ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення для вилучення з комерційних документів цінної для користувача інформації за допомогою Python»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Владислав МЕЛЬНИК
(підпис)

Виконав: здобувач вищої освіти групи ПД-41
Владислав МЕЛЬНИК

Керівник: Ігор АВЕРІЧЕВ
к.е.н.

Рецензент:

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мельник Владислав Вікторович

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення для вилучення з комерційних документів цінної для користувача інформації за допомогою Python»
керівник кваліфікаційної роботи к.е.н., доцент кафедри ІПЗ Ігор АВЕРІЧЕВ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи обробки природньої мови, опис методів вилучення інформації з тексту, технічна документація з описом бібліотек для обробки природньої мови.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Проведення аналізу предметної області та існуючих передумов для розробки
2. Аналіз аналогів рішень та допоміжних модулів для виконання роботи,

3. Проведення тестування розробленого сервісу та аналіз методів роботи з отриманими результатами.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів
2. Вимоги до застосунку
3. Програмні засоби реалізації
4. Діаграма варіантів використання
5. Блок-схема
6. Діаграма класів
7. Екранні форми
8. Демонстрація застосунку
9. Апробація результатів дослідження

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Проаналізувати теоретичні дослідження в сфері вилучення цінної інформації з комерційних документів	14.03-24.03.2024	
4	Дослідити методи вилучення цінної інформації з комерційних документів та визначити найефективніший з них	25.03-28.03.2024	
5	Розробити застосунок для вилучення цінної інформації з комерційних документів	29.03-19.04.2024	
6	Тестування застосунку	20.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Владислав МЕЛЬНИК

(підпис)

Керівник
кваліфікаційної роботи _____ Ігор АВЕРІЧЕВ

(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 67 стор., 2 табл., 9 рис., 15 джерел.

Мета роботи – спрощення процесу вилучення з комерційних документів цінної для користувача інформації за допомогою застосунку мовою Python.

Об'єкт дослідження – процес вилучення з комерційних документів цінної для користувача інформації.

Предмет дослідження – програмне забезпечення для вилучення з комерційних документів цінної для користувача інформації.

Короткий зміст роботи: в роботі проаналізовано алгоритми та методи для обробки природної мови в контексті комерційних документів. Проаналізовано аналоги: Textract, Pytesseract, Apache Tika, Rossum, Tabula. Розроблено алгоритм роботи застосунку та програмно реалізовані ключові функціональні можливості, зокрема: завантаження текстових документів (DOCX), вивід цінної інформації для користувача в зручну таблицю. Проведено функціональне та модульне тестування застосунку. В роботі використано бібліотеку NLTK, SpaCy для обробки текстів, PyQt5 для створення користувацького інтерфейсу.

Сферою використання застосунку є комерційні організації, такі як, навчальні заклади, приватні організації, та інші, кому потрібно вилучати цінну комерційну інформацію.

КЛЮЧОВІ СЛОВА: PYTHON, IDE, ОБРОБКА ПРИРОДНОЇ МОВИ (NLP), СТОП-СЛОВА, СТЕМІНГ, ФЛЕКСІЯ, PYCHARM, ECLIPSE.

ЗМІСТ

ВСТУП

Комерція становить величезну частину ділового світу і складається з таких документів, як договори купівлі-продажу, замовлення на купівлю, договори купівлі-продажу, гарантійні зобов'язання, рахунки-фактури, накладні, коносаменти, митні декларації та форми замовлень, яким зацікавлені сторони надають унікальну семантику при здійсненні бізнес-процесів.

Мета роботи - спрощення процесу вилучення з комерційних документів цінної для користувача інформації за допомогою застосунку мовою Python.

Об’єкт дослідження - процес вилучення з комерційних документів цінної для користувача інформації.

Предмет дослідження - програмне забезпечення для вилучення з комерційних документів цінної для користувача інформації.

Задачі дипломної роботи:

1. Дослідити особливості вирішення задачі вилучення інформації з текстових документів комерційного спрямування.
2. Проаналізувати цифрові рішення для вилучення з комерційних документів цінної для користувача інформації.
3. Визначити функціональні та нефункціональні вимоги для вилучення з комерційних документів цінної для користувача інформації.
4. Спроектувати схему функціонування програми
5. Розробка інструментальних засобів для вирішення задачі вилучення з комерційних документів цінної для користувача інформації.
6. Провести тестування застосунку.

Автоматизація бізнес-процесів - це процес, який допомагає установам замінити ручну роботу з документами на комп'ютерну, щоб дані можна було збирати та обробляти швидше і точніше.

Сучасна автоматизація бізнес-процесів (BPA) в основному залежить від поєднання системи планування ресурсів підприємства (ERP) і системи управління

бізнес-процесами (BPM), що вимагає високого ступеня узгодженості даних між цими двома компаніями. Крім того, дані, що використовуються в BPA, є відносно статичними, що не сприяє ані виникненню нових бізнес-потреб, ані використанню масштабних високоточних сервісних можливостей.

Для більшості існуючих програмних систем вилучення комерційної інформації необхідно виконувати через величезну ручну роботу для створення нового набору правил для нового типу комерційних документів і модифікацій старого набору правил. Масштабні витрати на розробку і низька можливість повторного використання набору правил створюють величезні проблеми для комерційних програмних систем.

У дослідженні пропонується нова ідея розробки програмної системи Gasolution, яка виводить набір правил і технологію ICR.TSR з процесу розробки і відповідає лише за інтеграцію API технології ICR.TSR, забезпечуючи високу масштабованість, швидку кастомізацію, інтелектуальність і легкість дизайну.

Для того, щоб задовольнити внутрішні потреби майже всіх циклів бізнес-процесу в BPA express, можливості системи BPA повинні бути розумнішими, активнішими та ефективнішими. В даний час машинне навчання і глибоке навчання були розроблені для вилучення важливої інформації для типових документів, особливо для типових офісних документів або інструментів, таких як Word, Excel і PowerPoint, але комерційні документи не підтримуються.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Потенційні проблеми

Завданнями розробки програмного забезпечення на мові Python для вилучення даних і трендів з комерційних документів, які супроводжують проект розробки програмного забезпечення, є: збір, структурування та фільтрація даних; етап вибору методів; розробка програмного забезпечення в поєднанні з розробкою функціоналу на мові Python; тестування продуктивності; оцінка прибутковості. Програмне забезпечення вирішує наступні завдання вилучення цінної інформації. Автоматичне виявлення текстових областей в цифрових документах з наступними типами даних: дата; порядковий; посилання; назва компанії; цифри, ціле число, валюта; сума, проміжний підсумок, якість, ціна за одиницю, опис одиниці, загальна ціна, загальна кількість, ціна, сума, опис, податок; код товару. Обчислюються значення - підсумок по стовпчику, підсумок по рядку, різниця, загальна сума.

Сучасним трендом у розвитку бізнес-аналітики та систем прийняття рішень є використання аналітики великих даних. Поряд з класичними джерелами великих даних, такими як соціальні мережі, голосові записи телефонних розмов, друковані видання, з'являються нові джерела даних. Такі комерційні документи, як платіжні доручення, рахунки-фактури, гарантії, мають великий обсяг на будь-якому підприємстві і є багатими на дані. Вилучення цінної інформації, необхідної для прийняття бізнес-рішень, з комерційних документів є трудомістким завданням. Для прискорення виконання цього завдання та мінімізації необхідної робочої сили необхідно розробити адекватне програмне забезпечення з високим ступенем вилучення інформації. Ця стаття представляє наступне завдання в рамках розробки такого програмного забезпечення, яке включає в себе постановку задачі, огляд стратегій розв'язання, опис вибраних методів мовою Python та обговорення отриманих результатів.

1.2 Пошук та його різновиди

Пошук - це процес знаходження інформації з різних джерел. Його види можна класифікувати за різними критеріями.

Текстовий пошук має дві основні форми: простий і контекстуальний. Простий текстовий пошук полягає в тому, що користувач вводить ключові слова, а система знаходить документи, де ці слова зустрічаються. У контекстуальному пошуку враховується контекст запиту, використовуючи синоніми або аналізуючи попередні запити користувача.

Пошук зображень також може бути здійснений за допомогою ключових слів або шляхом завантаження зображення, щоб система знаходила схожі. Те ж саме стосується і відео пошуку - його можна здійснити за описом або завантаживши кадр з відео.

Типи пошуку можуть бути розділені за способом обробки. Статичний пошук надає результати, які не змінюються без оновлення даних, тоді як динамічний пошук дозволяє результатам змінюватися в залежності від нових даних та інших факторів.

Локальний пошук відбувається на певному пристрої або в локальній мережі, в той час як глобальний пошук охоплює пошук в Інтернеті або в глобальних базах даних. Горизонтальний пошук охоплює широкий спектр тем і джерел і використовує алгоритми для індексації і знаходження інформації.

Гуманітарний пошук здійснюється людиною, часто великих базах даних або архівах, і частково полягає в пошуку зв'язків між різними елементами інформації.

Останнім аспектом є використання пошуку для різних цілей, включаючи SEO для підвищення видимості сайтів, пошуковий маркетинг для залучення трафіку, і пошукові системи, такі як Google та Bing, які використовують різні методи для надання релевантних результатів користувачам.

1.3 Особливості вилучення інформації з текстів

Виділення іменованих сутностей (NER):

-розпізнавання і класифікація імен, організацій, місць, дат, та інших категорій.Наприклад, у реченні "Google має штаб-квартиру в Маунтін-В'ю", "Google" буде розпізнано як організація, а "Маунтін-В'ю" як місце.

Виділення відносин:

-виявлення зв’язків між сутностями, наприклад, хто є генеральним директором компанії або які країни підписали договір.

Виділення подій:

-виявлення дій або інцидентів, що включають кілька сутностей і відносин.Наприклад, у реченні "Apple випустила новий iPhone у вересні", подія "випустила" пов'язує "Apple" з "новим iPhone" і "вереснем".

Методи вилучення інформації:

-використання заздалегідь визначених правил та регулярних виразів для знаходження сутностей та відносин у тексті. Вимагають значних зусиль для створення та підтримки, менш гнучкі. Наглянуте навчання використовує анотовані дані для навчання моделей розпізнавання сутностей та відносин.Не наглянуте навчання використовує методи кластеризації для виявлення сутностей без попередньої анотації.

Глибоке навчання:

-нейронні мережі, такі як LSTM, GRU, і трансформери (наприклад, BERT), ефективні для NER та виділення відносин завдяки здатності обробляти великі обсяги даних і враховувати контекст.

-слова можуть мати декілька значень у різних контекстах, що ускладнює точне вилучення інформації.

-недостатня кількість анотованих даних для навчання моделей може знижувати їх точність і ефективність.

-природні мови мають складні граматичні структури і нюанси, що створює додаткові труднощі для систем IE.

-автоматизація аналізу документів та отримання ключових показників для прийняття бізнес-рішень.

-виділення інформації з наукових статей для виявлення нових знань про хвороби та ліки.

Інформаційна безпека:

-поліпшення релевантності пошукових результатів завдяки кращому розумінню запитів користувачів.

