження моделей токенізатора та лематизатора, які будуть використовуватися для розділення тексту на токени та визначення їх базових форм. Після цього введений текст для аналізу тематики піддається токенізації та лематизації. Отримані токени та їх леми виводяться на консоль. Далі використовується приклад фільтрації, де визначаються топові ключові слова, виключаючи стоп-слова. При цьому реалізовано можливість розширення списку стоп-слів відповідно до конкретного контексту або завдання. Цей код може бути використаний як основа для розвитку більш складних аналітичних інструментів, наприклад, для визначення головних тем великих обсягів текстової інформації. Отримані результати можуть бути використані для подальшого вивчення структури тексту та виявлення суттєвих тем. Цей код структурований та легко розширюється, що робить його корисним інструментом для використання в проектах, пов'язаних з обробкою текстових даних у мові програмування Java.

Конкорданс - це інструмент аналізу тексту, який надає можливість дослідження контексту вживання певного слова в текстовому корпусі. У фреймворку OpenNLP конкорданс дозволяє швидко отримати контекст вживання конкретного слова, виводячи фрагменти тексту, які оточують це слово. Для використання конкордансу в OpenNLP спочатку необхідно завантажити модель токенізатора та інші необхідні ресурси. Далі введені текстові дані розбиваються на токени (слова або фрази) за допомогою токенізатора. Потім обирається слово, вживання якого ми хочемо дослідити. Конкорданс генерує вихідний текст, включаючи контекст вказаного слова. Цей контекст може включати слова, які стоять перед і після обраного слова в кожному входженні. Такий аналіз дозволяє вивчати, як дане слово використовується в різних контекстах та розуміти його значення в залежності від сусідніх слів. Використання конкордансу важливо при лінгвістичних та літературних дослідженнях, а також при аналізі текстових даних у сферах, де ключове значення має контекстуальний аспект. Цей інструмент стає ефективним засобом для вивчення вживання слів у текстах, аналізу стилістики та виявлення специфічних лінгвістичних відмінностей в текстових корпусах. Завдяки конкордансу в OpenNLP вивчення контекстуального вживання слів стає швидким та ефективним процесом для дослідників та аналітиків[8].

Цей Java-код використовує бібліотеку OpenNLP для створення конкордансу, що дозволяє аналізувати контекст вживання певного слова в тексті. Починаючи з завантаження моделі токенізатора та вказівки шляху до неї, код тонізує введений текст. Після цього обирається слово для дослідження, у цьому випадку "OpenNLP". Код створює конкорданс для обраного слова, аналізуючи контекст вказаного слова у тексті. Він виводить фрагменти тексту, що оточують це слово, у консоль. Параметри, які можна налаштовувати, включають розмір контексту та шлях до моделі токенізатора OpenNLP, що дає гнучкість при використанні. Цей код може бути корисним для лінгвістичних досліджень та аналізу текстових даних, де важливий

контекст вживання слів. Він надає можливість швидко отримувати інформацію про те, як дане слово використовується у різних частинах тексту, допомагаючи дослідникам отримувати важливі висновки зі збірних текстових даних.

```java
public class ConcordanceExample {

    public static void main(String[] args) {
        try {
            InputStream tokenizerModelIn = new FileInputStream("шлях/до/моделі/en-token.bin");
            TokenizerModel tokenizerModel = new TokenizerModel(tokenizerModelIn);
            Tokenizer tokenizer = new SimpleTokenizer();
            String textToAnalyze = "Це приклад тексту для аналізу конкордансу за допомогою OpenNLP.";
            String[] tokens = tokenizer.tokenize(textToAnalyze);
            String targetWord = "OpenNLP";
            List<String> concordance = createConcordance(tokens, targetWord);
            System.out.println("Конкорданс для слова '" + targetWord + "':");
            for (String context : concordance) {
                System.out.println(context);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    private static List<String> createConcordance(String[] tokens, String targetWord) {
        List<String> concordance = new ArrayList<>();

        for (int i = 0; i < tokens.length; i++) {
            if (tokens[i].equalsIgnoreCase(targetWord)) {
                int contextSize = 5; // Задаємо розмір контексту (змініть за потребою)
                int startIndex = Math.max(0, i - contextSize);
                int endIndex = Math.min(tokens.length, i + contextSize + 1);
                StringBuilder contextBuilder = new StringBuilder();
                for (int j = startIndex; j < endIndex; j++) {
                    contextBuilder.append(tokens[j]).append(" ");
                }
                concordance.add(contextBuilder.toString().trim());
            }
        }
        return concordance;
    }
```

Рисунок 2.3.8 — Приклад створення конкордансу

У цьому випадку прикладі "OpenNLP" було знайдено в тексті, і виведено фрагмент тексту, який оточує це слово. Контекст включає слова перед та після обраного слова в межах зазначеного контексту

```
Токени: [Це, приклад, тексту, для, аналізу, конкордансу, за, допомогою, Oper
Леми: [Це, приклад, текст, для, аналіз, конкорданс, за, допомогою, OpenNLP,
Конкорданс для слова 'OpenNLP':
Тексту для аналізу конкордансу за допомогою
```

Рисунок 2.3.9 — Приклад виведення конкордансу в консоль

В рамках магістерської роботи з теми використання методів аналізу природної мови (NLP) детально вивчається для покращення якості та продуктивності розробленого програмного забезпечення. Розглянемо різні аспекти використання NLP:

1. Використання NLP для пошуку ключових слів дозволяє виділяти та виділяти найважливіші терміни та поняття у тексті. Це корисно для автоматизованого створення та підтримки документації, визначення тем та контексту текстового матеріалу.
2. Моделювання тем за допомогою NLP дозволяє визначити основні теми та концепції, які присутні в текстових даних. Це може використовуватися для класифікації документів за темами, аналізу відгуків користувачів та виявлення основних проблем чи інтенцій у текстах.
3. Використання синтаксичного аналізу дозволяє розуміти взаємозв'язки та структуру мовлення у тексті. Це важливо для автоматизованого виявлення та виправлення синтаксичних помилок у програмному коді, а також для покращення якості згенерованих текстових відповідей.
4. Конкорданс допомагає аналізувати використання конкретного слова в тексті, дозволяючи переглядати контекст, в якому воно зустрічається. Це корисно для розуміння контексту використання термінів у програмному коді та текстах, а також для виявлення закономірностей у використанні мови [10].

2.4 Проектування архітектури інструменту

Діаграма компонентів, відома як також діаграма компонентів UML, подає структуру та взаємозв'язки фізичних компонентів у системі. Це графічне представлення взаємодій та залежностей між різними компонентами програмного забезпечення. Вона служить для візуального відображення структури системи та її ключових елементів. Кожен компонент відображає

окрему частину програми, а стрілки показують взаємодії між ними. Цей інструмент допомагає аналізу та проектуванню архітектури програм та сприяє порозумінню їхньої структури. Діаграма компонентів є важливим етапом у розробці програмних систем, дозволяючи розробникам та іншим учасникам отримати чіткий огляд системи. Ми створили діаграми компонентів для аналізу деталей реалізації моделі та перевірки, чи охоплені всі необхідні аспекти системи в ході запланованої розробки. Наведена діаграма компонентів (рис. 2.4.1) включає всі наявні мікросервіси та використовувані технології для забезпечення роботи системи.

