

ДЕРЖАВНИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Кафедра інженерії програмного забезпечення та кібербезпеки

*Захищено на кафедрі інженерії
програмного забезпечення та
кібербезпеки*

«16» травня 2023р.

з оцінкою _____

Підпис членів комісії:

КУРСОВА РОБОТА

з дисципліни

«ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»

НА ТЕМУ:

**«Розробка програмного забезпечення з генерації тексту за ключовим
словом «Generatext» мовою програмування C#»**

Виконала: студентка факультету
інформаційних технологій
4 групи 2 курсу
Аверіної Наталії Ігорівни

Науковий керівник:
доцент, кандидат політичних наук, доцент
кафедри інженерії програмного
забезпечення та кібербезпеки
Чубасєвський Віталій Іванович

Київ 2023

Державний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки


КАРТКА

завдання та контролю за ходом виконання курсової роботи

1. Аверіна Н.І.
2. ФІТ
3. 2 курс 4 група
4. Форма навчання денна
5. Номер залікової книжки _____
6. Назва теми курсової роботи «Розробка програмного забезпечення з генерації тексту за ключовим словом «Generatext» мовою програмування C#»

№	Назва етапів курсової роботи	Строк виконання етапів роботи		Підпис	
		За планом	Фактично	Студента	Відповідальної особи
1.	Погодження теми з науковим керівником	14.02.2023	14.02.2023		
2.	Затвердження плану роботи	23.02.2023	23.02.2023		
3.	Виконання теоретичної частини роботи (1 та 2 розділи)	16.03.2023	16.03.2023		
4.	Виконання практичної частини роботи (3 розділ)	17.04.2023	17.04.2023		
5.	Оформлення загальної структури курсової роботи	01.05.2023	01.05.2023		
6.	Дата подання виконаної роботи на кафедру	01.05.2023	01.05.2023		
7.	Дата захисту роботи	16.05.2023	16.05.2023		

7. Тему, план і строки прийняв до виконання _____ (підпис студента)

8. Науковий керівник  _____ (прізвище та підпис)

“ ____ ” _____ 20__ р.

ЗМІСТ

ВСТУП	5
РОЗДІЛ 1 АНАЛІЗ МЕТОДІВ ОБРОБКИ ПРИРОДНОЇ МОВИ ТА ПОШУК ЕФЕКТИВНОГО АЛГОРИТМУ З ГЕНЕРАЦІЇ ТЕКСТУ	7
1.1. Історія розвитку розробок з генерації тексту.....	7
1.2. Аналіз методів обробки природної мови	8
1.3. Пошук ефективного алгоритму з генерації тексту	10
Висновки до Розділу 1	11
РОЗДІЛ 2 ПОРІВНЯЛЬНИЙ АНАЛІЗ НАЯВНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ	13
2.1 Порівняльний аналіз конкурентноспроможних на ринку продуктів з генерації тексту за ключовим словом.....	13
2.2 Вибір середовища та інструментів розробки	17
Висновки до Розділу 2	21
РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ «Generatext»	22
3.1. Технології, використані для розробки продукту	22
3.2. Інтерфейс користувача	31
Висновок до Розділу 3	32
ВИСНОВКИ.....	33
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	34
ДОДАТКИ.....	35

АНОТАЦІЯ

на курсову роботу: «Розробка програмного забезпечення з генерації тексту за ключовим словом «Generatext» мовою програмування C#»

Курсова робота складається: 40 сторінок, 21 рисунок, 1 додаток.

Розроблений програмний продукт з генерації тексту передбачає введення користувачем ключового слова та отримання результату у вигляді згенерованого речення в інтерпретаторі командного рядка.

The developed software product for text generation involves user input of a keyword and receiving a generated sentence in the command-line interpreter.

ВСТУП

В сучасному світі інформаційних технологій стрімко набуває популярності генерація тексту на основі ключового слова. Це процес автоматичної генерації текстових даних з використанням алгоритмів машинного навчання, що дозволяє створювати великі обсяги тексту з мінімальною людською участю.

Цей підхід знайшов своє застосування в багатьох сферах, таких як соціальні мережі, засоби масової інформації, рекламні компанії, інформаційні портали, блоги та багато інших. Генерація тексту на основі ключового слова є важливим інструментом у веб-розробці та цифровому маркетингу, адже дозволяє автоматично створювати унікальний контент, що збільшує рейтинг сторінок у пошукових системах. Тож використання цієї технології дозволяє ефективно і швидко генерувати текстові дані на різноманітні теми, що значно підвищує продуктивність та ефективність роботи в різних сферах діяльності.

Оскільки генерація тексту на основі ключового слова є складним процесом, що вимагає обробки природної мови, у цій роботі будуть досліджені різні підходи до створення таких систем та їх ефективність. Також будуть проаналізовані можливості використання різних алгоритмів та моделей машинного навчання, що дозволять покращити якість генерації тексту на основі ключового слова.

Завдання курсової роботи:

- дослідити моделі та методи генерації тексту;
- проаналізувати наявні на ринку додатки-конкуренти;
- розробити додаток, який дозволяв би генерувати текст за введеним ключовим словом від користувача.

Метою даної курсової роботи є дослідження можливостей застосування методів генерації тексту на основі ключового слова та розробка програмного додатку з генерації правдоподібного тексту з використанням цих методів.

Об'єктом дослідження є алгоритми та моделі, що дозволяють автоматично генерувати текст на основі заданого ключового слова.

Предмет дослідження: розробка програмного забезпечення з генерації тексту за ключовим словом.

Під час виконання проєкту буде створений додаток під назвою «Generatext» мовою програмування C# з використанням платформи .NET Framework.

РОЗДІЛ 1

АНАЛІЗ МЕТОДІВ ОБРОБКИ ПРИРОДНОЇ МОВИ ТА ПОШУК ЕФЕКТИВНОГО АЛГОРИТМУ З ГЕНЕРАЦІЇ ТЕКСТУ

1.1. Історія розвитку розробок з генерації тексту

Історія генерації тексту сягає середини 20 століття, коли було створено перші системи, що використовували шаблони для автоматичного створення тексту.

У 1950-х роках була започаткована робота над програмою під назвою «Random Sentence Generator», що генерувала випадкові речення за допомогою комп'ютера. Методом генерації був випадковий вибір слів із заданого словника [3].

У 1970-х роках Террі Віноградом в рамках його докторської дисертації в Массачусетському технологічному інституті була створена комп'ютерна програма SHRDLU. Метою було дослідження можливостей комп'ютера в розумінні мовлення людини і його взаємодії з навколишнім світом через мову. Система мала можливість генерації речень на основі правил граматики [4].

У 1980-х роках з'явилася можливість використання статистичних методів для генерації тексту. Один з перших проєктів, що використовував цей підхід, був розроблений групою дослідників у Каліфорнійському університеті в Берклі. Вони використали статистичні моделі для генерації заголовків новин, що були схожі на ті, що створюють журналісти [5]. Цей проєкт використовував статистичні методи для аналізу структури речень та використання цієї інформації для генерації тексту. Зокрема, вони використовували статистичні моделі для визначення того, які слова повинні бути використані в реченні, та які слова повинні йти разом. Наприклад, якщо модель бачила слово «поліція», то вона вважала, що ймовірно також з'являться слова «арешт», «злочинець» та «слідство» [6].