-вилучення інформації є важливим інструментом в обробці текстових даних, що дозволяє автоматизувати багато завдань і підвищити ефективність аналізу текстової інформації. Джерела інформації включають різні статті, навчальні посібники та дослідження у галузі обробки природної мови та штучного інтелекту.

1.4 Види вилучення інформації

Вилучення інформації (Information Extraction, IE) включає різні методи і техніки для отримання структурованих даних з неструктурованих текстових джерел. Ось основні види вилучення інформації.

Вилучення іменованих сутностей (Named Entity Recognition, NER)

Цей тип вилучення автоматично розпізнає та класифікує іменовані сутності у тексті, такі як імена людей, назви організацій, місця, дати тощо. Наприклад, у реченні "Microsoft була заснована Біллом Гейтсом" "Microsoft" розпізнається як організація, а "Білл Гейтс" як людина. Методи включають використання правил і шаблонів, машинне навчання та глибоке навчання.

Вилучення відносин (Relation Extraction)

Цей тип вилучення виявляє зв'язки між різними сутностями, наприклад, хто є керівником якої організації або які компанії співпрацюють. У реченні "Google придбала YouTube у 2006 році" вилучається відношення "придбання" між "Google" і "YouTube". Методи включають машинне навчання, глибоке навчання та аналіз залежностей.

Вилучення подій (Event Extraction)

Цей тип вилучення виявляє події у тексті, включаючи дії, учасників, час та місце подій. Наприклад, у реченні "Apple випустила новий iPhone у вересні" подія "випуск" пов'язує "Apple" з "новим iPhone" і "вереснем". Методи включають нейронні мережі, моделі послідовностей та розпізнавання шаблонів.

Вилучення фактів (Fact Extraction):

Цей тип вилучення збирає та структурує конкретні факти з тексту, які можуть бути використані для заповнення баз даних або створення знань. Наприклад, вилучення інформації про нові наукові відкриття з наукових статей. Методи включають аналіз тексту, семантичний аналіз та синтаксичний аналіз.

Вилучення атрибутів (Attribute Extraction)

Цей тип вилучення витягує атрибути або характеристики об'єктів, таких як ціна продукту, технічні характеристики, опис тощо. Наприклад, вилучення ціни і характеристик товару з опису на веб-сайті магазину. Методи включають машинне навчання, глибоке навчання та контекстуальний аналіз.

Вилучення емоцій і настрою (Sentiment Analysis)

Цей тип вилучення визначає емоційне забарвлення тексту, наприклад, позитивне, негативне або нейтральне. Наприклад, аналіз відгуків користувачів для визначення їхнього настрою щодо продукту або послуги. Методи включають обробку природної мови (NLP), машинне навчання та глибоке навчання.

1.5 Етапи обробки тексту

Обробка тексту — це багатокроковий процес, який включає різні етапи для перетворення неструктурованих текстових даних у структури, що можуть бути оброблені алгоритмами машинного навчання або іншими системами. Основні етапи обробки тексту включають:

-попередня обробка тексту - цей етап включає кілька підетапів, спрямованих на очищення тексту і підготовку його до подальшого аналізу.

-токенізація: розбиття тексту на окремі слова або токени.

-нормалізація: приведення слів до базової або нормальної форми (наприклад, зниження до нижнього регістру).

-видалення стоп-слів: виключення загальних слів (наприклад, "і", "але", "що"), які не несуть значущої інформації.

-лематизація і стемінг: приведення слів до їх базової форми (лема) або кореня (стем).

Векторизація тексту - преобразование текстових даних у числові представлення для подальшої обробки.Bag of Words (BoW): Представлення тексту у вигляді вектора частот слів.TF-IDF (Term Frequency-Inverse Document Frequency): Оцінка важливості слів у тексті на основі їх частоти і зворотної частоти в документі.Word Embeddings: Використання моделей, таких як Word2Vec або GloVe, для представлення слів у вигляді векторів у багатовимірному просторі.
Розпізнавання іменованих сутностей (NER):

-виявлення та класифікація ключових сутностей в тексті, таких як імена людей, організацій, місця тощо.

-вилучення відносин

-виявлення зв’язків між сутностями, які були розпізнані на попередньому етапі.

Розбір тексту (Parsing) - аналіз граматичної структури тексту для розуміння взаємозв’язків між словами.Синтаксичний розбір: Визначення граматичних структур (частини мови, синтаксичні ролі).Семантичний розбір: Визначення значення слів і фраз у контексті.

Аналіз настрою (Sentiment Analysis) -визначення емоційного забарвлення тексту, наприклад, позитивного, негативного або нейтрального настрою.

Тематичне моделювання (Topic Modeling) - виявлення основних тем або категорій у великому наборі текстових даних.Latent Dirichlet Allocation (LDA): Популярний метод для автоматичного виділення тем у текстових документах.Non-Negative Matrix Factorization (NMF): Інший метод для тематичного моделювання, що розбиває текст на множину тем.

Вилучення подій - виявлення подій у тексті, включаючи дії, учасників, час та місце подій.

Генерація тексту - створення нового тексту на основі аналізу вхідних текстових даних, наприклад, резюмування або автоматичне написання статей.

Візуалізація даних - представлення результатів аналізу у вигляді графіків, діаграм або інших візуальних засобів для полегшення розуміння і інтерпретації даних.

Використання - обробка тексту застосовується в багатьох галузях, включаючи пошукові системи, аналітику даних, системи рекомендацій, аналіз соціальних мереж, бізнес-аналітику та багато інших.

1.6 Популярні метрики

1.6.1 Відстань Хеммінга

Відстань Хеммінга (Hamming Distance) — це метрика, яка використовується для вимірювання різниці між двома рядками однакової довжини. Вона визначається як кількість позицій, у яких відповідні символи цих рядків відрізняються. Цей показник часто використовується в теорії кодування, зокрема для виявлення та виправлення помилок в кодах, а також в інших областях, таких як генетика, обробка текстів і машинне навчання. Використання:

-теорія кодування - відстань Хеммінга використовується для виявлення та виправлення помилок у переданих повідомленнях. Наприклад, якщо відстань Хеммінга між двома кодовими словами дорівнює

-біоінформатика - відстань Хеммінга може використовуватися для порівняння генетичних послідовностей та визначення різниць між ними.

-обробка текстів - вона може використовуватися для порівняння рядків та виявлення незначних відмінностей між ними, таких як орфографічні помилки.

-машинне навчання - використовується в класифікаційних алгоритмах і для оцінки схожості між векторними представленнями.

1.6.2 Відстань Левенштейна

Відстань Левенштейна, також відома як редакційна відстань, є метрикою, яка визначає мінімальну кількість одно символьних операцій (вставка, видалення або заміна), необхідних для перетворення одного рядка в інший. Ця метрика широко використовується в різних областях, включаючи обробку природної мови, біоінформатику та теорію кодування.

1.6.3 Редакційний припис

Редакційний припис (редагування) — це процес внесення змін до тексту з метою покращення його ясності, точності, стилю та граматичної правильності. В контексті обробки тексту та обчислювальної лінгвістики, редакційний припис часто використовується в алгоритмах для автоматичної корекції помилок, включаючи виправлення орфографії, граматики і стилістичних помилок.

Основні етапи редакційного припису:

-орфографічні помилки: пошук і виправлення неправильно написаних слів.

-граматичні помилки: виявлення неправильних граматичних конструкцій.

-стилістичні помилки: пошук і виправлення стилістичних недоліків, таких як тавтології, зайві слова або недоречне використання фраз.

Аналіз тексту:

-токенізація: розбиття тексту на окремі слова або токени.

-частина мови (POS) тагінг: визначення частин мови для кожного слова в тексті.

Використання моделей, навчених на великих наборах даних для автоматичного виправлення помилок.

1.6.4 Відстань Дамерау-Левенштейна

Відстань Дамерау-Левенштейна (Damerau-Levenshtein distance) — це метрика, що визначає мінімальну кількість одно символьних операцій (вставка, видалення, заміна та транспозиція) необхідних для перетворення одного рядка в

інший. Ця відстань є узагальненням відстані Левенштейна з додатковою операцією транспозиції (обмін сусідніми символами).

1.7 Стоп-слова

Стоп-слова - це слова, які часто зустрічаються в текстах і зазвичай не несуть значущої інформації для аналізу. Вони можуть бути використані для побудови семантичних моделей, вилучення ключових слів або кластеризації текстів.

В процесі обробки тексту стоп-слова зазвичай виключаються з аналізу, оскільки вони не приносять додаткової інформації і можуть спотворити результати. Прикладами стоп-слів можуть бути артиклі, сполучники, прийменники та інші слова, які часто зустрічаються, але не несуть суттєвого значення для розуміння контексту.

1.8 Стемінг

Стемінг - це процес обробки тексту, який полягає у вилученні основи (стему) слова шляхом відкидання афіксів (приставок, суфіксів, закінчень). Основна мета стемінгу - зведення слова до його базової форми для полегшення подальшої обробки тексту та аналізу. Наприклад, слова "running", "runs" та "runner" мають одну стему "run".

Це дозволяє розглядати їх як одне слово, що спрощує задачі аналізу тексту, такі як пошук, порівняння та категоризація.Стемінг може виконуватися за допомогою різних алгоритмів, таких як алгоритм Портера або Ланкастера.

Алгоритми використовують правила для вилучення афіксів та перетворення слова на його базову форму. Хоча стемінг може допомогти у стандартизації тексту для подальшого аналізу, варто пам'ятати, що в деяких випадках він може призвести до втрати точності, оскільки він може перетворити різні слова на однакові стеми, що мають різний контекст.

1.9 Дослідження даних

Дослідження даних (Data Mining) – це автоматизований процес аналізу великих обсягів інформації, спрямований на виявлення корисних знань та шаблонів. Його мета – розкрити приховані закономірності та зв'язки в текстових даних.

Аналіз даних допомагає - виявити правила та закономірності: Ці правила ґрунтуються на статистичних даних і описують поведінку об'єктів та подій.

Отримати нові знання: Дослідження даних може допомогти знайти нові знання, які раніше не були відомі або не підтверджувалися існуючими теоріями.

Зробити висновки, які мають практичну цінність: Ці висновки можуть допомогти приймати кращі рішення, економити час та ресурси, а також отримувати економічну вигоду.

Зрозуміти складні явища: Дослідження даних може допомогти розкрити причини та наслідки складних явищ, які складно пояснити без аналізу великих обсягів інформації.

Data Mining використовується для вирішення різних задач:

-класифікація: Розподіл даних на групи за певними ознаками.

-кластеризація: Об'єднання даних у групи без чітко визначених ознак.

-асоціація: Виявлення зв'язків між подіями або об'єктами.

-послідовність: Виявлення закономірностей у послідовності подій.

-прогнозування: Передбачення майбутніх значень на основі історичних даних.

-візуалізація: Перетворення даних у зрозумілі графіки та діаграми.

-підведення підсумків: Виявлення найважливіших характеристик даних.

-виявлення відхилень: Виявлення аномальних значень, які можуть свідчити про проблеми або помилки.

Приклади використання Data Mining:

-маркетинг - виявлення цільової аудиторії, аналіз поведінки клієнтів, розробка ефективних рекламних кампаній.