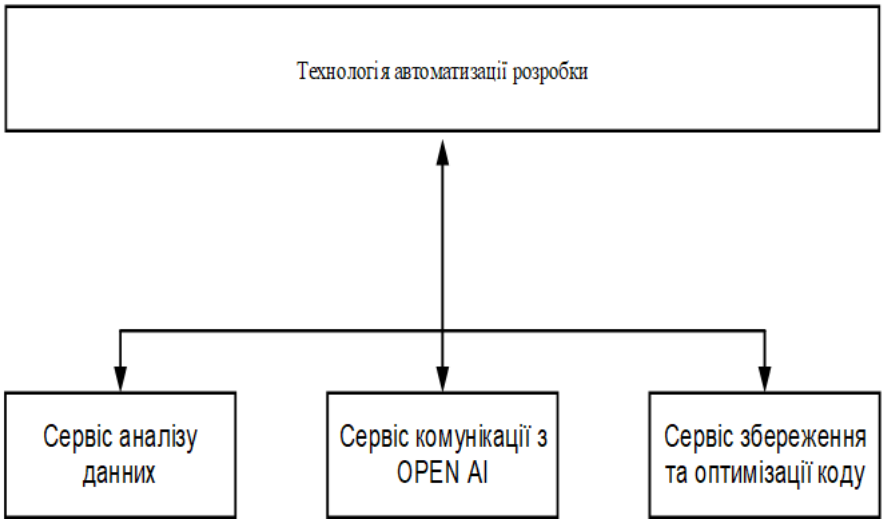


Рисунок 2.4.1 – Абстрактна діаграма компонентів системи

Діаграма послідовності відображає взаємодію об'єктів, яка впорядкована в часовій послідовності (див. Рис. 2.4.2), що сприяє уточненню, які об'єкти та взаємодії необхідні для виконання конкретної функціональності. Ці діаграми є потужним інструментом, особливо на етапі початкового проекту, оскільки крок за кроком вони демонструють, як систематизувати взаємодію між компонентами системи. Ми розробили діаграму послідовності для докладного аналізу основної частини функціоналу системи.

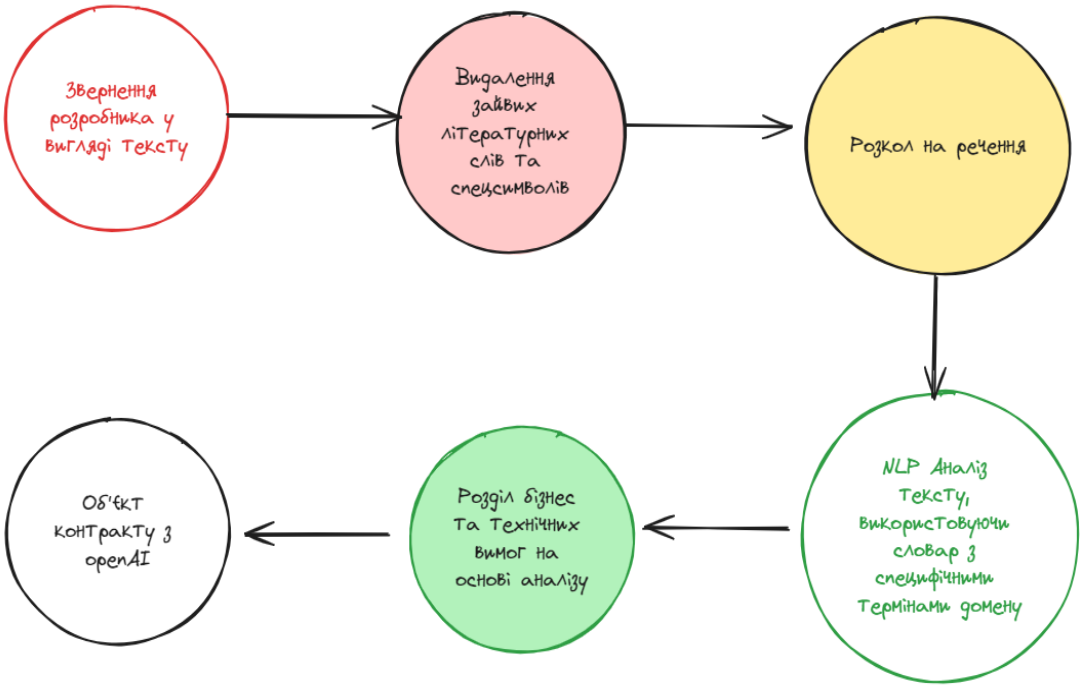


Рисунок 2.4.2 – Абстрактна діаграма компонентів системи

3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

3.1 Реалізація алгоритму роботи інформаційної системи

1. Ввід тексту користувача

Цей етап включає введення тексту у вигляді файлу та передача локації цього файлу у вигляді параметру консолі. Користувач має висловити всю необхідну технічно та бізнес інформацію про майбутній проект.

```
package com.bramix.project.generator.model;


public record CustomerRequest(
    String businessRequest,
    String technicalRequest
) {}
```

Рисунок 3.1.1 – приклад контракту для вводу інформації

```
if (args.length == 0) {
    System.out.println("Usage: java <file_path>");
    return;
}
String filePath = args[0];

File file = new File(args[0]);

if (!file.exists()) {
    System.out.println("File not found: " + filePath);
}

var customerRequest = objectMapper.readValue(file, CustomerRequest.class);
```

Рисунок 3.1.2 – Зчитування вводу користувача

2. NLP аналіз запиту користувача

Токенізація є важливою частиною роботи нашої системи і включає розділення тексту на окремі токени, які є базовими одиницями обробки тексту. В рамках нашої роботи токенізація дозволяє розділити текст на окремі слова. Це важливо для подальшого аналізу, так як більшість алгоритмів та моделей працюють на рівні окремих слів. Великі тексти можуть містити комбінації пробілів, табуляцій і розділових знаків, а токенізація допомагає

стандартизувати цей формат, спрощуючи подальшу обробку тексту. Розподіл тексту на токени допомагає визначити структуру речення та взаємодіяти з граматичними особливостями, що важливо для більш складних завдань, таких як синтаксичний аналіз. Також наша модель працює на рівні слів, тому токенізація є необхідною для підготовки даних. Також це розбити текст на менші одиниці, що сприяє зменшенню обсягу даних та полегшує їх подальше оброблення:

```
import opennlp.tools.tokenize.SimpleTokenizer;
import opennlp.tools.tokenize.Tokenizer;
import opennlp.tools.tokenize.TokenizerModel;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;

public class TokenizationService {
    private Tokenizer tokenizer;

    public TokenizationService() {
        try (InputStream modelIn = new FileInputStream("en-token.bin")) {
            TokenizerModel model = new TokenizerModel(modelIn);
            tokenizer = SimpleTokenizer.INSTANCE;
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public String[] tokenize(String inputText) {
        return tokenizer.tokenize(inputText);
    }
}
```

Рисунок 3.1.3 – Токенізація введеного користувачем тексту

2.1 Семантичний аналіз:

Застосування NLP-моделі для розуміння семантики тексту, визначення ключових слів та вираження основних ідей.
Для семантичного аналізу тексту, щоб перевірити, чи містить введений користувачем текст негативну інформацію, можна використовувати аналіз тональності тексту. В Apache OpenNLP такі функціональності напряму вбудовані. Для семантичного аналізу тексту, щоб перевірити, чи містить введений користувачем текст негативну інформацію, можна використовувати

аналіз тональності тексту. В Apache OpenNLP такі функціональності напряму вбудовані. Нижче наведено приклад простого сервісу на Java, який використовує Apache OpenNLP для семантичного аналізу та перевірки наявності негативної інформації в тексті. Для аналізу тональності використовується простий підхід - підрахунок кількості негативних слів та порівняння їх з загальною кількістю слів.

```
import opennlp.tools.sentiment.SentimentModel;
import opennlp.tools.sentiment.SentimentME;
import opennlp.tools.sentiment.SentimentModelLoader;
import opennlp.tools.tokenize.SimpleTokenizer;
import opennlp.tools.tokenize.Tokenizer;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

public class SentimentAnalysisService {
    private SentimentME sentimentAnalyzer;
    private Tokenizer tokenizer;

    public SentimentAnalysisService() {
        try (InputStream modelIn = new FileInputStream("en-sentiment-model.bin")) {
            SentimentModel sentimentModel = new SentimentModelLoader().load(modelIn);
            sentimentAnalyzer = new SentimentME(sentimentModel);
            tokenizer = SimpleTokenizer.INSTANCE;
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public String analyzeSentiment(String inputText) {
        String[] tokens = tokenizer.tokenize(inputText);
        double[] outcomes = sentimentAnalyzer.predict(tokens);
        return outcomes[0] > outcomes[1]);
    }
}
```