З появою штучного інтелекту та зростанням обчислювальної потужності розпочався новий етап розвитку генерації тексту. Сьогодні, методи машинного навчання та обробки природної мови є ключовими інструментами для генерації тексту. Один з таких методів – n-грам модель, яка використовується для прогнозування наступного слова в тексті з урахуванням його попередніх слів.

Зараз активно розробляються нові методи генерації тексту, такі як глибокі нейронні мережі та генеративні моделі, що використовуються для створення більш точних та якісних текстів. Крім того, з'являються нові області застосування генерації тексту, такі як автоматичне створення коду програм та складання музики.

1.2. Аналіз методів обробки природної мови

Методи обробки природної мови (англ. Natural Language Processing) – це галузь комп'ютерної лінгвістики, яка досліджує можливості застосування комп'ютерних алгоритмів для аналізу та розуміння природної мови. Для розвитку генерації тексту за ключовим словом та інших супутніх технологій важливим є використання методів обробки природної мови [7].

Основні методи NLP включають у себе:

- Токенізацію: процес розбиття тексту на окремі компоненти, називані токенами, які можуть бути словами, числами, символами пунктуації тощо. Цей процес є важливим етапом у багатьох проєктах з обробки природної мови, оскільки вона дозволяє перетворити текст на структурований формат, який можна подальше обробляти та аналізувати. Наприклад, використання tokenів є важливим етапом для створення n-грам, що дозволяє аналізувати залежності між словами у тексті [7].
- Частиномовний аналіз: процес визначення ролі кожного слова у реченні. Він допомагає розуміти граматичну структуру тексту та робити висновки про зміст [7].

- Синтаксичний аналіз (також відомий як парсинг): процес аналізу тексту на основі граматики, що визначає правила правильної синтаксичної структури мови. Під час синтаксичного аналізу, вхідний текст розбивається на послідовність лексем (токенів), і ці лексеми потім перетворюються в дерево синтаксичного розбору згідно з правилами граматики [7].
- Семантичний аналіз: процес визначення значення слова та його зв'язок з іншими словами у реченні. На етапі семантичного аналізу використовуються різні методи та техніки, такі як семантичні мережі, векторні представлення слів, методи машинного навчання та інші. Для покращення точності семантичного аналізу важливо мати достатньо великий корпус текстів, на яких можна навчати моделі та алгоритми [7].
- Розпізнавання іменованих сутностей: процес визначення в тексті іменованих об'єктів, таких як люди, місця, організації та інші. Ця задача важлива для багатьох застосувань обробки мови, таких як автоматичний переклад, аналіз соціальних медіа, аналіз текстів новин, інформаційний пошук та багато інших. Одним з відомих наборів даних для розпізнавання іменованих сутностей є CoNLL-2003, що містить новинні тексти з позначеними іменованими сутностями. Також існують відкриті бібліотеки, такі як spaCy, NLTK, Stanford NER та інші [7].
- Класифікацію текстів: процес визначення категорії, до якої належить текст. Це важливий етап у багатьох задачах обробки природної мови, таких як фільтрація спаму, аналіз настроїв, категоризація новин, розпізнавання мовленнєвих актів та багато інших [7].
- Аналіз тональності: процес визначення настрою тексту: позитивний, негативний чи нейтральний. Для реалізації аналізу тональності можна використовувати різні бібліотеки та

інструменти, такі як TextBlob, NLTK, Vader Sentiment та багато інших. Застосування аналізу тональності дуже широке, від аналізу відгуків користувачів в інтернет-магазинах до моніторингу соціальних мереж для виявлення настроїв аудиторії щодо певної події чи товару [7].

- Генерацію мовлення: процес створення тексту з допомогою комп'ютерної програми.

Одним з методів генерації мовлення є шаблонний підхід, при якому використовуються заздалегідь написані шаблони речень, які заповнюються відповідними даними. Цей метод досить простий, але не дозволяє створювати складні та оригінальні текстові висловлювання [8].

Ще одним методом є статистичний підхід, при якому комп'ютерний алгоритм аналізує велику кількість текстів та статистичні дані щодо вживання певних слів та фраз в різних контекстах. Потім алгоритм використовує ці дані для створення нових речень, які відповідають заданому контексту. Цей метод може створювати більш складні та оригінальні текстові висловлювання, але його точність залежить від якості та кількості вихідних даних, на основі яких він працює [8].

Іншим методом є глибинне навчання, при якому комп'ютерні алгоритми навчаються на великій кількості даних та самостійно здатні створювати текстові висловлювання на основі отриманих знань. Цей метод може створювати найбільш складні та оригінальні текстові висловлювання, але його точність залежить від якості та кількості навчальних даних [8].

1.3. Пошук ефективного алгоритму з генерації тексту

Пошук ефективного алгоритму для генерації тексту є актуальною проблемою в галузі обробки природних мов. Один з потенційних методів – використання n-грам. N-грами – це метод статистичної обробки природної мови, який використовується для аналізу та генерації тексту. Він базується на розбитті тексту на окремі фрагменти, зазвичай по словам, та аналізу частоти їх вживання

поруч. Наприклад, для створення n-грам на основі трьох слів, розбивають текст на три слова та рахують, скільки разів кожна трійка слів зустрічається в тексті.

Один з ефективних алгоритмів генерації тексту на основі n-грам – це Markov Chain. Він полягає в тому, що збирається велика кількість текстів, які використовуються для тренування моделі, та створюється таблиця з частотами вживання трійок слів. Потім для генерації нового тексту вибирається випадкова трійка слів, знаходиться найбільш ймовірна наступна трійка слів, що може йти після неї, і використовується для створення наступного слова. Цей процес повторюється доти, доки не буде сформований весь текст [6].

В даному проєкті буде використовуватися один з основних методів обробки природної мови – токенизація, та після неї створення n-грам. Програма буде приймати текст, який буде розділено на список слів. Після чого обчислюється кількість входження кожної біграми (пари сусідніх слів) в текст. Результатом є словник біграм, де ключі - перше слово в біграмі, а значення - словник, де ключі - друге слово в біграмі, а значення - кількість входжень біграми в текст. Так само буде зроблено і з триграмами. Результатом має бути словник, де ключі - біграми/триграми, а значення - наступне слово, що найчастіше зустрічається після цієї біграми/триграми.

Висновки до Розділу 1

Генерація тексту є важливою задачею в галузі обробки природної мови. Історія розвитку цієї галузі досліджень показує, що з появою більш потужних обчислювальних систем і набору великої кількості текстових даних з'явилася можливість створювати більш складні моделі для генерації тексту.

Методи обробки природної мови, такі як аналіз тональності, класифікація текстів та генерація мовлення, розвиваються з метою покращення різних аспектів обробки текстів. Пошук ефективного алгоритму з генерації тексту є важливою задачею, яка може бути вирішена за допомогою різних підходів. У цьому контексті використання n-грам є одним з найпоширеніших методів

генерації тексту. Використання n-грам дозволяє моделювати залежності між словами в тексті та генерувати більш природні тексти.

РОЗДІЛ 2

ПОРІВНЯЛЬНИЙ АНАЛІЗ НАЯВНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ

2.1 Порівняльний аналіз конкурентноспроможних на ринку продуктів з генерації тексту за ключовим словом

Для проведення порівняльного аналізу конкурентноспроможних на ринку продуктів з генерації тексту за ключовим словом необхідно спочатку визначитись з конкретними продуктами.