-фінанси - виявлення шахрайства, оцінка ризиків, прогнозування цін на активи.

-медицина - діагностика захворювань, прогнозування результатів лікування, розробка нових ліків.

-виробництво - оптимізація виробничих процесів, прогнозування попиту на продукцію, виявлення дефектів.

-державне управління - аналіз соціальних проблем, прогнозування злочинності, оптимізація бюджетних витрат.

Data Mining – це потужний інструмент, який може допомогти вирішити багато проблем в різних галузях. Завдяки автоматизованому аналізу великих обсягів даних, дослідження даних дозволяє отримати нові знання та зробити висновки, які мають практичну цінність.

Початково для роботи з даними та їх обробки використовували спеціально створені мови запитів для роботи з базами даних. Згодом основна увага зосередилася на аналітиці, для вирішення завдань якої потрібні математичні алгоритми у поєднанні з методами машинного навчання. До методів розв’язання відносяться: нейронні мережі, дерева рішень, еволюційні алгоритми.

Добування даних збору інформації з різних джерел для подальшого аналізу та використання. Цей процес може включати в себе різноманітні методи, такі як скрапінг веб-сторінок, використання API для доступу до даних, а також ручний збір інформації.

Скрапінг веб-сторінок полягає в автоматизованому або напівавтоматизованому зборі даних або Scrapy, для аналізу HTML-коду сторінок та вилучення необхідної інформації.

API (інтерфейс програмування застосунків) надає зручний спосіб отримання даних з веб-серверів. Багато веб-сайтів та онлайн сервісів надають API для доступу до своїх даних. Зазвичай це використовується для отримання структурованих даних у форматі JSON або XML.

Ручний збір даних використовується у випадках, коли автоматизовані методи не є можливими або неефективними. Це може включати в себе ручне заповнення архівів, анкет або збір інформації з друкованих джерел.

Загалом, добування даних є важливим етапом в аналізі даних та розробці програмних продуктів, оскільки якісні дані є основою для подальшого аналізу та прийняття рішень.

Додаткові методи добування даних включають аналіз великих наборів даних (Big Data), використання датчиків та IoT (інтернет речей), а також співпрацю зі спеціалізованими постачальниками даних.

Аналіз великих наборів даних дозволяє отримати інформацію з великих обсягів даних, що зберігаються у різних джерелах, таких як бази даних, дата-центри, веб-логи тощо. Цей аналіз може здійснюватися за допомогою спеціальних програмних засобів та алгоритмів, які дозволяють виявляти закономірності та тренди в даних.

Використання датчиків та IoT дозволяє отримувати дані в реальному часі з різних джерел, таких як сенсори, пристрої та обладнання, які підключені до мережі Інтернет. Ці дані можуть бути використані для моніторингу та аналізу різних параметрів, таких як температура, вологість, тиск тощо.

Співпраця з постачальниками даних дозволяє отримати доступ до спеціалізованих наборів даних, які можуть бути корисні для конкретних аналітичних завдань. Це може включати в себе підписку на послуги з доступу до даних або співпрацю зі спеціалізованими компаніями, які збирають та обробляють дані в певних галузях.

Ще одним методом добування даних є використання соціальних медіа та веб-скрапінгу. Соціальні мережі, такі як Twitter, Facebook, LinkedIn, можуть бути цінним джерелом інформації про публічні думки, тренди та події. Веб-скрапінг використовується для автоматичного отримання даних з веб-сайтів шляхом аналізу їх HTML-коду. Цей підхід дозволяє отримувати структуровану інформацію з різних джерел в Інтернеті.

1.9.1 Інформаційний пошук

Інформаційний пошук - це завдання пошуку інформації в неструктурованих базах даних. Об'єктами в базі можуть бути текстові документи, зображення та відео. Суть задачі полягає у знаходженні відповідних об'єктів, які відповідають пошуковому запиту. До завдань інформаційного пошуку належить пошук інформації в документах, знаходження самих документів, вилучення метаданих та тегів, пошук частин тексту, зображень, відео та звуку в базах даних.

Методи вирішення цієї обробки природної мови (NLP), використання спеціалізованих мов запитів та попередню структурування даних для полегшення пошуку.

Застосування Named Entity Recognition (NER)

NER використовується для розпізнавання та класифікації різних типів сутностей в тексті, таких як імена людей, назви місць, дати, електронні адреси тощо. Використання NER дозволяє класифікувати текстові дані та виявляти їхні зв'язки.

Наприклад, в інтернет-магазинах NER може бути використаний для автоматичного тегування товарів за їхніми характеристиками та атрибутами. Він також може бути застосований для класифікації відгуків користувачів за їхнім емоційним відтінком та виявлення сутностей, які згадуються в тексті.

В Інтернеті NER широко використовується в пошукових системах для покращення точності та релевантності пошукових результатів, а також для рекомендацій товарів в інтернет-магазинах.

Техніка важлива для різних завдань таких як розпізнавання інформації з текстових документів, аналіз новинних статей, витягування структурованої інформації з великих корпусів тексту та багато іншого.

Основні методи використання NER включають у себе створення моделей машинного навчання, які іменовані сутності у тексті. Ці моделі зазвичай навчаються де іменовані сутності позначені анотаторами. Вони можуть

використовувати різні такі моделі глибокого навчання, CRF (Conditional Random Fields), або комбінації різних методів.

NER використовується в різних галузях, включаючи пошукові системи (для покращення пошукових запитів та ранжування результатів), аналіз текстів соціальних медіа (для виявлення осіб, компаній, місць у повідомленнях), сегментацію тексту, розпізнавання мови та багато іншого.

Техніка NER є важливим елементом в розвитку різних програмних рішень, що працюють з текстовими даними, і грає важливу роль у розвитку систем штучного інтелекту та обробки природної мови.

Крім використання моделей машинного навчання, для NER також застосовуються правила та шаблони, які дозволяють виявляти іменовані сутності на основі певних лінгвістичних ознак. Наприклад, для виявлення назв організацій можуть використовуватися ключові слова, що вказують на організацію (наприклад, "корпорація", "компанія", "фонд"), або специфічні синтаксичні конструкції.

Також важливим аспектом NER є побудова власних корпусів тексту, які містять анотовані іменовані сутності. Ці дані можуть бути використані для навчання та оцінювання моделей, а також для створення нових методів виявлення іменованих сутностей.

У розробці систем NER важливо також враховувати особливості мови та типів іменованих сутностей, які потрібно виявляти. Наприклад, для англійської мови можуть бути застосовані інші методи порівняно з українською або китайською мовами, оскільки різні мови мають різні синтаксичні та морфологічні особливості.

Важливим аспектом розробки систем NER є також оцінка їх точності та ефективності. Для цього використовуються метрики, такі як точність (precision), відновлення (recall) та F-міра (F-measure), які дозволяють оцінити якість виявлення іменованих сутностей.

У практичних застосуваннях систем NER також часто використовуються у поєднанні з іншими методами обробки природної мови, такими як аналіз

синтаксичної структури тексту, класифікація тексту та машинний переклад. Такий підхід дозволяє отримати більш точні результати та покращити роботу систем обробки тексту.

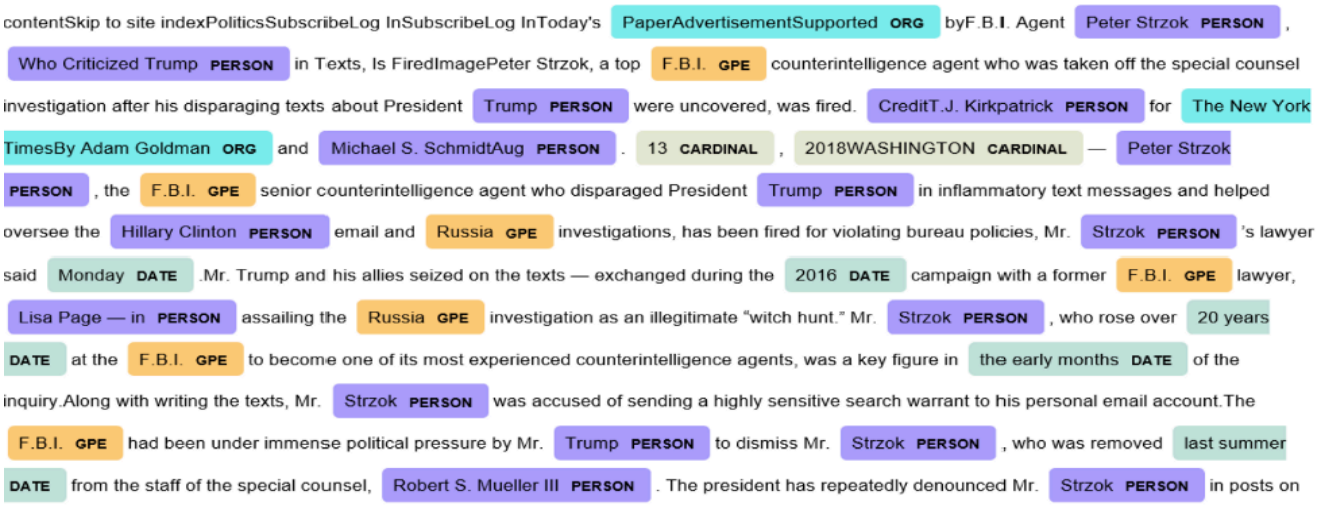


Рис. 1.1 Приклад виявлення сутностей в тексті

Типові підзадачі NER

1. Задача знаходження іменованих сутностей (сегментація тексту на значимі об’єкти).

2. Класифікація знайдених іменованих сутностей (визначення ймовірності відношення сутності до кожного з класів).

3*. Формування результуючої вибірки з сутностями лише заданих класів (категорій).

Методи вирішення задачі Named Entity Recognition (NER)

Існують різні методи розв'язання задачі NER, кожен з яких має свої переваги та недоліки. Одним із методів є використання словників сутностей. У цьому методі слова тексту порівнюються зі словником сутностей, і якщо вони співпадають, вони виділяються як сутності та класифікуються відповідно до визначених груп. Однак цей метод може вимагати значних ресурсів, особливо при великих об'ємах словників.

Іншим методом є визначення граматики та лексичних правил мови. Ці правила використовуються в якості алгоритмів для розпізнавання сутностей.

Програма намагається вгадати, чи є дане слово чи фраза сутністю та його типом. Цей метод може бути ефективним, але він часто працює недостатньо ефективно для різних мов та може потребувати додаткового налаштування для кожної конкретної мови.

Найпопулярнішим методом вирішення задачі NER на сьогоднішній день є використання нейронних мереж. Для цього необхідно навчити мережу на основі баз даних з правилами формування, шаблонів розпізнавання та словників сутностей. Програма працює за принципом проходження тексту та спроби вгадати, чи є певне слово або фраза сутністю. Точність розпізнавання залежить від обсягу та якості навчальної бази даних. Також для навчання можуть використовуватися вже оброблені тексти.