Рисунок 3.1.4 – Семантичний аналіз

2.2 Визначення ключових слів:

Розбиття тексту на ключові слова є важливим етапом і має ряд важливих функцій. Ключові слова можуть вказувати на основні теми або концепції, які важливі для розуміння змісту тексту. Вони є коротким відображенням основного змісту. Основні слова можуть включати емоційно заряджені терміни, які важливі при аналізі тональності тексту та визначенні його емоційного відтінку. Виділення ключових слів полегшує виявлення зв'язків та

відносин між різними частинами тексту. Це важливо для розуміння контексту та взаємодії слів. Зведення тексту до ключових слів дозволяє отримати компактне представлення інформації, що полегшує подальший аналіз та розуміння сутності тексту. Використання ключових слів може спростити процес індексації та пошуку тексту, забезпечуючи швидкий доступ до релевантної інформації. Ключові слова є основними складовими для створення керованого підсумкового висловлювання, що відображає основний зміст тексту в кількох словах

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class KeywordExtractionService {
    private Tokenizer tokenizer;
    private POSTaggerME posTagger;

    public KeywordExtractionService() {
        try (InputStream tokenizerModelIn = new FileInputStream("en-token.bin");
             InputStream posModelIn = new FileInputStream("en-pos-maxent.bin")) {

            TokenizerModel tokenizerModel = new TokenizerModel(tokenizerModelIn);
            POSModel posModel = new POSModel(posModelIn);

            tokenizer = SimpleTokenizer.INSTANCE;
            posTagger = new POSTaggerME(posModel);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public List<String> extractKeywords(String inputText) {
        String[] tokens = tokenizer.tokenize(inputText);
        String[] tags = posTagger.tag(tokens);
        List<String> keywords = new ArrayList<>();
        for (int i = 0; i < tokens.length; i++) {
            if (tags[i].startsWith("NN") || tags[i].startsWith("VB")) {
                keywords.add(tokens[i]);
            }
        }
        return keywords;
    }
}
```

Рисунок 3.1.5 – Отримання ключових слів

Етап 4: Використання OpenAI для отримання відповідей

4.1 Звернення до OpenAI:

- Передача сформованих під запитів до моделі OpenAI для отримання прикладів коду.

```
LinkedHashMap<String, Integer> scoreMap = textAnalyzerService.getWordFrequencyMap();
String summary = textAnalyzerService.generateSummary(customerRequest.businessRequest(), 10);
Double sentimentRes = SENTIMENT_SERVICE.calculateWeightedAverageSentiment(
    SENTIMENT_SERVICE.analyzeSentimentScores(SENTIMENT_SERVICE.stringifySentiments(scoreMap)),
    SENTIMENT_SERVICE.extractSentimentWeights(scoreMap));

if (sentimentRes == "NEGATIVE") throw new IllegalArgumentException("provided file contains negative cotnext");
var keywords = textAnalyzerService.serviceExtract(customerRequest);

var openAIRequest = chatRequestMapper.mapToOpenAIRequest(scoreMap, summary, sentimentRes, keywords);
var response = openAIService.request(openAIRequest);
openAIResponseMapper.map(response).forEach(
    it -> {
      try {
        it.file().getParentFile().mkdirs();
        it.file().createNewFile();
        Files.write(it.content(), it.file(), StandardCharsets.UTF_8);
      } catch (IOException e) {
        throw new RuntimeException(e);
      }
    }
);
```

Рисунок 3.1.6 – генерація запиту до OpenAI

4.2 Обробка результатів:

- Аналіз та обробка відповідей для подальшої інтеграції в комплексний опис.

```
public class OpenAIResponseMapper {

  private String PATH = ProjectGeneratorApplication.getPathFromEnvVariable();

  public List<ProgrammingCodeModel> map(ChatResponse chatResponse) {
    return Arrays.stream(chatResponse.choices().stream()
        .findFirst()
        .orElseThrow().message()
        .content()
            .replace("package", "#!#package")
            .split("#!#"))
        .filter(it -> !it.isEmpty())
        .map(it -> new ProgrammingCodeModel(new File(PATH + it.split("\n")[0]
            .replace("import", "")
            .replace("package", "")
            .replace(" ", "")
            .replace(";", "")
            .replace(".", "\\") + ".java"), it))
        .collect(Collectors.toList());
  }
}
```

Рисунок 3.1.7 – обробка відповіді з OpenAI

Етап 5: Запис результату

5.2 Виведення результату:

- Збереження коду у вигляді файлів проекту

```
openAIResponseMapper.map(response).forEach(
        it -> {
          try {
            it.file().getParentFile().mkdirs();
            it.file().createNewFile();
            Files.write(it.content(), it.file(), StandardCharsets.UTF_8);
          } catch (IOException e) {
            throw new RuntimeException(e);
          }
        }
    );
```

Рисунок 3.1.8 – Збереження відповіді з open AI

3.2 Формування навчальних та тестових даних

Навчання на основі тестових даних — це процес використання алгоритмів машинного навчання для вдосконалення ефективності та точності моделі шляхом аналізу та опрацювання тестових наборів даних. Цей підхід грає важливу роль у вдосконаленні роботи моделей та адаптації їх до конкретних завдань. Навчання на основі тестових даних є критично важливим етапом у розвитку та оптимізації систем штучного інтелекту. Це дозволяє моделям не тільки зберігати актуальність, але і постійно покращувати свої можливості для ефективного вирішення реальних завдань. Ось докладний опис важливості навчання на основі тестових даних:

1. Вдосконалення алгоритмів: Навчання на основі тестових даних дозволяє алгоритмам машинного навчання покращити свої навички та адаптуватися до різноманітних сценаріїв. Моделі можуть вдосконалювати свої прогнозуючі можливості, враховуючи нові дані.

2. Підвищення точності: Аналіз та використання тестових даних дозволяє виявляти та усувати неточності в роботі моделі. Це сприяє підвищенню точності прогнозів та забезпеченню надійності результатів, що важливо у великій кількості вирішуваних завдань.

3. Адаптація до нових умов: Зміни у вихідних даних часто вимагають адаптації моделі. Навчання на основі тестових даних дозволяє моделі пристосовуватися до нових умов і змінюваних вимог.

4. Підтримання актуальності: Технологічні, соціокультурні та економічні зміни можуть впливати на дані та завдання. Навчання на основі тестових даних допомагає забезпечити, що модель залишається актуальною і продуктивною у змінюючомуся оточенні.

5. Оптимізація роботи моделі: Під час навчання на тестових даних моделі можуть оптимізувати свої внутрішні параметри та ваги, щоб досягти кращого вирішення завдань. Це сприяє ефективному використанню ресурсів та прискоренню процесів.

6. Персоналізація рішень: Адаптація до індивідуальних потреб користувачів або конкретних вимог завдань стає можливою завдяки навчанню на тестових даних. Модель може набути унікальних знань, що підвищує персоналізацію рішень.

Розглянемо приклад для тренування моделі в контексті нашої роботи. Створимо приклад текстовий файл з навчальними даними:

```
Розробити та впровадити веб-сайт із системою електронного замовлення товарів.   веб-сайт    NOUN    BUSINESS_REQUEST
Створити інтегровану мобільну програму для моніторингу робочого часу співробітників.    мобільну    ADJ BUSINESS_REQUEST
Провести аудит мережевої інфраструктури для виявлення та усунення можливих загроз безпеці.  мережевої   ADJ TECHNICAL_REQUEST
Розробити та впровадити систему регулярних резервних копій для бази даних клієнтів. регулярних  ADJ TECHNICAL_REQUEST
Впровадити механізм двофакторної автентифікації для забезпечення безпеки користувачів.  двофакторної    ADJ TECHNICAL_REQUEST
Оптимізувати роботу бази даних для ефективного зберігання та доступу до великої кількості даних.    ефективного ADJ TECHNICAL_REQUEST
Створити імплементацію API для взаємодії з системами сторонніх партнерів.   API NOUN    TECHNICAL_REQUEST
Настроїти систему моніторингу та логування для реєстрації аномалій у роботі додатків.   моніторингу NOUN    TECHNICAL_REQUEST
Оновити інфраструктуру серверів для підтримки сучасних протоколів безпеки.  сучасних    ADJ TECHNICAL_REQUEST
Забезпечити масштабованість веб-сайту для ефективної роботи при великому навантаженні.  масштабованість NOUN    TECHNICAL_REQUEST
Створити модуль штучного інтелекту для автоматичного розпізнавання образів у завантажених фотографіях.  штучного    ADJ TECHNICAL_REQUEST
Розробити інтегровану систему керування ресурсами та планування завдань.    інтегровану ADJ BUSINESS_REQUEST
Провести тестування продукту на сумісність з різними операційними системами.    сумісність  NOUN    TECHNICAL_REQUEST
Оптимізувати алгоритм шифрування для забезпечення конфіденційності та надійності.   конфіденційності    NOUN    TECHNICAL_REQUEST
Вдосконалити механізм резервного копіювання для автоматичного виявлення та усунення проблем.    резервного  ADJ TECHNICAL_REQUEST
Провести оновлення системи безпеки для захисту від нових видів загроз.  нових   ADJ TECHNICAL_REQUEST
```

Рисунок 3.2.1 – Навчальні дані

Цей код використовує бібліотеку Apache OpenNLP для розпізнавання іменованих сутностей (Named Entity Recognition - NER) у введеному тексті. Давайте розглянемо, код в рамкая якого здійснюється завантаження навченої моделі NER з файлу, вказаного у змінній modelFilePath. Модель ініціалізується

за допомогою TokenNameFinderModel. Потім Створюється екземпляр NameFinderME, який використовує попередньо навчену модель для розпізнавання іменованих сутностей. Наступним кроком ми використовуємо SimpleTokenizer для розділення введеного тексту на окремі токени, застосовується модель для знаходження іменованих сутностей у токенах тексту. Результат - масив об'єктів **Span**, які представляють знайдені іменовані сутності.

```
import opennlp.tools.util.MarkableFileInputStreamFactory;
import opennlp.tools.namefind.NameSample;
import opennlp.tools.namefind.NameSampleDataStream;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;

public class NERTrainer {

    public static void train(String trainingDataFilePath, String  modelFilePath) throws IOException {

        FileInputStream fileInputStream = new FileInputStream(new File(trainingDataFilePath));
        ObjectStream<String> lineStream = new PlainTextByLineStream(new MarkableFileInputStreamFactory(fileInputStream), "UTF-8");
        ObjectStream<NameSample> sampleStream = new NameSampleDataStream(lineStream);

        TrainingParameters trainingParameters = TrainingParameters.defaultParams();
        trainingParameters.put(TrainingParameters.CUTOFF_PARAM, "5");
        trainingParameters.put(TrainingParameters.ITERATIONS_PARAM, "100");

        TokenNameFinderModel model;
        try {
            model = NameFinderME.train("uk", "ner", sampleStream, trainingParameters, new TokenNameFinderFactory());
        } finally {
            fileInputStream.close();
            lineStream.close();
            sampleStream.close();
        }

        try (FileOutputStream modelOut = new FileOutputStream(new File(modelFilePath))) {
            model.serialize(modelOut);
        }
    }
}
```

Рисунок 3.2.2 – код що відповідає за тренування

Після закінчення тренування можна використовувати її для розпізнавання іменованих сутностей у нових текстах.

```java
public class NERRecognizer {

    public static void recognize(String modelFilePath, String text) throws IOException {

        // Завантаження моделі
        InputStream modelIn = new FileInputStream(new File(modelFilePath));
        TokenNameFinderModel model = new TokenNameFinderModel(modelIn);

        // Ініціалізація NameFinderME
        NameFinderME nameFinder = new NameFinderME(model);

        // Токенізація тексту
        String[] tokens = SimpleTokenizer.INSTANCE.tokenize(text);

        // Знаходження іменованих сутностей
        Span[] spans = nameFinder.find(tokens);

        // Виведення результатів
        for (Span span : spans) {
            System.out.println("Entity: " + span.toString() + " - " + tokens[span.getStart()]);
        }

        // Закриття потоків
        modelIn.close();
    }
}
```

Рисунок 3.2.3 – код що відповідає за знаходження іменованих сутностей

4 ТЕСТУВАННЯ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

Тестування є ключовим етапом у визначенні ефективності та надійності розробленого програмного продукту в рамках магістерської роботи. У цьому відділі проводиться комплексне тестування, спрямоване на підтвердження відповідності розробленого рішення визначеним вимогам та досягненню високої якості в реальних умовах використання.

Для належної роботи проекту, необхідно налагодити конфігураційні параметри, що включають ключі доступу, файли та вихідні директорії.

Першим етапом необхідно надати clientId та clientSecret, які служитимуть ключами доступу до сервісів OpenAI. Ці ключі можна отримати на веб-сайті OpenAI після реєстрації та створення проекту.

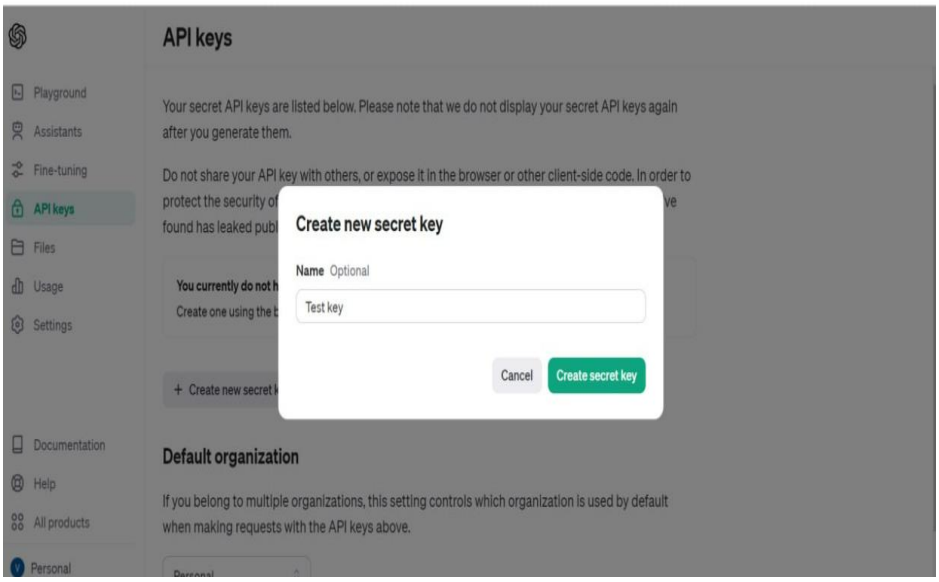


Рисунок 4.1 – Створення Secret key

Також треба вказати шлях до файлу stopWords.txt, який містить список стоп-слів. Ці слова використовуються у фільтрації та обробці тексту для кращого аналізу. Наступним етап – задання вихідної директорії для

збереження створених артефактів та результатів проекту. Всі генеровані файли, моделі та інші результати будуть зберігатися у цій директорії.