За запитом «text generation» в пошуковій системі Google можна виявити такі популярні програми на ринку:

- Chat GPT - це продукт компанії OpenAI, який базується на глибоких нейронних мережах та забезпечує високу якість генерації тексту. GPT може використовуватись для автоматичної генерації новин, текстів для соціальних мереж, статей, творів, віршів тощо. Моделі GPT написані мовами програмування Python або C++. Зазвичай, для навчання та використання моделей GPT використовуються фреймворки, такі як TensorFlow, PyTorch або Keras.

Переваги:

- Адаптивність. Генератор адаптується до різних контекстів та задач, що забезпечує його універсальність в різних сферах застосування.
- Багатомовність. Chat GPT розпізнає та генерує текст на багатьох мовах світу, що робить його достойним інструментом для використання в інтернаціональних проєктах та звичайним помічником для людей в кожній точці планети.
- Зручний та зрозумілий інтерфейс зроблений у вигляді чату.

- Висока якість генерації тексту. Генератор може вчитися на великих обсягах текстових даних та зберігати ці знання, що дозволяє покращувати якість генерації тексту.

Хоча Chat GPT має багато переваг, але він також має деякі недоліки:

- Модель може генерувати відповіді, які здаються граматично правильними, але зовсім не відповідають запиту.
- Іноді виникають повтори фраз або ідей, які вже були використані.
- Модель може бути вимогливою до обчислювальних ресурсів
- Питання безпеки використання. Chat GPT навчається на великій кількості тексту, включаючи переписки, соціальні мережі та інші джерела. Це означає, що при розмові з Chat GPT можуть виникати ситуації, коли система може отримувати конфіденційну інформацію, таку як персональні дані, фінансову інформацію, медичні записи та інше. Такі дані можуть стати доступними поганцям, якщо система не забезпечує належний рівень безпеки.
- TextGenrnn - це відкрите програмне забезпечення для генерації тексту, яке базується на рекурентних нейронних мережах. Його можна використовувати для генерації різних типів текстів, включаючи новини, вірші, історії та інше. У розробці даного програмного забезпечення була використана мова програмування Python.

Переваги:

- Добре підходить для генерації тексту з довільною темою і стилем.
- Простий та зрозумілий інтерфейс.
- Має можливість використовувати вагові коефіцієнти, що дає змогу змінювати важливість певних слів та фраз при генерації тексту.

Недоліки:

- Недостатньо ефективний при генерації довгих текстів з складною граматикою або технічних текстів.
 - Зазвичай потребує досить велику кількість тренувальних даних для досягнення достатнього рівня точності.
 - Може бути схильний до генерації нелогічних та неправдоподібних фраз або повторів.
 - Обмежена функціональність.
- IBM Watson - це рішення зі штучного інтелекту, яке забезпечує генерацію тексту з використанням природної мови. IBM Watson може використовуватись для автоматичної генерації новин, статей, описів продуктів та іншого. Для аналізу та генерації тексту Watson Natural Language Understanding підтримує багато мов програмування, включаючи Python, Java, C#, JavaScript та інші.

Переваги:

- Широкий спектр інструментів і можливостей для роботи з природною мовою, таких як генерація тексту, розпізнавання мовлення, переклад тексту та інші.
- Можливість використовувати IBM Watson на різних мовах, що робить його більш універсальним і доступним для різних користувачів.
- Інтеграція зі сторонніми системами та додатками, що дозволяє легко використовувати IBM Watson в існуючих проєктах.
- Великий обсяг документації, навчальних матеріалів та ресурсів, що допомагає швидко навчитися користуватися платформою.

Недоліки:

- Обмеження в налаштуванні та кастомізації платформи, що може обмежувати розвиток індивідуальних рішень.

- Питання безпеки та конфіденційності даних при роботі з платформою.
- Складність використання.
- Markovify: це простий генератор тексту, який використовує модель Маркова для генерації тексту. Додаток реалізований на мові програмування Python.

Переваги:

- Легкий у використанні та налаштуванні.
- Може працювати з будь-яким типом текстів та мовами.
- Можливість налаштувати рівень складності моделі, щоб підібрати оптимальне співвідношення між точністю та швидкістю генерації тексту.
- Відкритий код та широкий спектр підтримки.

Недоліки:

- Не здатний згенерувати довгі, складні або логічно зв'язані тексти.
- Не завжди здатний зберегти стиль, тон або специфіку вихідного тексту.
- Не здатний генерувати тексти з відтворенням граматичної правильності.
- Не здатний згенерувати тексти з новими ідеями, що відрізняються від вихідного корпусу.

Створений нами продукт «Generatext», який працює на основі використання n-грамної моделі генерації тексту, буде мати наступні переваги порівняно з конкурентами:

- Висока швидкість обробки. Оскільки додаток «Generatext» має меншу кількість параметрів, використовує просту статистичну модель та не вимагає високопродуктивного обладнання, як

наприклад GPT-3, працювати він буде ефективно та значно швидше.

- Низькі витрати. Розробка генератора тексту на основі n-грам не потребує великих витрат на обладнання та програмне забезпечення, що робить його доступним для широкого кола користувачів.
- Вбудована підтримка мовних моделей. Це дозволяє покращити якість генерації тексту, користуючись найновішими досягненнями у галузі обробки природної мови.
- Дуже простий інтерфейс. Користувачі без досвіду в програмуванні та користуванні найновішими технологіями зможуть швидко та легко згенерувати текст за ключовим словом, виконуючи прості дії, які будуть вказані на екрані.

Недоліки, які в майбутньому будуть виправлятися з впровадженням нових технологій:

- Не здатний згенерувати довгі, складні або логічно зв'язані тексти.
- Може бути схильний до генерації нелогічних та неправдоподібних фраз або повторів.
- Обмежена функціональність.
- Може працювати лише з однією мовою.

2.2 Вибір середовища та інструментів розробки

Мова програмування є необхідним інструментом у написанні будь-якої програми. Аналогічно до людського спілкування – не знаючи мови складно пояснити щось так щоб тебе зрозуміли. Так само і в програмуванні – комп'ютер не зрозуміє яку задачу треба виконати і як, якщо написати код незрозумілою йому мовою, наприклад українською.

Мова програмування – це формальний мовний засіб, який використовується для написання програмного коду, який згодом буде виконуватись комп'ютером або іншою обчислювальною машиною.

Суворіше визначення: мова програмування — це система позначень для опису алгоритмів і структур даних, певна штучна формальна система, засобами якої можна виражати алгоритми. Мову програмування визначає набір лексичних, синтаксичних і семантичних правил, що задають зовнішній вигляд програми та дії, які виконує виконавець (комп'ютер) під її управлінням [1].

Історія мов програмування починається з 19 століття, коли було розроблено перші «машини на картах», які використовували спеціальні отвори в перфокартах для виконання обчислювальних операцій. У 20-му столітті з'явилися перші електронні комп'ютери, для яких були розроблені високорівневі і низькорівневі мови програмування [1].

Перша високорівнева мова програмування – FORTRAN, була розроблена 1950-х роках для наукових обчислень. Пізніше з'явилися інші мови високого рівня, такі як COBOL, BASIC та C, які використовуються і до сьогодні [1].

У 1980-х роках були розроблені об'єктно-орієнтовані мови програмування, такі як C++, які дозволяють програмістам створювати складні програми з більшим рівнем абстракції та повторного використання коду.

З появою Інтернету та розширенням індустрії програмного забезпечення з'явилися нові мови програмування, такі як JavaScript, Python та Ruby. Ці мови програмування використовуються для веб-розробки, машинного навчання, аналізу даних та багатьох інших задач.