Багато мов програмування мають вбудовані модулі для розпізнавання іменованих сутностей. Крім того, існують різноманітні бібліотеки, такі як NLTK, HanLP, PullEnti, AdaptNLP, Spacy, Stanford CoreNLP, які дозволяють виявляти та працювати з іменованими сутностями. За останні роки також з'явилося багато веб-сервісів для вирішення задачі NER, які мають інтеграцію з іншими задачами обробки природної мови, такими як розпізнавання тексту на зображеннях, аналіз емоційного змісту тексту та генерація тексту.

Методи вирішення задачі Named Entity Recognition (NER) - це широкий спектр підходів та технік, які використовуються для виявлення та класифікації іменованих сутностей (наприклад, осіб, організацій, місць тощо) в текстових даних. Ось деякі з найпоширеніших методів:

1. Підхід на основі правил: Цей метод використовує набір правил або шаблонів для виявлення іменованих сутностей на основі їх структури, контексту або інших характеристик. Наприклад, правило може бути сформульоване ідентифікувати будь-яке слово або фразу, яка починається з великої літери та відноситься до організаційного або географічного імені.
2. Методи машинного навчання: Вони використовують алгоритми машинного навчання для навчання моделей, які можуть автоматично виявляти іменовані сутності на основі прикладів даних. Ці методи включають у себе такі

техніки, як використання нейронних мереж, статистичні моделі (наприклад, CRF - Conditional Random Fields), а також алгоритми класифікації, наприклад, на основі Support Vector Machines (SVM) або рішіть дерева.

3. Застосування глибокого навчання: Глибокі нейронні мережі, зокрема рекурентні нейронні мережі (RNN) та зокрема Long Short-Term Memory (LSTM) або Transformer-based моделі, дозволяють досягти вражаючих результатів у вирішенні задачі NER. Ці моделі можуть ефективно враховувати контекст та послідовність слів в тексті для точного виявлення іменованих сутностей.
4. Застосування попередньо навчених векторних представлень: Використання векторних представлень слів, таких як Word Embeddings (наприклад, Word2Vec, GloVe або FastText), може допомогти поліпшити продуктивність моделі NER, навченої на обмеженому обсязі даних, шляхом використання попередньо навчених векторних представлень слів.

Ці методи можуть бути застосовані як окремо, так і в поєднанні один з одним для досягнення більшої точності та ефективності у вирішенні задачі NER.

Поміж сучасних методів NER варто відзначити також:

5. Бертовські моделі: Ці моделі, засновані на архітектурі Transformer, виявляють іменовані сутності шляхом розуміння контексту всього тексту. Вони здатні досягати вражаючих результатів, зокрема на мовах з складною граматикою чи вираженими діалектами.
6. Ансамбль методи: Комбінування результатів декількох моделей може покращити точність і стійкість до різних типів даних і шуму в тексті. Використання ансамблів може знизити ризик перенавчання та покращити загальну надійність системи.
7. Самонавчання: Техніки самонавчання, такі як Active Learning або Reinforcement Learning, дозволяють моделі взаємодіяти з джерелами інформації та покращуватися з часом, пристосовуючись до нових даних та викликів.

8. Контекстно-залежні або динамічні методи: Ці методи використовують інформацію про контекст та взаємозв'язки між різними частинами тексту для кращого розуміння іменованих сутностей. Вони можуть виявляти сутності, що змінюють свій контекст залежно від текстового оточення.
9. Методи активного навчання: Ці методи використовуються для ефективного збирання та використання додаткових міток даних для покращення навчання моделей NER. Вони дозволяють моделі активно вибирати найінформативніші приклади для навчання, що може зменшити необхідну кількість позначених даних.

Ці методи і техніки надають широкий спектр інструментів для вирішення задачі Named Entity Recognition у різних контекстах та з різними вимогами. Комбінування їх може допомогти побудувати найефективнішу та найточнішу систему виявлення іменованих сутностей.

Поміж сучасних методів NER варто відзначити також методи, що базуються на нейронних мережах, такі як BiLSTM-CRF та BiGRU-CRF. Ці моделі використовують рекурентні шари (LSTM або GRU) для аналізу послідовностей та узгодженого виявлення іменованих сутностей. Вони здатні досягати високої точності завдяки своїй здатності аналізувати контекст тексту на різних рівнях складності.

Також варто відзначити широке використання предобробки тексту та векторного вбудовування слів, що дозволяє враховувати семантичні зв'язки між словами під час навчання моделі.

Паралельно з нейронними методами, існують і традиційні підходи до вирішення задачі NER, такі як правила на основі регулярних виразів або правила, побудовані на граматичних особливостях мови.

Методи можуть бути ефективними в певних випадках, особливо якщо доступна обмежена кількість даних або якщо завдання виявлення іменованих сутностей має відносно просту структуру. Такі традиційні методи можуть бути корисними як перший крок у вирішенні задачі NER або як частина складніших

систем обробки природної мови, де необхідно поєднувати їх з іншими підходами, такими як машинне навчання.

Додатковими методами вирішення задачі Named Entity Recognition (NER) є використання словників і тезаурусів, правила на основі синтаксичних шаблонів, а також комбіновані підходи, які об'єднують кілька методів для досягнення кращих результатів.

Використання словників може бути ефективним для визначення іменованих сутностей, які мають чіткі терміни або ключові слова, такі як назви організацій або місць. Синтаксичні шаблони використовують граматичні структури речень для визначення іменованих сутностей на основі їх синтаксичних зв'язків з іншими словами у реченні.

Комбіновані підходи можуть поєднувати кілька методів, наприклад, використовуючи словники для попереднього виявлення іменованих сутностей та потім застосовуючи правила на основі синтаксичних шаблонів для подальшої роботи з текстом. Кожен з цих методів має свої переваги і недоліки і може бути використаний залежно від конкретних вимог задачі та характеристик даних.

Ще одним методом вирішення задачі Named Entity Recognition є використання глибокого навчання, або комбінації цих підходів, таких як багатошарові рекурентні згорткові мережі (LSTM-CNN).

Визначати іменовані сутності без необхідності вручну складати правила або використовувати заздалегідь підготовлені словники. Ці моделі можуть бути навчені на великих корпусах тексту, що дозволяє їм ефективно виявляти іменовані сутності в реальному часі та адаптуватися до різноманітних лінгвістичних особливостей та структур даних. Використання глибокого навчання для вирішення задачі NER є актуальним і досліджуваним напрямком у сфері обробки природної мови.

Ще одним підходом до вирішення задачі Named Entity Recognition є використання правилових систем, які базуються на заздалегідь складених правилах та шаблонах. Ці системи використовують експертно складені правила

для ідентифікації іменованих сутностей на основі специфічних лінгвістичних особливостей тексту.

Хоча цей підхід може бути менш автоматизованим порівняно з методами машинного навчання, він може бути ефективним у випадках, коли існують чіткі шаблони для іменованих сутностей або коли потрібна висока точність в рядках даних, які мають обмежений обсяг або динамічну структуру. Випадки, коли важко зібрати достатньо великий та різноманітний набір даних для навчання моделей машинного навчання, або коли потрібно враховувати специфічні вимоги домену задачі.

Ще одним підходом до розв'язання задачі Named Entity Recognition є використання глибокого навчання, зокрема рекурентних нейронних мереж (RNN), зокрема Long Short-Term Memory (LSTM) або Gated Recurrent Unit (GRU), або згорткових нейронних мереж (CNN), а також їх комбінацій. Ці моделі можуть автоматично визначати іменовані сутності, використовуючи глибокі рекурентні або згорткові шари для аналізу послідовностей слів у тексті та вивчення їхніх контекстуальних зв'язків. Глибокі нейронні мережі дозволяють автоматично витягувати властивості та ознаки з тексту, не потребуючи ручної інженерії ознак, що робить їх особливо потужними для задач обробки природної мови. Такий підхід може використовувати навчальні дані для навчання моделі на основі прикладів та покращення її точності з часом завдяки здатності нейронних мереж до самонавчання на великих обсягах даних.

Ще одним підходом є використання моделей, заснованих на трансформерах. Трансформери - це тип архітектури глибоких нейронних мереж, які спеціалізуються на обробці послідовностей.

BERT може ефективно вирішувати задачі NER, оскільки він здатний враховувати контекст слова у всьому тексті. Він може автоматично вивчати контекстуальні зв'язки між словами та визначати іменовані сутності з високою точністю.

Крім того, можна використовувати попередньо навчені моделі BERT, що дозволяє використовувати великі обсяги текстових даних для покращення

результатів. Такий підхід стає все більш популярним у сфері обробки природної мови та дозволяє досягти вражаючих результатів у завданнях розпізнавання іменованих сутностей.

Ще одним перспективним методом є використання CRF (Conditional Random Fields), що є статистичним методом машинного навчання, особливо ефективним у задачах обробки текстів. CRF дозволяє моделювати залежності між послідовностями міток, що дозволяє враховувати контекст при класифікації. Також можна використовувати глибокі нейронні мережі, зокрема рекурентні нейронні мережі (RNN) або їх модифікації, такі як Long Short-Term Memory (LSTM) або Gated Recurrent Unit (GRU). Ці архітектури спеціалізуються на обробці послідовностей та можуть успішно вирішувати задачі NER, зокрема враховуючи довгострокові залежності в тексті. Таким чином, використання різних методів, таких як трансформери, CRF та глибокі нейронні мережі, може допомогти покращити результати в задачі визначення іменованих сутностей.

Проблеми NER:

-неоднозначності на рівні мови (багатозначності слів, омоніми)

-низька якість вхідного текстового корпусу (наприклад, друкарські помилки)

-проблема доменів – моделі, розроблені для одного домену не обов’язково гарно працюють в інших предметних галузях

-одна сутність може бути представлена як одним словом, так і виразом: «Генрі Форд» -> «засновник компанії Форд Мотор»

-проблема корефрентності. Кореферентність – лінгвістичний термін, який позначає посилання декількох компонентів тексту до однієї і тієї ж сутності.

Недосконаліть моделей

-нестійкість до друкарських помилок

-великі розміри словників

-обмеженість моделей

-ручна обробка даних

1.9.2 Аналіз аналогів

Засоби для вилучення цінної інформації з комерційних документів можуть бути використані наступні:

-оптичне розпізнавання символів (OCR) і програми розпізнавання тексту: Ці інструменти дозволяють автоматично виділяти, аналізувати та витягувати текстову інформацію з документів у різних форматах, включаючи PDF, JPG, PNG тощо. Приклади таких програм включають Textract, Pytesseract та Apache Tika.

-автоматизовані системи обробки документів (DMS): Ці системи розроблені спеціально для управління та обробки документів у великих обсягах. Вони можуть включати функції розпізнавання тексту, класифікації документів та екстракції даних. Прикладом такого інструменту є Rossum.

-спеціалізовані інструменти для обробки таблиць: Деякі документи можуть містити таблиці з важливою інформацією, яку необхідно вилучити. Інструменти, такі як Tabula, спрямовані на вилучення даних з таблиць у форматі PDF.

1.9.3 Textract

Textract - це сервіс AWS, який надає можливість вилучення тексту, форматування, табличних даних та інших елементів з різних типів документів, включаючи PDF, DOCX, JPG і багато інших. Використовуючи штучний інтелект, Textract розпізнає структуру документа і надає доступ до окремих елементів для подальшого аналізу.

Особливості:

-вилучення тексту, таблиць, графіків та інших елементів з різних типів документів.