```
OPEN_AI_CLIENT_ID: "id_example"
OPEN_AI_CLIENT_SECRET: "secret_example"
STOP_WORDS_PATH: "stopWords.txt"
OUTPUT_PROJECT_PATH: "./"
```

Рисунок 4.2 – Приклад конфігураційних параметрів

Наступним етапом є створення вхідних даних: бізнес та технічних вимог. Бізнес вимоги повинні містити текстові описи бізнес-реквайрментів для інформаційно-технічного проекту. Ці запитання можуть включати стратегічні та функціональні вимоги замовника проекту. Вони повинні чітко визначати очікувані результати та бажані функціональності системи. Приклад бізнес вимоги: “створити зручний інтерфейс користувача для замовлення товарів”. Технічні вимоги повинні включати технічні вимоги та деталі, які стосуються реалізації проекту. Це може включати в себе технічні аспекти, такі як вибір технологій, конфігурацію серверів, застосування алгоритмів, безпеку даних та інші технічні аспекти розробки проекту. Приклад технічної вимоги: “використати мову програмування Java для розробки бекенду системи”

```
{
  "businessRequests" : [
      "The application should generate random password every time."
       "The application should validate generated password."
       "The application should have example of usage. For example, print a result to console"
  ],

  "technicalRequests" : [
    "Use java 17 for resolving of this problem"
  ]
}
```

Рисунок 4.3 — Вхідні дані системи

Для запуску необхідно виконати наступну команду: java -jar your_project.jar input_file.txt

Розглянемо логи під час роботи технології.

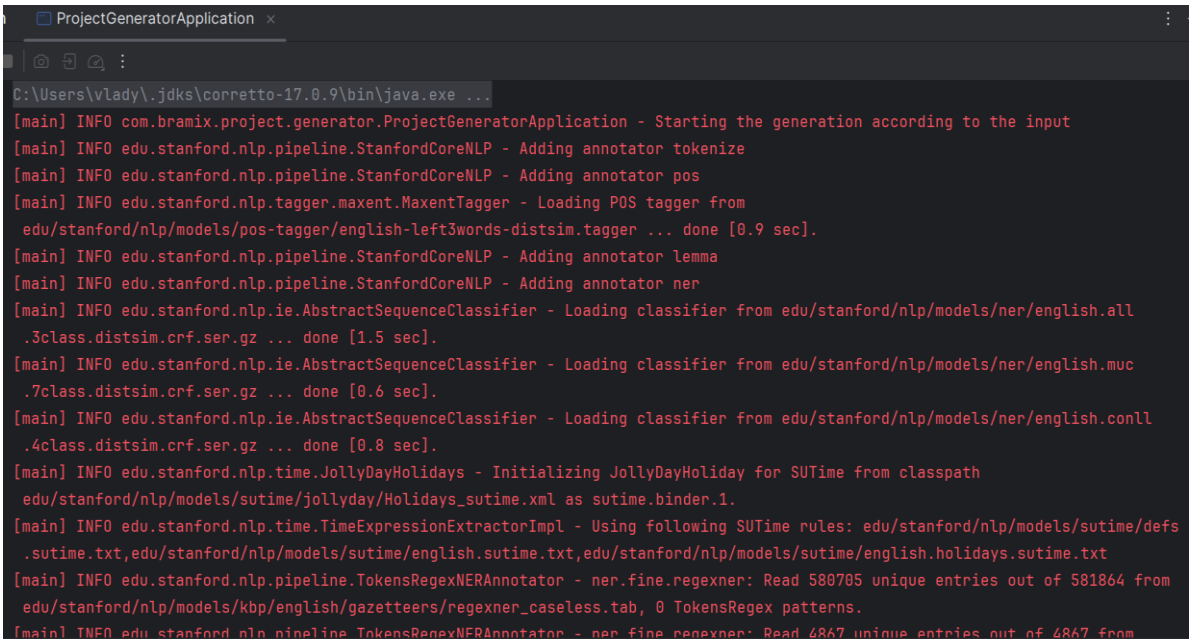


Рисунок 4.4 — Логи роботи технології

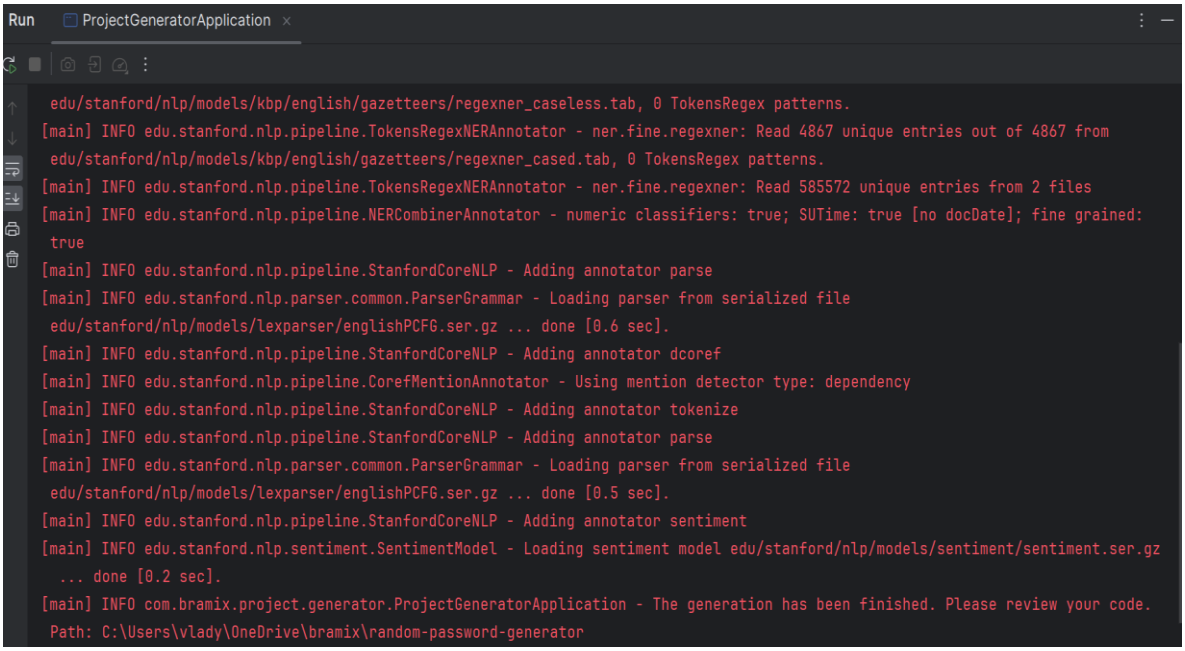


Рисунок 4.5 — Логи роботи технології. Друга частина

В кінці логів бачимо повідомлення про успішне завершення роботи з посиланням на створений проект. В результаті роботи інструменту було згенеровано наступну структуру проекту:

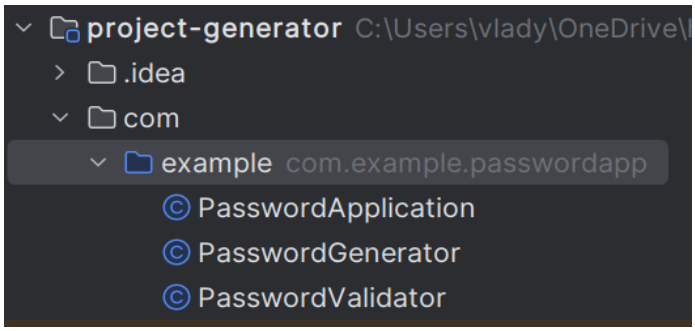


Рисунок 4.6 — Структура згенерованого проету

Розглянемо код кожного файлу більш детально.

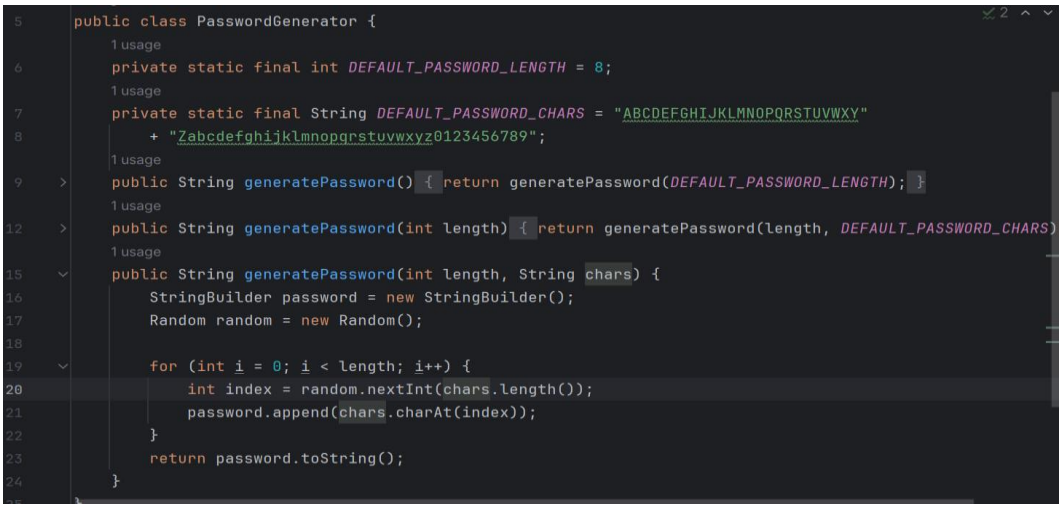
```java
package com.example.passwordapp;