Сьогодні існує безліч мов програмування, кожна з яких має свої особливості та призначення. Вибір мови програмування залежить від задачі, яку потрібно вирішити, та від особистих уподобань програміста.

Саме для реалізації системи генерації тексту за ключовим словом необхідні спеціальні інструменти, що дозволяють створювати та редагувати словник N-грамм. Один з найбільш ефективних інструментів для створення та редагування текстових даних є мова програмування Python, зокрема бібліотека NLTK, яка надає можливості для обробки природних мов та створення N-грамм.

Але для розробки такої системи було обрано середовище програмування Visual Studio та мову програмування C#. Visual Studio було обрано з огляду на

те, що це середовище програмування дозволяє швидко створювати різноманітні застосунки та інтерфейси, має велику кількість вбудованих інструментів для автоматизації рутинних задач та відлагодження коду. Мова програмування C# була обрана з огляду на її об'єктно-орієнтовану структуру та високу продуктивність.

Об'єктно-орієнтоване програмування (ООП) – одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою.

ООП дозволяє створювати більш складні програми з більшими обсягами коду, які можуть бути більш легко підтримувані, змінювані та розширювані. Програмування в ООП також дозволяє зменшити кількість дублюючого коду, що полегшує розробку та зберігання програмного забезпечення.

Основні принципи ООП:

- Інкапсуляція - властивість об'єктів, що дозволяє об'єднувати дані та методи, які працюють з цими даними, в єдиний об'єкт. Це дозволяє забезпечити приховування деталей реалізації та підвищити безпеку програми [9].
- Наслідування - процес, коли один клас наслідує властивості та методи іншого класу. Це дозволяє зменшити кількість коду та підвищити його читабельність [9].
- Поліморфізм - властивість об'єктів, що дозволяє їм виконувати різні функції, залежно від контексту виклику. Це дозволяє забезпечити більш гнучкий та складний функціонал [9].
- Абстракція - процес визначення загальних характеристик об'єктів та процесів, відокремлення їх від деталей реалізації. Це дозволяє забезпечити більш зрозумілу та легко змінювану архітектуру програмного продукту [9].

Використання принципів ООП дозволяє забезпечити більш гнучку та масштабовану архітектуру програмного продукту, що дозволяє легко додавати новий функціонал та збільшувати його ефективність та надійність.

Також додатково було використано такі інструменти:

- .NET Framework - це платформа для розробки та виконання програмного забезпечення, яка надає зручний інтерфейс для програмування мовами, такими як C#.
- LINQ (англ. Language Integrated Query - запити, інтегровані в мову) – компонент Microsoft .NET Framework, який додає нативні можливості виконання запитів даних до мов, що входять у .NET [2].
- Git - це розподілена система контролю версій, яка дозволяє зберігати та відстежувати зміни в коді проекту. Git надає можливість повернутися до будь-якої попередньої версії коду, яка була збережена, що дозволяє відновити роботу в разі помилки чи неправильної зміни коду.

Якщо програма розробляється декількома розробниками або навіть компанією, то Git дозволяє розробникам працювати з одним і тим же кодом, зберігаючи кожну зміну як версію. Кожен розробник може працювати зі своєю власною копією коду та об'єднувати зміни з іншими розробниками, що робить роботу над проектом більш організованою та ефективною.

А для того, щоб мати доступ до свого проєкту в будь-якому місці та з будь-якого комп'ютера, або передати код на розгляд або правки іншим розробникам, використовувався хмарний сервіс GitHub для зберігання та управління проєктами програмного забезпечення. Для зручного керування версіями та виконання комітів використовувався графічний клієнт GitHub Desktop.

Висновки до Розділу 2

Проведено порівняльний аналіз продуктів з генерації тексту за ключовим словом, таких як Chat GPT, TextGenrnn, IBM Watson та Markovify, на конкурентному ринку товарів. Підсумовуючи, можна сказати, що різні генератори тексту мають свої переваги та недоліки, і вибір продукту користувачем залежить від конкретної задачі та потреб.

Наш генератор тексту буде мати декілька переваг порівняно з конкурентами, такі як: відмінна швидкість генерації тексту, легкість у використанні та зрозумілість алгоритму. Однак, у порівнянні з іншими генераторами тексту, може бути менш точним та менш ефективним у генерації більш складних текстів, які мають велику кількість різноманітної інформації.

Вибір Visual Studio та мови програмування C# дозволить швидко та ефективно створити систему генерації тексту за ключовим словом з високою продуктивністю та широкими можливостями для налагодження та тестування. А користування .NET Framework, LINQ та Git сприяє покращенню продуктивності та якості розробки проєкту.

РОЗДІЛ 3

РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ «Generatext»

3.1. Технології, використані для розробки продукту

Даний програмний додаток, як вже було сказано, розроблено мовою програмування C# та побудовано із застосуванням шаблону «Console App (.NET Framework)». Шаблони надають користувачам деякий базовий код та структуру, які спрощують початок написання власного коду.

Використовуючи один з принципів об'єктно-орієнтованого програмування – інкапсуляцію, написання коду було поділено на 5 класів, кожен з яких ще на декілька методів. Це дозволило приховати реалізацію деталей внутрішнього функціонування об'єкту від користувача, зберігаючи при цьому його стан та можливість взаємодії з ним через визначені методи.

- **BookSelector.cs**

Даний клас містить методи, що потрібні під час вибору користувачем книги та введення ним даних, необхідних для генерації тексту. Всі методи оголошені з модифікатором `public`, бо використовуються в інших класах та викликають інші методи всередині своєї реалізації.

Метод **`PrintListOfBooks(string[] files)`** друкує в консоль список книг, а якщо точніше – список файлів з вказаної директорії. Користувач з цього списку має вибрати номер файлу (книги, завдяки якій буде відбуватися генерація) або цифру '0' – файл користувача (рис. 3.4).

```
public static void PrintListOfBooks(string[] files)
{
    Console.WriteLine("List of books:");
    for (int i = 0; i < files.Length; i++)
    {
        Console.WriteLine($"{i + 1}. {Path.GetFileName(files[i])}");
    }
    Console.WriteLine("\t0. {Your book}");
}
```

Рис. 3.4 Виведення списку файлів у консоль

Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)

Після чого метод **BookChoice(string[] files)** вже безпосередньо отримує від користувача номер обраного файлу і передає текст з нього в змінну *text* (рис. 3.5).

```
public static string BookChoice(string[] files)
{
    string input;
    int userChoice;
    string text = "";
    int attempts = 0;
    bool isValid = false;
    while (!isValid)
    {
        Console.WriteLine("\nChoose the book (enter the number): ");
        input = Console.ReadLine();
        if (int.TryParse(input, out userChoice) && userChoice <= files.Length)
        {
            for (int i = 0; i < files.Length; i++)
            {
                if (i + 1 == userChoice)
                {
                    text = File.ReadAllText(files[i]);
                }
                else if (userChoice == 0)
                {
                    string userFilePath;
                    do
                    {
                        Console.WriteLine("\n" + @"Enter the path to your book (C:\\example\\file.txt):");
                        userFilePath = Console.ReadLine();
                        if (IsValidFilePath(userFilePath))
                        {
                            text = File.ReadAllText(userFilePath);
                        }
                        else
                        {
                            InvalidInput(attempts);
                            attempts++;
                        }
                    }
                    while (!IsValidFilePath(userFilePath));
                }
            }
            isValid = true;
        }
        else
        {
            InvalidInput(attempts);
            attempts++;
        }
    }
    return text;
}
```