-повна автоматизація процесу розпізнавання без необхідності ручної настройки.

-широкі можливості інтеграції з іншими сервісами AWS та зовнішніми системами.

-висока швидкість та точність розпізнавання завдяки використанню передових алгоритмів машинного навчання.

Недоліки:

-залежність від сервісу AWS: використання Textract передбачає залежність від інфраструктури та сервісів AWS, що може призвести до обмежень у використанні та вартості.

-вартість використання сервісу AWS підпорядковується тарифним планам, що може призвести до додаткових витрат для користувачів з великим обсягом даних.

Приклад готового рішення за допомогою Textract, наведено на рис. 1.2

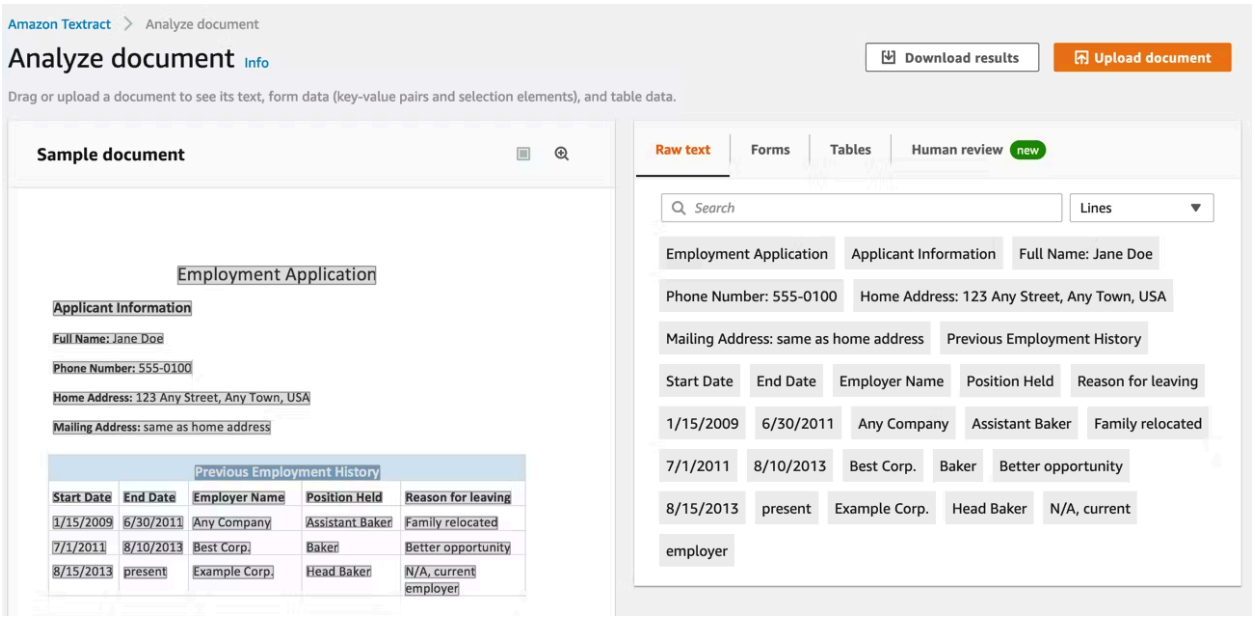


Рис. 1.2 Приклад готового рішення за допомогою Textract

1.9.4 Pytesseract

Pytesseract - це Python-бібліотека, яка надає можливість розпізнавання тексту на зображеннях за допомогою Tesseract OCR (Optical Character Recognition). Вона дозволяє використовувати потужність Tesseract для розпізнавання тексту у реальному часі та автоматизувати процес обробки зображень.

Особливості:

-простий у використанні API для розпізнавання тексту зі зображень.

-підтримка різних мов та алфавітів.

-можливість налаштування параметрів розпізнавання для покращення точності.

-висока швидкість роботи та низькі вимоги до ресурсів.

Недоліки:

-обмежена точність: Pytesseract може демонструвати меншу точність розпізнавання у порівнянні з комерційними аналогами, особливо при обробці складних зображень або текстів зі складним форматуванням.

-необхідність підгонки параметрів: для досягнення оптимальної точності розпізнавання може знадобитися налаштування параметрів бібліотеки, що може бути часомістким та вимагати досвіду.

1.9.5 Apache Tika

Apache Tika - це бібліотека для вилучення текстових даних з різних типів документів, включаючи PDF, DOCX, HTML, XML і багато інших. Вона надає широкий спектр можливостей для розпізнавання та аналізу структурованих даних з документів різних форматів.

Особливості:

-підтримка багатьох форматів файлів для вилучення тексту та інших даних.

-простий у використанні API для розробників.

-можливість використання алгоритмів машинного навчання для покращення точності розпізнавання.

-широкі можливості налаштування параметрів розпізнавання для різних типів документів та потреб користувача.

Недоліки:

-обмежені можливості розпізнавання зображень: в порівнянні з іншими інструментами Apache Tika має обмежені можливості розпізнавання тексту на зображеннях та інших нетекстових елементах.

-великий обсяг пам'яті: для обробки документів великого розміру Apache Tika може вимагати значних ресурсів пам'яті, що може призвести до перевищення обсягу доступної пам'яті на сервері.

1.9.6 Rossum

Rossum - це розроблений на базі штучного інтелекту інструмент для автоматизації обробки документів. Він надає можливість вилучення тексту, таблиць та інших елементів з документів, а також автоматичну обробку та аналіз структурованих даних.

Особливості:

-повністю автоматизований процес обробки документів без необхідності втручання людини.

-вилучення та аналіз структурованих даних з різних типів документів.

-широкі можливості інтеграції з іншими системами та сервісами для обміну даними та автоматизації робочих процесів.

-висока швидкість та точність розпізнавання завдяки використанню передових алгоритмів машинного навчання та обробки природної мови.

Приклад интерфейсу Rossum, наведено на рисунку 1.3.

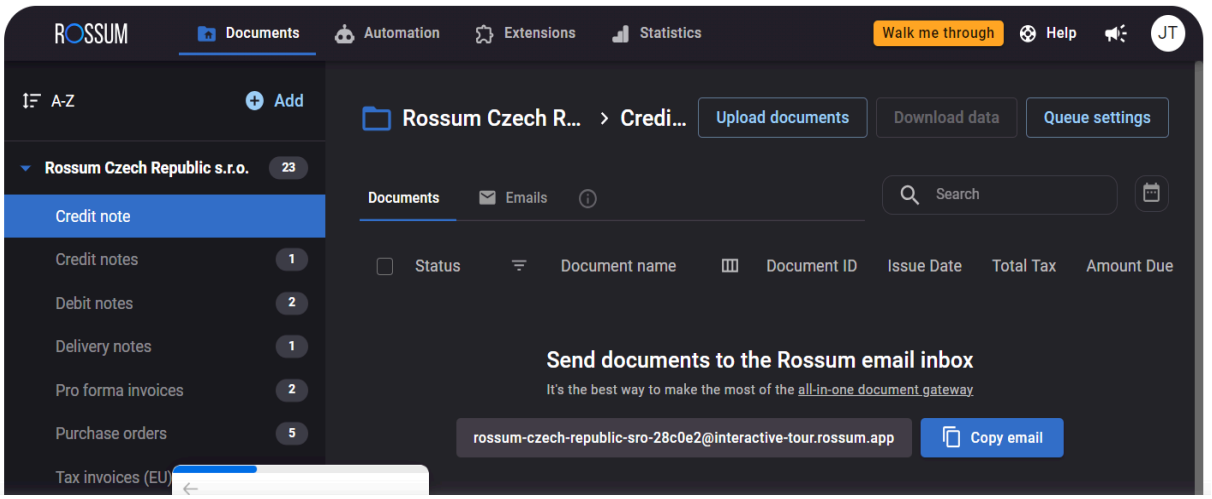


Рис. 1.3 Приклад интерфейсу Rossum

Недоліки:

-обмежена локалізація: Rossum може показати гіршу точність розпізнавання при роботі з документами, які містять специфічні для певних регіонів мови, форматування або структуру.

-вартість: як і в інших рішеннях, використання Rossum може призвести до додаткових витрат на підтримку та розширення функціоналу, що може стати фактором у виборі.

В таблиці 1.1 наведені результати аналізу характеристик додатків.

Таблиця 1.1

Аналіз характеристик додатків

Функціональна можливість	Textract	Pytesseract	Apache Tika	Rossum	Розроблений застосунок
Вилучення номер телефону	+	+	–	–	+
Вилучення електронної пошти	+	–	+	–	+
Українська локалізація	–	–	–	–	+
Збереження результатів обробки	–	–	–	–	+
Вилучення дат заходів	–	+	+	+	+
Вилучення назв компанії	+	–	+	+	+
Вилучення адрес	–	–	–	–	+
Завантаження файлу TXT	–	–	–	+	+
Користувач повинен робити додаткові налаштування	+	+	+	+	–
Орієнтація на комерційні документи	–	–	–	–	+

2 ОБҐРУНТУВАННЯ ВИБОРУ ЕЛЕМЕНТІВ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Обґрунтування вибору мови програмування

Python - це потужна та популярна мова програмування, яка має декілька сильних сторін.

Простота та Читабельність: Python має простий і зрозумілий синтаксис, що робить його ідеальним для новачків у програмуванні. Код, написаний на Python, легко читати і розуміти, що полегшує співпрацю між розробниками.

Багатий велику кількість сторонніх бібліотек та фреймворків для різних цілей, включаючи машинне навчання, обробку природної мови, веб-розробку та багато іншого. Це робить Python потужним інструментом для швидкої розробки програмного забезпечення.

Широке застосування в машинному навчанні: Python є однією з основних навчання моделей машинного навчання.

Швидкодія: Python може бути менш ефективним за інші мови програмування, такі як C++ або Java, особливо у випадках, коли потрібна велика швидкодія виконання.

Обмежена мобільність - Python не є ідеальним вибором для розробки мобільних додатків. Хоча існують фреймворки, такі як Kivy, для створення мобільних додатків на Python, але це не дуже поширений вибір порівняно з мовами, такими як Java або Kotlin для Android або Swift для iOS.

Отже, хоча Python є потужною мовою програмування з багатьма перевагами, перед вибором мови для розробки програмного забезпечення варто врахувати конкретні вимоги проекту та досвід команди розробників. У випадку розробки програмного забезпечення для вилучення інформації з комерційних документів за допомогою Python, ця мова може бути оптимальним вибором, особливо з урахуванням її широкого застосування у сфері машинного навчання.

Ваш аналіз Python явно відображає його переваги та недоліки, а також вказує на його відповідність для вашого проекту. Ось кілька ключових моментів, які підкреслюють важливість використання Python для вашого програмного забезпечення:

-простота використання: Python має простий синтаксис, що полегшує розробку програм та сприяє їхній зрозумілості.

-багатий набір бібліотек та фреймворків: фреймворкі, зокрема для машинного навчання, робить Python ідеальним вибором для вашого проекту.

-гнучкість та розширюваність: Python може бути легко розширений за допомогою додаткових модулів, бібліотек та фреймворків, що дає вам більшу гнучкість у вирішенні різноманітних завдань.