public class PasswordApplication {
    public static void main(String[] args) {
        PasswordGenerator generator = new PasswordGenerator();
        String password = generator.generatePassword();

        PasswordValidator validator = new PasswordValidator();
        boolean isValid = validator.isValid(password);

        System.out.println("Generated Password: " + password);
        System.out.println("Is Password Valid? " + isValid);
    }
}
```

Рисунок 4.7 — Main згенерованого коду



```
public class PasswordGenerator {
    1 usage
    private static final int DEFAULT_PASSWORD_LENGTH = 8;
    1 usage
    private static final String DEFAULT_PASSWORD_CHARS = "ABCDEFGHIJKLMNOPQRSTUVWXY"
        + "Zabcdefghijklmnopqrstuvwxyz0123456789";
    1 usage
    public String generatePassword() { return generatePassword(DEFAULT_PASSWORD_LENGTH); }
    1 usage
    public String generatePassword(int length) { return generatePassword(length, DEFAULT_PASSWORD_CHARS)
    1 usage
    public String generatePassword(int length, String chars) {
        StringBuilder password = new StringBuilder();
        Random random = new Random();

        for (int i = 0; i < length; i++) {
            int index = random.nextInt(chars.length());
            password.append(chars.charAt(index));
        }
        return password.toString();
    }
```

Рисунок 4.8 — Класс PasswordGenerator

```
public class PasswordValidator {
    1 usage
    private static final int MIN_LENGTH = 8;
    1 usage
    private static final int MAX_LENGTH = 20;

    1 usage
    public boolean isValid(String password) {
        return hasValidLength(password)
            && hasValidCharacters(password);
    }
    1 usage
    private boolean hasValidLength(String password) {
        int length = password.length();
        return length >= MIN_LENGTH && length <= MAX_LENGTH;
    }

    1 usage
    private boolean hasValidCharacters(String password) {
        String pattern = "^[a-zA-Z0-9]+$";
        return password.matches(pattern);
    }
```

Рисунок 4.9 — Класс PasswordGenerator

Однією з вимог до системи було навести приклад роботи згенерованого коду шляхом виводу результатів в консоль. Запустимо отриманий проект і отримаємо наступні логи:

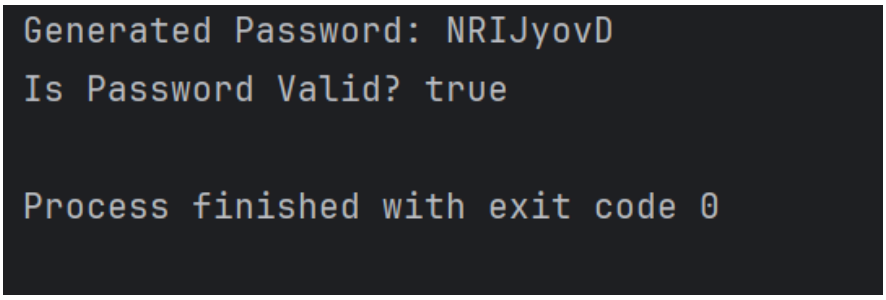


Рисунок 4.10 — Логи згенерованого продукту

ВИСНОВКИ

В даній магістерській кваліфікаційній роботі розроблено та впроваджено інформаційну технологію автоматизації розробки мікросервісів та програмних бібліотек на основі штучного інтелекту OpenAI. Запроваджено програмний продукт з використанням мови програмування Java, а також алгоритмів NLP в основі технології. Проект надає додатковий інструментарій розробникам, зробивши реалізацію рутинних задач з імплементації бізнес проектів більш простим.

Перший розділ дослідження присвячений огляду таких аспектів, як загальне вивчення предметної області, рецензування існуючих рішень, визначення переваг і недоліків. Це дозволяє оцінити потребу та актуальність створення власного продукту. Другий розділ розглядає важливі аспекти, такі як вибір програмних засобів, алгоритмів розроблення, методів оброблення природної мови (NLP), характеристики процесів токенізації і приділяє увагу характеристикам моделей машинного та глибокого навчання. Наступна глава фокусується на реалізації та навчанні запропонованої технології. У четвертому розділі продукт тестується на відповідність встановленим вимогам та перевіряється на правильність отриманих результатів.

В результаті даної роботи нами було розроблено та протестовано інформаційну технологію автоматизації розробки мікросервісів та програмних бібліотек. Розроблена технологія має перспективи для впровадження в реальні умови процесу імплементації IT рішень. Вона автоматизує, спрощує та прискорює процес виконання багатьох рутинних задач.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Agarwal, B., Mittal, N., Bansal, P., Garg, S.: Sentiment analysis using common-sense and context information. Comput. Intell. Neurosci. 2015. — 112 с.

2. Regression analysis/ Kanishka Tyagi — London: Academic Press, 2022. — 158 с.

3. Comparative study of regressor and classifier with decision tree using modern tools/ Jitendra Singh Kushwah — London: Proceedings, 2022. — 195 с.

4. Renu Khandelwal, Evaluating performance of an object detection model, 2020. URL: https://towardsdatascience.com/evaluating-performance-of-an-object-detection-model-137a349c517b (дата звернення 17.12.2023).

5. Tokenization in NLP: Types, Challenges, Examples, Tools [Електронний ресурс]. — Режим доступу: https://neptune.ai/blog/tokenization-in-nlp – (дата звернення 21.11.2023) — Назва з титулу екрана.

6. A Simple Explanation of the Bag-of-Words Model [Електронний ресурс]. — Режим доступу: https://victorzhou.com/blog/bag-of-words/ – (дата звернення 11.11.2023) — Назва з титулу екрана.

7. Методи векторизації текстів [Електронний ресурс]. — Режим доступу: https://medium.com/@bigdataschool/4f8ac90e4175a – (дата звернення 21.11.2023) — Назва з титулу екрана.

8. Understanding TF-IDF for Machine Learning [Електронний ресурс]. — Режим доступу: https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/ – (дата звернення 21.11.2023) — Назва з титулу екрана.

9. Sumeet Kumar Agrawal, Metrics to Evaluate your Classification Model to take the right decisions, 2023. URL: https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/ (дата звернення 15.12.2023).

10. Семантика і технологія Word2Vec [Електронний ресурс]. —

Режим доступу: https://habr.com/ru/articles/585838/ – (дата звернення 05.11.2023) — Назва з титулу екрана.

11. Amnon Geifman, The Correct Way to Measure Inference Time of Deep Neural Networks, 2023. URL: https://deci.ai/blog/measure-inference-time-deep-neural-networks/ (дата звернення 08.12.2023).

12. Howard, Andrew G, Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017. 3-5 с. URL: https://arxiv.org/abs/1704.04861 (дата звернення 07.12.2023).

13. Top 5 Predictive Analytics Models and Algorithms [Електронний ресурс]. — Режим доступу: https://insightsoftware.com/blog/top-5-predictive-analytics-models-and-algorithms/ – (дата звернення 21.11.2023) — Назва з титулу екрана.

14. Компоненти синтаксису IDEF0 [Електронний ресурс]. — Режим доступу: https://elib.lntu.edu.ua/sites/default/files/elib_upload/page9.html – (дата звернення 21.11.2023) — Назва з титулу екрана.

15. Copilot [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/features/copilot. (дата звернення 01.12.2023)

16. Tabnine [Електронний ресурс] – Режим доступу до ресурсу: https://www.tabnine.com. (дата звернення 01.12.2023)

17. Stackoverflow [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com. (дата звернення 15.12.2023).

18. Draw.io [Електронний ресурс] – Режим доступу до ресурсу: https://app.diagrams.net/(дата звернення 15.12.2023).

19. Ningning Ma, Xiangyu Zhang, Light-weight CNN Architecture Design for Fast Inference, 2018. 5-12 с. URL: https://openaccess.thecvf.com/content_ECCV_2018/papers/Ningning_Light-weight_CNN_Architecture_ECCV_2018_paper.pdf (дата звернення 17.12.2023).

ДОДАТКИ

Додаток А

Класс ProjectGeneratorApplication