Рис. 3.5 Обробка вибору файлу користувачем

Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)

Далі методи **GetUserBeginningWord()** (рис. 3.6) та **GetUserWordsCount()** (рис. 3.7) обробляють введені користувачем значення ключового (першого) слова та кількості слів у реченні відповідно, після чого вертають їх.

```

public static string GetUserBeginningWord()
{
    var beginning = "";
    int attempts = 0;
    bool isValid = false;
    while (!isValid)
    {
        Console.WriteLine("\nEnter the first word (for example, start): ");
        beginning = Console.ReadLine();
        if (SentencesParser.IsWord(beginning))
        {
            isValid = true;
        }
        else
        {
            InvalidInput(attempts);
            attempts++;
        }
    }
    return beginning;
}

```

Рис. 3.6 Обробка введенного ключового слова

Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)

```

public static int GetUserWordsCount()
{
    int countWords = 0;
    int attempts = 0;
    bool isValid = false;
    while (!isValid)
    {
        Console.WriteLine("\nEnter the count of words in sentence: ");
        string inputCount = Console.ReadLine();
        if (int.TryParse(inputCount, out countWords))
        {
            isValid = true;
        }
        else
        {
            InvalidInput(attempts++);
        }
    }
    return countWords;
}

```

Рис. 3.7 Обробка введенного значення кількості слів

Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)

Щоб уникнути помилок виконання програми, якщо користувач введе недопустиме значення (наприклад літеру замість цифри), в кожному методі, що приймає значення від користувача, виконуються перевірки.

Наприклад для того, щоб перевірити чи правильно було введено шлях до користувацького файлу або чи існує він взагалі, використовується метод **IsValidFilePath(string path)** (рис. 3.8).

```
public static bool IsValidFilePath(string path)
{
    return File.Exists(path);
}
```

Рис. 3.8 Перевірка існування файлу за введеним шляхом

Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)

Для того, щоб повідомляти користувача, що той ввів недопустиме значення, виконується метод **InvalidInput(int attempts)** (рис. 3.9).

```
public static void InvalidInput(int attempts)
{
    if (attempts < 2)
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Invalid input. Try again!");
        Console.ResetColor();
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Too many attempts.");
        Console.ResetColor();
        RetryOrExit();
    }
}
```

Рис. 3.9 Виведення в консоль повідомлень про помилку

Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)

Після третьої спроби (лічильником є змінна *attempts*) програма пропонує користувачу вибір: продовжити чи завершити виконання програми (за допомогою метода **RetryOrExit()**) (рис. 3.10).

```

public static void RetryOrExit()
{
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("\nPress any key to try again or '0' to exit");
    Console.ResetColor();
    string choice = Console.ReadLine();
    if (choice == "0")
    {
        Environment.Exit(0);
    }
}

```

Рис. 3.10 Вибір щодо продовження виконання програми

Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)

- **SentencesParser.cs**

Цей клас виконує розбиття тексту, на основі якого буде відбуватися генерація, на речення та слова. Для того, щоб розділити текст на речення, був створений метод **ParseSentences(string text)**, який приймає текст типу *string* з файлу та вертає *List<List<string>>()* (рис. 3.11). В ньому використовується вбудований метод **string.Split** та створений власноруч метод **IsWord(string word)** типу *bool* (рис. 3.12), який перевіряє, чи є символи в реченні буквами. Після перевірки символів використовується рядковий клас **StringBuilder**, визначений у просторі імен *System.Text*.

```

public static List<List<string>> ParseSentences(string text)
{
    var sentencesList = new List<List<string>>();
    char[] separators = { '.', '!', '?', '(', ')', ':', ';' };
    var splitText = text.Split(separators, System.StringSplitOptions.RemoveEmptyEntries);
    foreach (var sentence in splitText)
    {
        var builder = new StringBuilder();
        foreach (char symbol in sentence)
        {
            if (char.IsLetter(symbol) || symbol == '\\') builder.Append(symbol);
            else builder.Append(" ");
        }
        var splitSentence = builder.ToString().Split(' ', (char)StringSplitOptions.RemoveEmptyEntries);
        var wordsInOneSentence = new List<string>();
        foreach (var word in splitSentence)
        {
            if (!string.IsNullOrEmpty(word)) wordsInOneSentence.Add(word.ToLower());
        }
        sentencesList.Add(wordsInOneSentence);
    }
    sentencesList.RemoveAll(x => x == null || x.Count == 0);
    return sentencesList;
}

```

Рис. 3.11 Виконання розбиття тексту

Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)

```

public static bool IsWord(string word)
{
    bool isLetter = true;
    foreach (var symbol in word)
    {
        if (char.IsLetter(symbol) || symbol == '\\') isLetter = isLetter && true;
        else isLetter = false;
    }
    return isLetter;
}

```

Рис. 3.12 Перевірка чи є частина розбитого тексту словом

Джерело: розроблено автором у додатку *Visual Studio* (знімок з екрану)

- **FrequencyAnalysis.cs**

В даному класі створюється словник найчастіших продовжень біграм (рис. 3.13) та триграм (рис. 3.14).

Наприклад, з тексту: «*She stood up. Then she left.*» можна виділити такі біграми «*she stood*», «*stood up*», «*then she*» та «*she left*», але не «*up then*». І дві триграми «*she stood up*» та «*then she left*», але не «*stood up then*».

Отже, результатом використання цього класу є словник ***result***, ключами якого є всі можливі початки біграм і триграм, а значеннями їх найчастіші продовження (рис. 3.15). Якщо є кілька продовжень з однаковою частотою, використовуються ті, що лексикографічно менше. Для лексикографічного порівняння був використаний вбудований у .NET спосіб порівняння ***string.CompareOrdinal***.

```

public static Dictionary<string, Dictionary<string, int>> FillBigrams(List<List<string>> text)
{
    var bigrams = new Dictionary<string, Dictionary<string, int>>();
    foreach (var sentence in text)
    {
        for (int i = 0; i < sentence.Count - 1; i++)
        {
            if (!bigrams.ContainsKey(sentence[i]))
            {
                bigrams.Add(sentence[i], new Dictionary<string, int>());
            }
            if (!bigrams[sentence[i]].ContainsKey(sentence[i + 1]))
            {
                bigrams[sentence[i]].Add(sentence[i + 1], 1);
            }
            else
            {
                bigrams[sentence[i]][sentence[i + 1]] += 1;
            }
        }
    }
    return bigrams;
}

```

Рис. 3.13 Створення словнику біграм

Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)

```

public static Dictionary<string, Dictionary<string, int>> FillTrigrams(List<List<string>> text)
{
    var trigrams = new Dictionary<string, Dictionary<string, int>>();
    foreach (var sentence in text)
    {
        for (int i = 0; i < sentence.Count - 2; i++)
        {
            if (!trigrams.ContainsKey(sentence[i] + " " + sentence[i + 1]))
            {
                trigrams.Add(sentence[i] + " " + sentence[i + 1], new Dictionary<string, int>());
            }
            if (!trigrams[sentence[i] + " " + sentence[i + 1]].ContainsKey(sentence[i + 2]))
            {
                trigrams[sentence[i] + " " + sentence[i + 1]].Add(sentence[i + 2], 1);
            }
            else
            {
                trigrams[sentence[i] + " " + sentence[i + 1]][sentence[i + 2]] += 1;
            }
        }
    }
    return trigrams;
}