-підтримка спільноти: Активна спільнота розробників Python постійно вносить внесок у розвиток мови та підтримку бібліотек, що забезпечує вам доступ до широкого кола рішень та експертної підтримки.

-легкість інтеграції з іншими мовами: Python може бути легко поєднаний з іншими мовами програмування, зокрема з C/C++, що дає вам можливість використовувати найкращі рішення для вашого проекту.

З урахуванням цих переваг, використання Python для розробки програмного забезпечення для вилучення інформації з комерційних документів здається розумним вибором. Ви можете розраховувати на швидкий розвиток та ефективну роботу програми, використовуючи Python.

2.2 Обґрунтування вибору середовища розробки

Під час моєї роботи, я провів аналіз кількох різних інтегрованих середовищ розробки, з метою визначення найбільш оптимального для створення програмного забезпечення, призначеного для вилучення корисної інформації з комерційних документів за допомогою Python.

PyCharm – це потужне середовище розробки для Python, яке надає широкий спектр функцій, таких як налагодження, автодоповнення коду та контроль версій.

Воно також має інтеграцію з такими бібліотеками машинного навчання, як TensorFlow і Keras.

Одне з найпопулярніших та повнофункціональних інтегрованих середовищ розробки для Python. Воно зручний редактор коду та функції швидкого рефакторингу. PyCharm підтримує різні фреймворки для веб-розробки, такі як HTML, а також надає можливості редагування коду. Крім того, воно має хорошу інтеграцію

Це середовище розробки доступне для кількох платформ, macOS, і має як безкоштовну, так і платну професійну версії. Безкоштовна версія PyCharm має достатньо функціональності для більшості завдань, пов'язаних з програмуванням та розробкою. До її головних переваг варто віднести різноманітні вбудовані функції, високу якість підтримки мови Python та підтримку віртуальних середовищ, таких як Anaconda.

2.2.1 Середовище розробки Python IDLE

IDLE - це базова інтегрована середовище розробки, яка переважно використовується початківцями, які хочуть набути практичний досвід у програмуванні на Python. До основних особливостей IDLE можна віднести:

-IDLE Python є кросплатформовою IDE, що дозволяє користувачам бути більш гнучкими у виборі операційної системи.

-розроблено виключно на Python у співпраці з графічним інтерфейсом Tkinter.

-простота використання: IDLE Python.

Додаткові особливості IDLE Python

Підтримка інтерактивного режиму: IDLE Python має інтерактивну оболонку, яка дозволяє миттєво виконувати команди Python та переглядати результати. Це робить її ідеальним інструментом для навчання та експериментів з кодом.

Багатовіконний режим: Крім основного вікна редактора, IDLE дозволяє відкривати декілька вікон для редагування, що зручно великими проектами або паралельного тестування різних частин коду.

Інтеграція з бібліотеками: IDLE Python добре інтегрується з основними бібліотеками Python, такими як NumPy, Pandas та Matplotlib. Це дозволяє використовувати потужні інструменти для наукових обчислень та візуалізації даних безпосередньо в середовищі розробки.

Розширюваність через плагіни: Незважаючи на свою базову функціональність, IDLE Python може бути розширена за допомогою сторонніх плагінів та інструментів, що дозволяє додавати нові можливості та налаштовувати середовище під власні потреби.

Простота налаштування середовища: IDLE не вимагає складної інсталяції чи конфігурації, що робить її доступною для початківців. Це також зручно для викладачів, які можуть швидко підготувати середовище для навчальних занять.

Автозбереження та резервне копіювання: Функція автозбереження у IDLE дозволяє автоматично зберігати роботу, що мінімізує ризик втрати даних у випадку непередбачених обставин, таких як збій системи або вимкнення електроенергії.

IDLE Python може бути ефективно використане для для розробки Це зумовлено наступними причинами:

Широкий набір вбудованих функцій: IDLE надає такі важливі інструменти, як інтерактивна оболонка Python, підсвічування синтаксису, автоматичне вирівнювання та базовий відладчик. Ці інструменти значно полегшують процес розробки та тестування програмного забезпечення.

Інтеграція з машинним навчанням: Завдяки підтримці популярних бібліотек Python для машинного навчання, таких як TensorFlow та Scikit-learn, IDLE дозволяє створювати та тестувати алгоритми для аналізу тексту.

2.1.2 PyQt6

PyQt6 представляє собою високопродуктивну бібліотеку, яка дозволяє створювати графічні користувацькі інтерфейси (GUI) для програм на Python. Ця бібліотека містить у вигляді модулів Python. Основні переваги узагальнити наступним чином:

Комплексність та повнота: PyQt6 базується на відомій Додаткові аспекти, які можна врахувати:

Продуктивність: Однією з важливих характеристик PyQt6 є висока продуктивність. Завдяки ефективному використанню ресурсів та оптимізації коду, GUI-додатки, створені на основі PyQt6, працюють швидко і стабільно, комерційних та критично важливих застосунків.

Модульність та повторне використання коду: PyQt6 підтримує модульний підхід до розробки, що дозволяє розробникам створювати окремі компоненти та віджети, які можуть бути повторно використані в інших проектах. Це сприяє зниженню витрат на розробку та підтримку програмного забезпечення.

Технологіями: PyQt6 легко інтегрується з іншими технологіями та бібліотеками Python, такими як NumPy для обробки числових даних, pandas для аналізу даних, matplotlib для побудови графіків та багатьма іншими. Це забезпечує широкі можливості для розробки складних та функціональних додатків.

Можливості розширення: PyQt6 надає механізми для створення власних розширень та плагінів, що дозволяє додатково розширити функціональність бібліотеки відповідно до конкретних потреб проекту.

Підтримка сучасних стандартів: PyQt6 регулярно оновлюється та підтримує сучасні стандарти розробки програмного забезпечення, що забезпечує сумісність з останніми версіями операційних систем та іншого програмного забезпечення.

Інструменти розробки: PyQt6 включає в себе різноманітні інструменти для розробки, такі як дизайнер інтерфейсів Qt Designer, який дозволяє створювати графічні інтерфейси за допомогою інтуїтивно зрозумілого візуального редактора.

Враховуючи всі ці аспекти, PyQt6 є комплексним та надійним рішенням для розробки сучасних графічних інтерфейсів користувача на Python, відмінним вибором для проектів будь-якої складності та масштабу.

PyQt6 також легко інтегрується з іншими технологіями та бібліотеками Python. Це дозволяє розробникам створювати комплексні додатки, які використовують різноманітні засоби і підходи, наприклад, для обробки даних, роботи з базами даних чи побудови графіків.

Підтримка багатопоточності - PyQt6 забезпечує підтримку багатопоточності, що дозволяє створювати більш ефективні та продуктивні додатки. Це особливо важливо для складних програм, де виконання декількох завдань одночасно може значно підвищити продуктивність.

Легкість навчання - завдяки великій кількості доступної документації, прикладів коду та навчальних матеріалів, PyQt6 є доступною для вивчення навіть для початківців. Це знижує поріг входу для нових розробників та сприяє швидкому освоєнню бібліотеки.

Розширюваність - PyQt6 дозволяє легко створювати власні віджети та елементи управління, що надає додаткову гнучкість у розробці інтерфейсів. Це робить можливим створення унікальних та функціональних GUI, які можуть задовольнити специфічні потреби користувачів.

Висока продуктивність - Завдяки своїй основі на Qt, PyQt6 забезпечує високу продуктивність роботи додатків. Це особливо важливо для розробки інтерфейсів, які повинні швидко реагувати на дії користувача та у реальному часі.

Підтримка мультимедіа - PyQt6 включає засоби мультимедійними даними, такими як аудіо та відео. Це дозволяє створювати інтерактивні додатки з багатим користувацьким досвідом, які можуть відтворювати медіафайли та працювати з потоковими даними.

Інструменти розробки та відладки - PyQt6 надає потужні інструменти для розробки та відладки, які спрощують процес створення додатків. Інтегровані засоби для тестування, профілювання та налагодження допомагають розробникам швидше знаходити та виправляти помилки, а також оптимізувати продуктивність додатків.

Активне співтовариство - Активне співтовариство розробників PyQt6 створює безліч розширень, бібліотек та інструментів, що додатково спрощує процес розробки. Це також означає, що у разі виникнення питань чи проблем завжди можна знайти допомогу та підтримку.

Ці додаткові аспекти роблять PyQt6 ще більш привабливим вибором для розробників, які шукають потужний та гнучкий інструмент для створення графічних інтерфейсів у своїх додатках на Python

IDLE Python також має вбудовану систему допомоги та документації, що робить процес вивчення та розробки більш зручним і продуктивним. Можливість швидко отримати інформацію про функції та методи зробить процес вивчення Python та його бібліотек більш ефективним.

Також, IDLE Python підтримує розширення через сторонні плагіни, що дає можливість розширити його функціональність та адаптувати під індивідуальні потреби розробника. Завдяки цьому, користувачі можуть налаштувати робоче середовище так, як їм зручно, що полегшує роботу та підвищує продуктивність.

Ще однією перевагою IDLE Python є можливість використання його в освітніх цілях. Його зручний інтерфейс та доступність для користувачів різного рівня досвіду роблять його відмінним інструментом для навчання програмування та алгоритмів.

Python IDLE виявляється досить потужним інструментом. Його простота, зручність та широкий функціонал роблять його привабливим вибором для розробки програмного забезпечення для цієї конкретної задачі. Інтеграція з іншими бібліотеками та інструментами Python дає можливість розширити можливості IDLE та створити потужні інструменти.

2.3 Моделювання вимог до програмного забезпечення

Інструментом моделювання, що використовується для зображення взаємодії між акторами (користувачами або зовнішніми системами) та програмним забезпеченням. Вона дозволяє описати функціональність користувачів та визначити можливі варіанти використання програми.

Які дії можуть виконувати актори та як вони взаємодіють з програмою. Вона включає такі елементи:

-актори: представляють ролі або сутності з програмою. Це можуть бути користувачі.

-взаємодія: показує, як актори взаємодіють з програмою шляхом виклику різних варіантів використання.

Рис. 2.1 Діаграма варіантів використання

Для вилучення цінної інформації з комерційних документів за допомогою мови програмування Python. Дана діаграма показує можливості користувача та перелік функцій, покладених на модуль обробки тексту, що описані вище.