```
package com.bramix.project.generator;

import com.bramix.project.generator.mapper.OpenAIResponseMapper;
import com.bramix.project.generator.model.CustomerRequest;
import com.bramix.project.generator.model.request.ChatRequest;
import com.bramix.project.generator.model.request.Message;
import com.bramix.project.generator.model.request.OpenAIRequest;
import com.bramix.project.generator.service.OpenAIService;
import com.bramix.project.generator.service.SentimentService;
import com.bramix.project.generator.service.TextAnalyzerService;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.SneakyThrows;
import org.nd4j.shade.guava.io.Files;

import java.io.File;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.LinkedHashMap;
import java.util.List;

public class ProjectGeneratorApplication {

  private final static ObjectMapper objectMapper = new ObjectMapper();
  private final static SentimentService SENTIMENT_SERVICE = new SentimentService();

  private final static TextAnalyzerService textAnalyzerService = new TextAnalyzerService();

  private final static OpenAIResponseMapper openAIResponseMapper = new
OpenAIResponseMapper();

  private final static OpenAIService openAIService = new OpenAIService();
  @SneakyThrows
  public static void main(String[] args) {
    if (args.length == 0) {
      System.out.println("Usage: java <file_path>");
      return;
    }
    String filePath = args[0];

    File file = new File(args[0]);

    if (!file.exists()) {
      System.out.println("File not found: " + filePath);
    }
```

```
    var customerRequest = objectMapper.readValue(file, CustomerRequest.class);

    LinkedHashMap<String, Integer> scoreMap =
textAnalyzerService.getWordFrequencyMap();
    String summary = textAnalyzerService.generateSummary(customerRequest.businessRequest(),
10);
    Double sentimentRes =
SENTIMENT_SERVICE.calculateWeightedAverageSentiment(SENTIMENT_SERVICE.analyzeS
entimentScores(SENTIMENT_SERVICE.stringifySentiments(scoreMap)),
SENTIMENT_SERVICE.extractSentimentWeights(scoreMap));

    var openAIRequest = new OpenAIRequest(customerRequest.businessRequest(),
customerRequest.technicalRequest(), summary);
    // if (sentimentRes == 1)
    ChatRequest request = new ChatRequest("gpt-3.5-turbo", List.of(new Message[]{new
Message("user", openAIRequest.toString())}), 1, 1);

    var response = openAIService.request(request);

    openAIResponseMapper.map(response).forEach(
        it -> {
          try {
            it.file().getParentFile().mkdirs();
            it.file().createNewFile();
            Files.write(it.content(), it.file(), StandardCharsets.UTF_8);
          } catch (IOException e) {
            throw new RuntimeException(e);
          }
        }
    );
  }
}
```

Класс TextAnalyzerService

```
package com.bramix.project.generator.service;

import edu.stanford.nlp.ling.CoreAnnotations;
import edu.stanford.nlp.ling.CoreLabel;
import edu.stanford.nlp.pipeline.Annotation;
import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import edu.stanford.nlp.util.CoreMap;
import lombok.Getter;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

@Getter
```

```
public class TextAnalyzerService {
  private final Set<String> stopWords = loadStopWords();
  private final LinkedHashMap<String, Integer> wordFrequencyMap = new
LinkedHashMap<>();

  public String generateSummary(String cleanText, int maxSentences) {
   LinkedHashMap<String, Integer>  sentenceScoreMap = calculateSentenceScores(cleanText);
   TreeMap<Integer, ArrayList<String>> sortedSentenceMap =
buildSortedSentenceMap(sentenceScoreMap);

   Iterator<Map.Entry<Integer, ArrayList<String>>> sortedMapIterator =
sortedSentenceMap.entrySet().iterator();
   ArrayList<String> finalSentences = new ArrayList<>();
   int sentenceCounter = 0;

   while (sortedMapIterator.hasNext() && sentenceCounter < maxSentences) {
    Map.Entry<Integer, ArrayList<String>> scoreAndSentences = sortedMapIterator.next();
    ArrayList<String> sentencesInList = new ArrayList<>(scoreAndSentences.getValue());

    for (String sentence : sentencesInList) {
     finalSentences.add(sentence);
     sentenceCounter++;

     if (sentenceCounter >= maxSentences) {
      break;
     }aa
    }
   }

   StringBuilder summary = new StringBuilder();
   int counter = 0;
   Iterator<Map.Entry<String, Integer>> scoreMapIterator =
sentenceScoreMap.entrySet().iterator();

   while (scoreMapIterator.hasNext() && counter < maxSentences) {
    Map.Entry<String, Integer> sentenceAndScore = scoreMapIterator.next();

    if (finalSentences.contains(sentenceAndScore.getKey())) {
     summary.append(sentenceAndScore.getKey()).append("\n");
     counter++;
    }
   }

   return summary.toString();
  }

  private LinkedHashMap<String, Integer> calculateSentenceScores(String text) {
   Properties properties = new Properties();
   properties.setProperty("annotators", "tokenize, ssplit, pos, lemma, ner, parse, dcoref");
   StanfordCoreNLP pipeline = new StanfordCoreNLP(properties);
```

```
  Annotation document = new Annotation(text);
  pipeline.annotate(document);

  List<CoreMap> sentences = document.get(CoreAnnotations.SentencesAnnotation.class);
  LinkedHashMap<String, Integer> sentenceScoreMap = new LinkedHashMap<>();

  for (CoreMap sentence : sentences) {
   String sentenceString = sentence.toString();
   sentenceScoreMap.put(sentenceString, 0);
  }

  for (CoreMap sentence : sentences) {
   int score = calculateScoreForSentence(sentence);
   sentenceScoreMap.put(sentence.toString(), score);
  }

  addWordFrequencyWeight(sentenceScoreMap);
  return sentenceScoreMap;
 }

  private TreeMap<Integer, ArrayList<String>>
buildSortedSentenceMap(LinkedHashMap<String, Integer> scoreMap) {
  TreeMap<Integer, ArrayList<String>> sortedMap = new
TreeMap<>(Collections.reverseOrder());

  scoreMap.forEach((sentence, score) -> {
   if (!sortedMap.containsKey(score)) {
    ArrayList<String> sentenceList = new ArrayList<>();
    sentenceList.add(sentence);
    sortedMap.put(score, sentenceList);
   } else {
    ArrayList<String> sentenceList = new ArrayList<>(sortedMap.get(score));
    sentenceList.add(sentence);
    sortedMap.put(score, sentenceList);
   }
  });