```

Рис. 3.14 Створення словнику триграм

Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)

```

public static Dictionary<string, string> GetMostFrequentNextWords(List<List<string>> text)
{
    var biAndTriGrams = new Dictionary<string, Dictionary<string, int>>();
    biAndTriGrams = FillBigrams(text).Concat(FillTrigrams(text)).ToDictionary(x => x.Key, x => x.Value);

    var result = new Dictionary<string, string>();

    foreach (var pair in biAndTriGrams)
    {
        var maxRepeatValue = 0;
        string mostFrequentContinue = null;
        foreach (var continuation in pair.Value)
        {
            if (continuation.Value == maxRepeatValue)
            {
                if (string.CompareOrdinal(mostFrequentContinue, continuation.Key) > 0)
                {
                    mostFrequentContinue = continuation.Key;
                }
            }
            else if (continuation.Value > maxRepeatValue)
            {
                mostFrequentContinue = continuation.Key;
                maxRepeatValue = continuation.Value;
            }
            if (!result.ContainsKey(pair.Key)) result.Add(pair.Key, mostFrequentContinue);
            else result[pair.Key] = mostFrequentContinue;
        }
    }
    return result;
}

```

Рис. 3.15 Отримання найбільш частого продовження

Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)

- **TextGenerator.cs**

На вхід алгоритму передається словник **NextWords**, отриманий у класі **FrequencyAnalysis.cs**, одне або кілька перших слів фрази **phraseBeginning** і **wordsCount** — кількість слів, які потрібно дописати до початкової фрази.

Інакше, якщо у словнику є ключ, що складається з одного останнього слова фрази, то продовжувати потрібно словом, що зберігається у словнику цього ключа.

Інакше потрібно достроково закінчити генерування фрази і повернути згенерований на даний момент результат.

Метод **GetNextWord** розділяє рядок **total** на слова, використовуючи пробіл як роздільник, і відбирає останнє слово, яке є початком наступного слова. Якщо у **total** міститься більше одного слова, метод відбирає останні два слова, які є початком наступного слова.

Далі метод перевіряє, чи містить словник `nextWords` наступні слова після останнього слова `total`. Якщо так, метод додає наступне слово в рядок `total` і повертає його. Якщо словник не містить наступних слів після останнього слова `total`, метод перевіряє, чи є наступне слово, що може йти після передостаннього слова. Якщо так, метод додає наступне слово в рядок `total` і повертає його. Якщо і це не можливо, метод повертає рядок `total` без змін.

```
private static string GetNextWord(string total, Dictionary<string, string> nextWords)
{
    var start = total.Split(' ').Skip(total.Split(' ').Count() - 2);
    var startBigram = start.LastOrDefault();
    var startTrigram = start.Count() > 1 ? start.FirstOrDefault() + " " + startBigram : "";

    return nextWords.ContainsKey(startTrigram) ? total + " " + nextWords[startTrigram] :
           nextWords.ContainsKey(startBigram) ? total + " " + nextWords[startBigram] : total;
}
```

Рис. 3.16 Отримання наступного слова

Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)

```
public static string ContinuePhrase(Dictionary<string, string> nextWords, string phraseBeginning, int wordsCount) =>
    Enumerable.Repeat("", wordsCount)
        .Aggregate(string
            .Join(" ", phraseBeginning
                .Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)),
            (total, next) => GetNextWord(total, nextWords));
```

Рис. 3.17 Продовження фрази отриманим наступним словом

Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)

- **Program.cs**

Цей клас містить метод **Main** (рис. 3.18), що є точкою входу до програми. В ньому виконуються початкові налаштування та викликаються інші методи, які виконують основну роботу додатку.

```

internal class Program
{
    static void Main(string[] args)
    {
        string folderPath = Path.Combine(Directory.GetCurrentDirectory(), "..", "..", "Books");
        string[] files = Directory.GetFiles(folderPath);
        BookSelector.PrintListOfBooks(files);
        var sentences = SentencesParser.ParseSentences(BookSelector.BookChoice(files));
        var frequency = FrequencyAnalysis.GetMostFrequentNextWords(sentences);

        while (true)
        {
            string beginning = BookSelector.GetUserBeginningWord();
            int countWords = BookSelector.GetUserWordsCount();
            if (string.IsNullOrEmpty(beginning)) return;
            var phrase = TextGenerator.ContinuePhrase(frequency, beginning.ToLower(), countWords);
            Console.WriteLine($"Your sentence: \n{phrase}.");
            BookSelector.RetryOrExit();
        }
    }
}

```

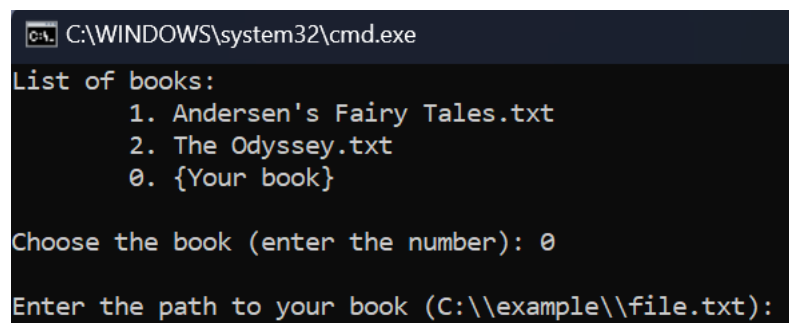
Рис. 3.18 Виклик методів в основній програмі, генерація тексту

Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)

3.2. Інтерфейс користувача

Інтерфейс користувача реалізований в консолі за допомогою текстових повідомлень, які виводяться на екран, та команд, які користувач може вводити з клавіатури.

На початку роботи консольного додатка виводиться нумерований список книг, серед яких користувач обирає будь-який номер або цифру '0', щоб ввести свою (рис. 3.19).



```

C:\WINDOWS\system32\cmd.exe
List of books:
    1. Andersen's Fairy Tales.txt
    2. The Odyssey.txt
    0. {Your book}

Choose the book (enter the number): 0

Enter the path to your book (C:\\example\\file.txt):

```

Рис. 3.19 Початок роботи програми в інтерпретаторі командного рядка

Джерело: розроблено автором у додатку std.exe (знімок з екрану)

Далі користувач вводить початкове (ключове) слово та кількість слів в реченні. Після виведення результату користувач може вийти з додатку натиснувши '0' або будь-яку клавішу, якщо хоче повторити генерацію (рис. 3.20).

```
Enter the first word (for example, start): start
Enter the count of words in sentence: 9
Your sentence:
start at our opposite neighbor's house and the little boy.
Press any key to try again or '0' to exit
```

Рис. 3.20 Отримання результату

Джерело: розроблено автором у додатку std.exe (знімок з екрану)

Якщо користувач вводить неправильне значення, то в консолі виводиться червоним кольором повідомлення «Invalid input. Try again!».

```
Enter the first word (for example, start): 1\-=
Invalid input. Try again!
```

Рис. 3.21 Виведення повідомлення про помилку

Джерело: розроблено автором у додатку std.exe (знімок з екрану)

Висновок до Розділу 3

Програмний продукт було розроблено мовою програмування C# та побудовано із застосуванням шаблону «Console App (.NET Framework)». Для роботи програми було створено 5 класів: BookSelector.cs (вибір книги користувачем), SentencesParser.cs (розділення тексту на слова), FrequencyAnalysis.cs (отримання найбільш частого продовження слова), TextGenerator.cs (генерація тексту) та Program.cs (виклик методів, виконання основної роботи додатку). Результатом роботи програми є виведення даних комп'ютером і введення даних користувачем в інтерпретаторі командного рядка.