2.4 Розробка класів

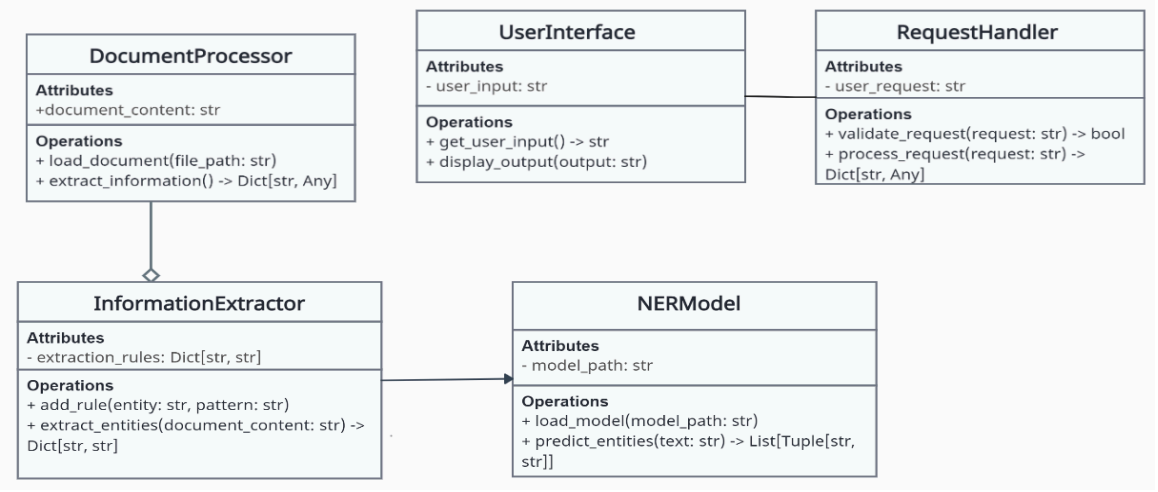


Рис. 2.2 Діаграма класів

Діаграма компонентів зображає архітектуру програми для вилучення цінної інформації з комерційних документів, показуючи взаємодію між основними компонентами. Ось опис кожного з них та їх взаємодію:

Компоненти та їх функціональність:

1. User Interface (Інтерфейс користувача)

   -Призначення: Здійснює взаємодію з користувачем.

   -Функції:

   -Відображає інформацію

   -Приймає введення від користувача

   -Показує результати

   -Взаємодія: Спілкується з Request Handler, приймаючи введення від користувача та відображаючи результати обробки.

2. Request Handler (Обробник запитів)

   -Призначення: Управляє запитами від користувача.

   -Функції:

   -Перевіряє запити

   -Направляє запити на обробку

   -Збирає результати

   -Взаємодія: Спілкується з User Interface, Document Processor, та Information Extractor, забезпечуючи правильну обробку запитів і передачу результатів.

3. Document Processor (Обробник документів)

   -Призначення: Обробляє документи для вилучення текстового вмісту.

   -Функції:

   -Зчитує документи

   -Витягує текстовий вміст

   -Взаємодія: Приймає документи від Request Handler та передає текстовий вміст до Information Extractor.

4. Information Extractor (Витягувач інформації)

-Призначення: Застосовує правила вилучення для ідентифікації сутностей у тексті.

-Функції:

-Застосовує правила вилучення

-Ідентифікує сутності

-Взаємодія: Отримує текстовий вміст від Document Processor та передає ідентифіковані сутності до Request Handler. Також взаємодіє з NER Model для ідентифікації сутностей.

5. NER Model (Модель NER)

-Призначення: Завантажує та використовує попередньо навчені моделі для розпізнавання іменованих сутностей.

-Функції:

-Завантажує попередньо навчені моделі NER

-Передбачає сутності у тексті

-Взаємодія: Отримує текст від Information Extractor та передає назад передбачені сутності.

Взаємодія між компонентами:

-User Interface <-> Request Handler: Інтерфейс користувача передає введення від користувача до обробника запитів і отримує результати для відображення.

-Request Handler <-> Document Processor: Обробник запитів передає запити до обробника документів для зчитування та вилучення тексту.

-Document Processor <-> Information Extractor: Після вилучення тексту, обробник документів передає його до витягувача інформації для подальшого аналізу.

-Information Extractor <-> NER Model: Витягувач інформації взаємодіє з моделлю NER для передбачення та ідентифікації сутностей у тексті.

-Information Extractor <-> Request Handler: Витягувач інформації передає результати аналізу (ідентифіковані сутності) обробнику запитів, який збирає всі результати для передачі інтерфейсу користувача.

2.5 Блок Схеми

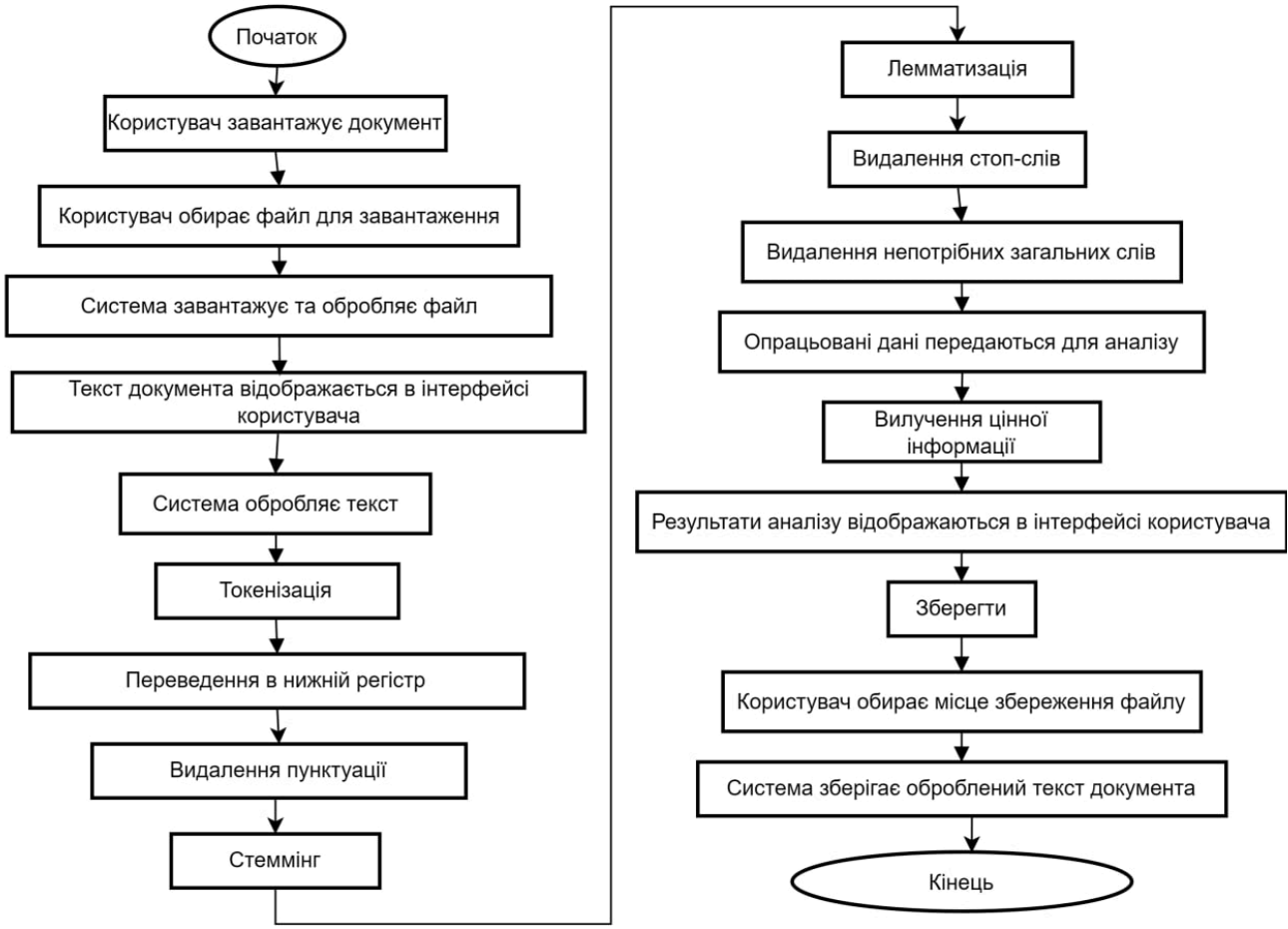


Рис. 2.3 Блок схема

2.6 Процес розробки програмного забезпечення

Процес розробки програмного забезпечення - це комплексний набір дій і етапів, спрямованих на створення програмних продуктів з визначеними вимогами та функціональністю. Зазвичай цей процес включає такі етапи:

-аналіз вимог - це початковий етап, під час якого визначаються потреби користувачів та функціональні вимоги до програмного продукту.

-проектування - архітектура системи, вибираються технології та інструменти, які будуть використовуватися для реалізації програми.

-розробка - це фактична реалізація програмного продукту за допомогою програмування. Розробники пишуть код, тестують його та виправляють помилки.

-тестування - після завершення розробки програма піддається різним видам тестів для перевірки.

Після успішного проходження тестування програма готується до впровадження в робоче середовище. Цей етап може включати інсталяцію програмного забезпечення на сервери або комп'ютери користувачів.

Підтримка та обслуговування - після впровадження програмного продукту в експлуатацію важливо забезпечити його підтримку та обслуговування, включаючи виправлення помилок, оновлення та підтримку користувачів.

Технічна підтримка - окрім розробки нового функціоналу, забезпечення безперебійної роботи. Це включає в себе відповіді на запитання користувачів, вирішення проблем, пов'язаних з використанням програми, а також підтримку постачальників та інших зацікавлених сторін.

Майбутній розвиток - після випуску першої версії програмного продукту, команда розробників може продовжувати працювати над його вдосконаленням та розвитком. Це може включати в себе випуск оновлень з новими функціями, виправлення виявлених помилок, адаптацію до змін у вимогах користувачів та технологій, а також підтримку нових платформ і пристроїв.

Оцінка та звітність під час всього процесу розробки важливо проводити оцінку результатів роботи, а також складати звіти про прогрес і досягнення. Це допомагає виявляти проблеми та вдосконалювати процес розробки, а також забезпечує прозорість і співпрацю всіх учасників проекту.

Забезпечення якості - один з ключових аспектів розробки програмного забезпечення - це забезпечення його якості. Це включає в себе проведення різноманітних видів тестування, таких як модульне тестування, інтеграційне тестування, системне тестування, а також тестування в реальних умовах або

бета-тестування. Крім того, важливо використовувати методи та практики розробки якості, такі як Continuous Integration (CI) та Continuous Deployment (CD), щоб забезпечити автоматизацію процесів тестування та доставки програмного забезпечення.

Управління змінами та конфігурацією - у процесі розробки програмного забезпечення можуть виникати зміни, які впливають на код, документацію, архітектуру та інші аспекти проекту. Тому важливо вести систематичний контроль змін та конфігурацію, щоб забезпечити стабільність та надійність програми. Це включає в себе використання систем контролю версій, таких як Git, а також засобів для автоматизації управління змінами та конфігурацією.

Безпека та конфіденційність - о скільки програмне забезпечення все частіше використовується для обробки конфіденційної інформації та здійснення важливих операцій, безпека стає критично важливою. У процесі розробки необхідно приділяти увагу заходам безпеки, таким як захист від атак, шифрування даних, аутентифікація та авторизація користувачів, а також регулярні аудити безпеки.

Документування та навчання - для забезпечення ефективної підтримки та розвитку програмного забезпечення важливо створювати якісну документацію, яка описує функціональність, архітектуру, API та інші аспекти програми. Крім того, необхідно забезпечити навчання для користувачів та розробників, щоб забезпечити правильне використання та розвиток програмного забезпечення.

У процесі розробки програмного забезпечення важливо враховувати багато аспектів, починаючи від аналізу вимог і закінчуючи підтримкою та майбутнім розвитком. Один з ключових аспектів - це забезпечення якості, що включає в себе різноманітні види тестування та використання методів розробки якості.