  return sortedMap;
 }

 private int calculateScoreForSentence(CoreMap sentence) {
  int score = 0;
  for (CoreLabel token : sentence.get(CoreAnnotations.TokensAnnotation.class)) {
   String namedEntity = token.get(CoreAnnotations.NamedEntityTagAnnotation.class);

   if (getPosTags().contains(namedEntity)) {
    score++;
   }
```

```
  String originalWord = token.get(CoreAnnotations.TextAnnotation.class);
  String word = originalWord.toLowerCase();
  calculateWordFrequency(word);
 }
 return score;
}

private void calculateWordFrequency(String word) {
 if (!this.stopWords.contains(word)) {
  wordFrequencyMap.putIfAbsent(word, 0);
  wordFrequencyMap.put(word, wordFrequencyMap.get(word) + 1);
 }
}

private Set<String> getPosTags() {
 Set<String> posTagsSet = new HashSet<>();
 posTagsSet.add("ORGANIZATION");
 posTagsSet.add("TITLE");
 posTagsSet.add("PERSON");
 posTagsSet.add("CITY");
 posTagsSet.add("LOCATION");
 posTagsSet.add("DATE");
 posTagsSet.add("IDEOLOGY");
 posTagsSet.add("MONEY");
 return posTagsSet;
}

private Set<String> loadStopWords() {
 Set<String> stopWordsSet = new HashSet<>();

 File stopWordsFile = new File("stopWords.txt");
 try (Scanner fileScanner = new Scanner(stopWordsFile)) {
  while (fileScanner.hasNextLine()) {
   String stopWord = fileScanner.nextLine().toLowerCase();
   stopWordsSet.add(stopWord);
  }
 } catch (FileNotFoundException ignored) {
 }

 return stopWordsSet;
}

private LinkedHashMap<String, Integer> addWordFrequencyWeight(LinkedHashMap<String,
Integer> sentenceScoreMap) {
 Set<String> frequentWordSet = new HashSet<>();
 final int MIN_FREQUENCY = 3;

 wordFrequencyMap.forEach((word, frequency) -> {
  if (frequency >= MIN_FREQUENCY) {
   frequentWordSet.add(word);
```

```
    }
   });

   sentenceScoreMap.forEach((sentence, score) -> {
    for (String word : frequentWordSet) {
     if (sentence.toLowerCase().contains(word)) {
      sentenceScoreMap.put(sentence, sentenceScoreMap.get(sentence) + 1);
     }
    }
   });

   return sentenceScoreMap;
  }
}
```

Класс SentimentService

```
package com.bramix.project.generator.service;

import edu.stanford.nlp.ling.CoreAnnotations;
import edu.stanford.nlp.neural.rnn.RNNCoreAnnotations;
import edu.stanford.nlp.pipeline.Annotation;
import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import edu.stanford.nlp.sentiment.SentimentCoreAnnotations;
import edu.stanford.nlp.trees.Tree;
import edu.stanford.nlp.util.CoreMap;

import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.Properties;

public class SentimentService {

  public String stringifySentiments(LinkedHashMap<String, Integer> sentimentScoreMap) {
   StringBuilder resultText = new StringBuilder();

   for (String sentence : sentimentScoreMap.keySet()) {
    resultText.append(" ").append(sentence);
   }
   return resultText.toString().trim();
  }

  public ArrayList<Integer> extractSentimentWeights(LinkedHashMap<String, Integer>
sentimentScoreMap) {

    return new ArrayList<>(sentimentScoreMap.values());
  }
```

```
  public ArrayList<Integer> analyzeSentimentScores(String inputText) {
   ArrayList<Integer> scoresList = new ArrayList<>();
   Properties nlpProperties = new Properties();
   nlpProperties.setProperty("annotators", "tokenize, ssplit, parse, sentiment");
   StanfordCoreNLP nlpPipeline = new StanfordCoreNLP(nlpProperties);

   Annotation nlpAnnotation = nlpPipeline.process(inputText);

   for (CoreMap sentenceElement :
nlpAnnotation.get(CoreAnnotations.SentencesAnnotation.class)) {
    Tree sentimentTree =
sentenceElement.get(SentimentCoreAnnotations.SentimentAnnotatedTree.class);
    scoresList.add(RNNCoreAnnotations.getPredictedClass(sentimentTree));
   }
   return scoresList;
  }

  public Double calculateWeightedAverageSentiment(ArrayList<Integer> sentimentList,
ArrayList<Integer> weights) {
   double weightedAverage = 0;
   double totalWeight = weights.stream().mapToDouble(Integer::doubleValue).sum();

   for (int i = 0; i < sentimentList.size(); i++) {
    double weight = sentimentList.get(i) * weights.get(i) / totalWeight;
    weightedAverage += weight;
   }

   return weightedAverage;
  }

  public Integer interpretSentimentScore(Double weightedAverage) {
   return (int) Math.rint(weightedAverage);
  }
}
```

Класс OpenAIService

```
package com.bramix.project.generator.service;

import com.bramix.project.generator.model.config.OpenAIRestTemplateConfig;
import com.bramix.project.generator.model.request.ChatRequest;
import com.bramix.project.generator.model.response.ChatResponse;

public class OpenAIService {
```

```
  public ChatResponse request(ChatRequest chatRequest) {

    ChatResponse response = OpenAIRestTemplateConfig.openaiRestTemplate()
        .postForObject("https://api.openai.com/v1/chat/completions", chatRequest,
ChatResponse.class);

    if (response == null || response.choices() == null) {
      throw new RuntimeException();
    }
    return response;

  }
}

package com.bramix.project.generator.model;


public record CustomerRequest(
    String businessRequest,
    String technicalRequest
) {}


package com.bramix.project.generator.model.response;

import com.bramix.project.generator.model.request.Message;

public record Choice(
    int index, Message message
) {}



package com.bramix.project.generator.model.response;

import java.util.List;

public record ChatResponse(
    List<Choice> choices
) {

}

package com.bramix.project.generator.model.request;

public record OpenAIRequest(
    String initialBusinessRequirements,
    String initialTechnicalRequirements,
    String businessSummary
```

```
) {

  @Override
  public String toString() {
    return "Provide a real code examples. Don't provide any descriptions for code snippets. Use
OOP single responsibility and separate a code to different classes. Provide a full code like full
class name and package + classname.java before every code snippet for every class" +
"BusinessRequirements='" + initialBusinessRequirements + '\''
        + ", TTechnicalRequirements='" + initialTechnicalRequirements + '\'' + ",
businessSummary='"
        + businessSummary + '\'' + '}';
  }
}

package com.bramix.project.generator.model.request;

public record Message (
    String role,
    String content) {}

package com.bramix.project.generator.model.request;

import java.util.List;

public record ChatRequest(
    String model,
    List<Message> messages,
    int n,
    double temperature
) {}

package com.bramix.project.generator.mapper;

import java.io.File;

public record ProgrammingCodeModel(
    File file,
    String content
) {

}
```

Класс OpenAIResponseMapper

```
package com.bramix.project.generator.mapper;

import com.bramix.project.generator.model.response.ChatResponse;

import java.io.File;
import java.util.Arrays;
```

```
import java.util.List;
import java.util.stream.Collectors;

public class OpenAIResponseMapper {

  private String PATH = parseInputPath();

  public List<ProgrammingCodeModel> map(ChatResponse chatResponse) {
    return Arrays.stream(chatResponse.choices().stream()
        .findFirst()
        .orElseThrow().message()
        .content()
           .replace("package", "#!#package")
           .split("#!#"))
        .filter(it -> !it.isEmpty())
        .map(it -> new ProgrammingCodeModel(new File(PATH + it.split("\n")[0]
           .replace("import", "")
           .replace("package", "")
           .replace(" ", "")
           .replace(";", "")
           .replace(".", "\\") + ".java"), it))
        .collect(Collectors.toList());
  }
}
```