ВИСНОВКИ

У результаті виконання роботи було досліджено алгоритми та моделі, що дозволяють автоматично генерувати текст на основі заданого ключового слова, та створено програмне забезпечення «Generatext», яке виконує власне генерацію.

Програмний продукт дозволяє користувачу швидко генерувати текстові дані на різноманітні теми, що значно підвищує продуктивність та ефективність роботи в різних сферах діяльності.

Головною функцією додатку «Generatext» є отримання від користувача інформації щодо файлу, що буде використаний для генерації, та ключового слова, і виведення комп'ютером згенерованого речення.

Мета і завдання курсової роботи були виконані.

Під час виконання роботи було використано мову програмування C# через її об'єктно-орієнтованість та високу продуктивність з використанням платформи .NET Framework. Вибір середовища розробки впав на Microsoft Visual Studio через наявність широкого функціоналу та зручність використання.

Серед моделей для генерації тексту було обрано одну з найпопулярніших: використання n-грам. Вони дозволяють моделювати залежності між словами в тексті та генерувати достатньо природні речення.

Додаток «Generatext» був розроблений з метою автоматизувати генерацію тексту в галузі обробки природної мови. Використання «Generatext» може зекономити час та зусилля, які зазвичай витрачаються на написання тексту вручну. Додаток також може забезпечити високу якість тексту, що генерується, оскільки він базується на передових алгоритмах генерації тексту.

Загалом, «Generatext» може бути корисним інструментом для будь-кого, хто працює з генерацією тексту в галузі маркетингу, журналістики, копірайтингу, веб-розробки тощо.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gabbrielli, Maurizio. Programming languages principles and paradigms. London, New York: Springer, 2010.
2. Microsoft. Language-Integrated Query (LINQ) (документація). Microsoft Developer Network. 2021. URL: <https://docs.microsoft.com/uk-ua/dotnet/csharp/linq/>
3. Mellish, C.. The History of Natural Language Generation: An Introduction. In Natural Language Generation in Interactive Systems. Springer, Dordrecht. 2005.
4. SHRDLU: A Robot-Based Language-Game for a Child's World, Terry Winograd (1972)
5. Hausser, H., Martin, J., & Pao, Y. H. A probabilistic approach to natural language generation. Artificial Intelligence. 1984.
6. Manning, C. D., & Schütze, H. Foundations of Statistical Natural Language Processing. Cambridge, MA: MIT Press. 1999.
7. Jurafsky, D., & Martin, J. H. Speech and Language Processing. Pearson. 2019.
8. Taylor, P. How to Build a Speech Recognition Application: A Guide to the Best Tools and Software. 2019. URL: <https://builtin.com/artificial-intelligence/speech-recognition>
9. Meyer, B. Object-oriented software construction (2nd ed.). Prentice Hall. 1997.

ДОДАТКИ

Програмний код класу Program.cs

```
using System;
using System.IO;

namespace Generatext
{
    internal class Program
    {
        static void Main(string[] args)
        {
            string folderPath = Path.Combine(Directory.GetCurrentDirectory(), "..",
            "..", "Books");
            string[] files = Directory.GetFiles(folderPath);
            BookSelector.PrintListOfBooks(files);
            var sentences =
            SentencesParser.ParseSentences(BookSelector.BookChoice(files));
            var frequency = FrequencyAnalysis.GetMostFrequentNextWords(sentences);

            while (true)
            {
                string beginning = BookSelector.GetUserBeginningWord();
                int countWords = BookSelector.GetUserWordsCount();
                if (string.IsNullOrEmpty(beginning)) return;
                var phrase = TextGenerator.ContinuePhrase(frequency,
                beginning.ToLower(), countWords);
                Console.WriteLine($"Your sentence: \n{phrase}.");
                BookSelector.RetryOrExit();
            }
        }
    }
}
```

Програмний код класу BookSelector.cs

```
using System;
using System.IO;

namespace Generatext
{
    static class BookSelector
    {
        public static void PrintListOfBooks(string[] files)
        {
            Console.WriteLine("List of books:");
            for (int i = 0; i < files.Length; i++)
            {
                Console.WriteLine($"{i + 1}. {Path.GetFileName(files[i])}");
            }
            Console.WriteLine("\t0. Your book");
        }

        public static string BookChoice(string[] files)
        {
            string input;
            int userChoice;
            string text = "";
            int attempts = 0;
            bool isValid = false;
            while (!isValid)
            {
                Console.WriteLine("Choose the book (enter the number): ");
            }
        }
    }
}
```

```

input = Console.ReadLine();
if (int.TryParse(input, out userChoice) && userChoice <= files.Length)
{
    for (int i = 0; i < files.Length; i++)
    {
        if (i + 1 == userChoice)
        {
            text = File.ReadAllText(files[i]);
        }
        else if (userChoice == 0)
        {
            string userFilePath;
            do
            {
                Console.WriteLine("\n" + @"Enter the path to your book
(C:\\example\\file.txt):");
                userFilePath = Console.ReadLine();
                if (IsValidFilePath(userFilePath))
                {
                    text = File.ReadAllText(userFilePath);
                }
                else
                {
                    InvalidInput(attempts);
                    attempts++;
                }
            } while (!IsValidFilePath(userFilePath));
        }
        isValid = true;
    }
    else
    {
        InvalidInput(attempts);
        attempts++;
    }
}
return text;
}
public static void InvalidInput(int attempts)
{
    if (attempts < 2)
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Invalid input. Try again!");
        Console.ResetColor();
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine("Too many attempts.");
        Console.ResetColor();
        RetryOrExit();
    }
}
public static bool IsValidFilePath(string path)
{
    return File.Exists(path);
}
public static void RetryOrExit()
{
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("\nPress any key to try again or '0' to exit");
}

```

```

        Console.ResetColor();
        string choice = Console.ReadLine();
        if (choice == "0")
        {
            Environment.Exit(0);
        }
    }
    public static string GetUserBeginningWord()
    {
        var beginning = "";
        int attempts = 0;
        bool isValid = false;
        while (!isValid)
        {
            Console.WriteLine("\nEnter the first word (for example, start): ");
            beginning = Console.ReadLine();
            if (SentencesParser.IsWord(beginning))
            {
                isValid = true;
            }
            else
            {
                InvalidInput(attempts);
                attempts++;
            }
        }
        return beginning;
    }
    public static int GetUserWordsCount()
    {
        int countWords = 0;
        int attempts = 0;
        bool isValid = false;
        while (!isValid)
        {
            Console.WriteLine("\nEnter the count of words in sentence: ");
            string inputCount = Console.ReadLine();
            if (int.TryParse(inputCount, out countWords))
            {
                isValid = true;
            }
            else
            {
                InvalidInput(attempts++);
            }
        }
        return countWords;
    }
}
}
}

```

Програмний код класу SentencesParser.cs

```

using System;
using System.Collections.Generic;
using System.Text;

namespace Generatext
{
    static class SentencesParser
    {
        public static bool IsWord(string word)
        {
            bool isLetter = true;
            foreach (var symbol in word)

```

```

        {
            if (char.IsLetter(symbol) || symbol == '\\') isLetter = isLetter &&
true;
            else isLetter = false;
        }
        return isLetter;
    }
    public static List<List<string>> ParseSentences(string text)
    {
        var sentencesList = new List<List<string>>();
        char[] separators = { '.', '!', '?', '(', ')', ':', ';' };
        var splitText = text.Split(separators,
System.StringSplitOptions.RemoveEmptyEntries);
        foreach (var sentence in splitText)
        {
            var builder = new StringBuilder();
            foreach (char symbol in sentence)
            {
                if (char.IsLetter(symbol) || symbol == '\\')
builder.Append(symbol);
                else builder.Append(" ");
            }
            var splitSentence = builder.ToString().Split(' ',
(char)StringSplitOptions.RemoveEmptyEntries);
            var wordsInOneSentence = new List<string>();
            foreach (var word in splitSentence)
            {
                if (!string.IsNullOrEmpty(word))
wordsInOneSentence.Add(word.ToLower());
            }
            sentencesList.Add(wordsInOneSentence);
        }
        sentencesList.RemoveAll(x => x == null || x.Count == 0);
        return sentencesList;
    }
}
}
}

```

Программний код класу FrequencyAnalysis.cs

```

using System.Collections.Generic;
using System.Linq;

namespace Generatext
{
    static class FrequencyAnalysis
    {
        public static Dictionary<string, Dictionary<string, int>>
FillBigrams(List<List<string>> text)
        {
            var bigrams = new Dictionary<string, Dictionary<string, int>>();
            foreach (var sentence in text)
            {
                for (int i = 0; i < sentence.Count - 1; i++)
                {
                    if (!bigrams.ContainsKey(sentence[i]))
                    {
                        bigrams.Add(sentence[i], new Dictionary<string, int>());
                    }
                    if (!bigrams[sentence[i]].ContainsKey(sentence[i + 1]))
                    {
                        bigrams[sentence[i]].Add(sentence[i + 1], 1);
                    }
                    else
                    {

```

```

        bigrams[sentence[i]][sentence[i + 1]] += 1;
    }
}
}
return bigrams;
}
public static Dictionary<string, Dictionary<string, int>>
FillTrigrams(List<List<string>> text)
{
    var trigrams = new Dictionary<string, Dictionary<string, int>>();
    foreach (var sentence in text)
    {
        for (int i = 0; i < sentence.Count - 2; i++)
        {
            if (!trigrams.ContainsKey(sentence[i] + " " + sentence[i + 1]))
            {
                trigrams.Add(sentence[i] + " " + sentence[i + 1], new
Dictionary<string, int>());
            }
            if (!trigrams[sentence[i] + " " + sentence[i +
1]].ContainsKey(sentence[i + 2]))
            {
                trigrams[sentence[i] + " " + sentence[i + 1]].Add(sentence[i +
2], 1);
            }
            else
            {
                trigrams[sentence[i] + " " + sentence[i + 1]][sentence[i + 2]]
+= 1;
            }
        }
    }
    return trigrams;
}

public static Dictionary<string, string>
GetMostFrequentNextWords(List<List<string>> text)
{
    var biAndTriGrams = new Dictionary<string, Dictionary<string, int>>();
    biAndTriGrams =
FillBigrams(text).Concat(FillTrigrams(text)).ToDictionary(x => x.Key, x => x.Value);

    var result = new Dictionary<string, string>();

    foreach (var pair in biAndTriGrams)
    {
        var maxRepeatValue = 0;
        string mostFrequentContinue = null;
        foreach (var continuation in pair.Value)
        {
            if (continuation.Value == maxRepeatValue)
            {
                if (string.CompareOrdinal(mostFrequentContinue,
continuation.Key) > 0)
                {
                    mostFrequentContinue = continuation.Key;
                }
            }
            else if (continuation.Value > maxRepeatValue)
            {
                mostFrequentContinue = continuation.Key;
                maxRepeatValue = continuation.Value;
            }
        }
    }
}

```

```

        if (!result.ContainsKey(pair.Key)) result.Add(pair.Key,
mostFrequentContinue);
        else result[pair.Key] = mostFrequentContinue;
    }
}
return result;
}
}
}

```

Програмний код класу TextGenerator.cs

```

using System.Collections.Generic;
using System;
using System.Linq;

namespace Generatext
{
    static class TextGenerator
    {
        private static string GetNextWord(string total, Dictionary<string, string>
nextWords)
        {
            var start = total.Split(' ').Skip(total.Split(' ').Count() - 2);
            var startBigram = start.LastOrDefault();
            var startTrigram = start.Count() > 1 ? start.FirstOrDefault() + " " +
startBigram : "";

            return nextWords.ContainsKey(startTrigram) ? total + " " +
nextWords[startTrigram] :
                nextWords.ContainsKey(startBigram) ? total + " " +
nextWords[startBigram] : total;
        }

        public static string ContinuePhrase(Dictionary<string, string> nextWords,
string phraseBeginning, int wordsCount) =>
            Enumerable.Repeat("", wordsCount)
                .Aggregate(string
                    .Join(" ", phraseBeginning
                        .Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)),
                    (total, next) => GetNextWord(total, nextWords));
    }
}

```


ДЕРЖАВНИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Рецензія на курсову роботу (проект) і результат захисту

Студента Аверіна Наталія Ігорівна

(прізвище, ім'я та по батькові)

2 курсу 4 групи ФІТ факультету

Курсова робота (проект) з «Об'єктно-орієнтованого програмування»

(назва навчальної дисципліни)

Тема «Розробка програмного забезпечення з генерації тексту за ключовим словом «Generatext» мовою програмування C#»

Реєстраційний № , дата одержання 01.05.2023 р.

Науковий керівник доцент, кандидат політичних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки Чубаєвський В.І.

(вчене звання, прізвище, ініціали)

Зміст рецензії

У курсовій роботі студентки Аверіної Н. І. на тему «Розробка програмного забезпечення з генерації тексту за ключовим словом «Generatext» мовою програмування C#» зазначено актуальність дослідження, мету, об'єкт, предмет та задачі дослідження. У теоретичній частині курсової роботи розглянуто історію розробок з генерації тексту, основні методи обробки природної мови та ефективні алгоритми генерації тексту. У другому розділі студент розглянув питання, пов'язані з конкурентоспроможністю на ринку продуктів з генерації тексту та з огляду на це обрані середовище розробки та інструменти. У третьому розділі описано хід та методи розробки додатку в середовищі Visual Studio. Автором детально описано та охарактеризовано структуру програмного застосунку, призначення кожного окремого методу та класу і зв'язки між складовими проекту. Для реалізації інтерфейсу користувача, автор розробив відображення даних у зручному вигляді для користувача в інтерпретаторі командного рядка. У цілому, курсова робота студентки Аверіної Н. І. на тему «Розробка програмного забезпечення з генерації тексту за ключовим словом «Generatext» мовою програмування C#» оформлена у відповідності до Вимог, може бути допущена до захисту та заслуговує на високу оцінку.

Допущено до захисту “01” травня 2023 р.

Захист планується о 10:00 “16” 05 2023 р.

(час)

кафедра інженерії програмного забезпечення та кібербезпеки
(місце роботи комісії)

(підпис наукового керівника)

Курсова робота захищена “16” 05 2023 р.

з оцінкою _____

(за шкалою ДТЕУ, національною шкалою та шкалою ЄКТС)

Комісія:

1. _____
(підпис)

Бешешко Б.Т.
(прізвище, ініціали)

2. _____
(підпис)

Десятко А.М.
(прізвище, ініціали)

3. _____
(підпис)

Хорольська К.В.
(прізвище, ініціали)