Далі, важливо здійснювати управління змінами та конфігурацією, щоб забезпечити стабільність та надійність програми. З урахуванням росту кількості кіберзагроз важливо також приділяти увагу безпеці та конфіденційності даних. Не

менш важливою є створення якісної документації та навчання користувачів, що сприяє ефективній підтримці та розвитку програмного забезпечення.

Крім того, необхідно забезпечити ефективну технічну підтримку користувачів, яка включає в себе вчасне вирішення їхніх запитів та проблем, надання необхідної допомоги та консультацій.

Технічна підтримка відіграє ключову роль у забезпеченні задоволення користувачів та підтриманні репутації продукту на ринку. Щодо майбутнього розвитку програмного забезпечення, важливо враховувати технологічні тренди та потреби ринку.

Наприклад, інтеграція штучного інтелекту, розширення функціональності, оптимізація продуктивності та підвищення безпеки можуть бути важливими напрямками розвитку. Також важливо враховувати зміни в законодавстві та вимоги щодо захисту даних, щоб відповідати сучасним стандартам і запобігати можливим проблемам у майбутньому.

Крім цього, розвиток програмного забезпечення може включати у себе розширення підтримуваних платформ і пристроїв, щоб забезпечити доступність продукту для широкого кола користувачів. Регулярні оновлення та вдосконалення також є важливим етапом у процесі розвитку програмного забезпечення, оскільки вони дозволяють вирішувати виявлені помилки, вдосконалювати функціональність та відповідати змінюючимся потребам користувачів і технологічному середовищу.

Таким чином, стратегія технічної підтримки та майбутнього розвитку важлива для забезпечення успішності програмного забезпечення на ринку та задоволення потреб користувачів.

Додатково, ефективна технічна підтримка може включати в себе створення документації, навчальних матеріалів та онлайн-ресурсів для користувачів, щоб забезпечити їм необхідну інформацію та допомогу у використанні програмного забезпечення.

Також важливою складовою є активна взаємодія з користувачами через форуми, чати підтримки та системи відслідковування помилок, щоб оперативно

вирішувати їхні проблеми та відповідати на їхні запитання. Крім цього, розробники можуть регулярно проводити опитування користувачів та аналізувати їхні відгуки, щоб зрозуміти їхні потреби та пропозиції щодо покращень програмного забезпечення. Такий підхід дозволяє зберігати високий рівень задоволеності користувачів та забезпечувати їм продуктивну роботу з програмним забезпеченням.

Крім того, для забезпечення майбутнього розвитку програмного забезпечення необхідно постійно вдосконалювати його функціональність та додавати нові можливості відповідно до змінних потреб користувачів і ринкових умов.

Це може включати розробку нових модулів, підтримку нових технологій та стандартів, а також виправлення виявлених помилок та усунення існуючих недоліків. Постійне оновлення програмного забезпечення дозволяє зберігати його конкурентоспроможність на ринку та задовольняти зростаючі потреби користувачів.

Для цього можуть використовуватися методи Agile-розробки, ітераційний підхід до розробки програмного забезпечення, а також відкритий обмін ідеями та зворотніми зв'язками з користувачами та зацікавленими сторонами. Такий підхід дозволяє забезпечити постійне вдосконалення програмного забезпечення і відповідати на змінні потреби користувачів та вимоги ринку.

Продовженням цього процесу є внесення коригувань і вдосконалень на основі зворотного зв'язку від користувачів та виявлених проблем у реальних умовах експлуатації програмного продукту. Також важливим аспектом майбутнього розвитку є забезпечення сумісності програмного забезпечення з новими версіями операційних систем та іншого важливого програмного забезпечення, що використовується.

Враховуючи постійні зміни в технологічному ландшафті та вимоги ринку, програмне забезпечення повинно бути готове до швидкого адаптування та реагування на нові виклики і можливості.

2.7 Технічна підтримка та майбутній розвиток

Підтримка та розвиток програмного забезпечення - це ключові аспекти, що визначають тривалість його життєвого циклу та задоволення потреб користувачів у майбутньому. Щоб забезпечити ефективну технічну підтримку та сприяти майбутньому розвитку продукту, необхідно ретельно проробити кілька аспектів.

Технічна підтримка - визначення чітких процедур і механізмів технічної підтримки, включаючи канали зв'язку з користувачами, системи відстеження проблем та швидкий відгук на запити користувачів. Потрібно забезпечити, щоб проблеми були вирішені оперативно та ефективно, що сприятиме задоволенню користувачів та покращенню репутації продукту.

Оновлення та вдосконалення - систематичне планування оновлень та вдосконалень програмного забезпечення на основі зворотного зв'язку від користувачів та аналізу ринкових тенденцій. Регулярні випуски нових версій з новими функціями та виправленнями допоможуть зберігати конкурентоспроможність продукту і привертати нових користувачів.

Взаємодія зі спільнотою - створення активної спільноти користувачів та розробників навколо продукту, яка сприятиме обміну ідеями, відкритому обговоренню проблем та спільному розв'язанню завдань. Регулярні вебінари, форуми та події спільноти допоможуть залучати увагу до продукту та підтримувати зацікавленість користувачів.

Аналіз ринку та конкурентоспроможність - постійне відстеження змін на ринку та дослідження потреб користувачів допоможе визначити напрямки майбутнього розвитку продукту. Важливо проводити порівняльний аналіз з конкурентами, щоб зрозуміти їхні переваги та недоліки та використовувати цю інформацію для вдосконалення свого продукту.

Документація та навчання - розробка і підтримка документації для користувачів та розробників, яка допоможе їм краще розуміти функціонал продукту та використовувати його на повну потужність. Організація навчальних

курсів та вебінарів з метою підвищення кваліфікації користувачів також буде корисною.

Безпека та стабільність - постійна робота над підвищенням безпеки та стабільності програмного забезпечення, виявлення та виправлення вразливостей, а також застосування найкращих практик у сфері кібербезпеки. Забезпечення стійкості та надійності продукту буде визначальним для збереження довіри користувачів та успішного конкурування на ринку.

Моніторинг та звітність - встановлення системи моніторингу за рівнем задоволення користувачів, ефективністю підтримки та іншими ключовими показниками продуктивності. Регулярна підготовка звітів та аналіз отриманих даних дозволить вчасно виявляти проблеми та приймати відповідні заходи.

Планування майбутніх версій - розробка стратегії розвитку та планування майбутніх версій продукту на основі потреб користувачів, технологічних тенденцій та конкурентного аналізу. Чітке визначення функціональних вимог та реалізація планів розвитку допоможе зберігати конкурентоспроможність продукту в майбутньому.

Експерименти та інновації - сприяння культурі експериментів та інновацій в команді розробників для постійного вдосконалення продукту. Стимулювання та підтримка ідей та креативних підходів сприятиме впровадженню новаторських функцій та рішень.

Гнучкість та адаптивність - забезпечення гнучкості та адаптивності продукту до змін у вимогах користувачів та ринкових умов. Швидка реакція на зміни допоможе зберегти конкурентні переваги та забезпечити відповідність продукту сучасним вимогам.

Безпека та конфіденційність - забезпечення високого рівня безпеки та захисту конфіденційності даних користувачів. Постійне вдосконалення заходів забезпечення безпеки та впровадження новітніх технологій шифрування допоможе уникнути порушень безпеки та зберегти довіру клієнтів.

Автоматизація та оптимізація процесів - впровадження автоматизованих систем управління, тестування та моніторингу для зменшення людського

втручання та підвищення ефективності процесів. Використання інструментів автоматизації допоможе швидко реагувати на проблеми та знизити ризик помилок.

Скалабельність та продуктивність - забезпечення можливості горизонтального та вертикального масштабування системи для підтримки зростаючої кількості користувачів та обсягу даних. Оптимізація продуктивності допоможе забезпечити стабільну роботу системи навіть при великому навантаженні.

Стратегія врядування ризиками - розробка стратегії управління ризиками, включаючи виявлення, оцінку та зменшення ризиків, що можуть вплинути на роботу продукту. Ефективне врядування ризиками дозволить уникнути потенційних проблем та зберегти стабільність системи.

Неперервна підтримка та оновлення - забезпечення неперервної підтримки клієнтів та регулярне оновлення програмного забезпечення з метою виправлення помилок, усунення вразливостей та впровадження нових функцій. Постійне вдосконалення продукту дозволить зберегти його актуальність та конкурентоспроможність на ринку.

Реагування на зміни вимог - важливо мати механізми для ефективного реагування на зміни вимог з боку користувачів або ринкових умов. Гнучкість та швидкість адаптації допоможуть зберегти конкурентоспроможність продукту та задовольнити потреби змінюючогося ринку.

Інтеграція з іншими системами - планування та реалізація механізмів для ефективної інтеграції програмного забезпечення з іншими системами та сервісами. Це дозволить розширити функціональність продукту та покращити його взаємодію з іншими екосистемами.

Моніторинг та аналіз продуктивності - постійний моніторинг та аналіз продуктивності програмного забезпечення з метою виявлення можливих проблем та вдосконалення ефективності роботи системи. Це дозволить оперативно вживати заходів для усунення проблем та оптимізації ресурсів.

Експертна підтримка та консультації - забезпечення доступу до експертної підтримки та консультацій для користувачів у разі виникнення складних технічних

питань або проблем. Це допоможе клієнтам швидше та ефективніше вирішувати проблеми та отримувати необхідну допомогу.

Стратегія розвитку продукту - розробка чіткої стратегії для майбутнього розвитку програмного забезпечення, включаючи плани по впровадженню нових функцій, покращень та інновацій. Це дозволить продукту залишатися актуальним та конкурентоспроможним у змінному технологічному середовищі.

Загальна технічна підтримка включає в себе низку процесів, таких як управління помилками та усунення несправностей, підтримка безпеки та захисту даних, а також надання оновлень та патчів для виправлення виявлених уразливостей. Важливо мати ефективні канали зв'язку з користувачами, де вони можуть повідомляти про проблеми та отримувати швидку допомогу в їх вирішенні. Також необхідно мати процеси реагування на екстрені ситуації та прийняття невідкладних заходів для відновлення працездатності системи у разі виникнення серйозних проблем або атак.

У контексті майбутнього розвитку програмного забезпечення важливо визначити напрямки розвитку та стратегії впровадження нових технологій. Це може включати дослідження нових підходів та методів розробки, функціональності, а також аналіз та використання даних для прийняття стратегічних рішень. Крім того, важливо стежити за технологічними тенденціями та впроваджувати інноваційні рішення, які дозволять продукту залишатися конкурентоспроможним у швидкозмінному інформаційному середовищі.

Додатковою складовою ефективної технічної підтримки є надання користувачам доступу до документації, навчальних матеріалів та онлайн-ресурсів, які допоможуть їм зрозуміти та використовувати програмне забезпечення більш ефективно. Важливо також регулярно оновлювати та покращувати документацію, щоб вона завжди відображала поточний стан продукту та надавала користувачам необхідну інформацію для розв'язання їх питань та проблем.

Щодо майбутнього розвитку програмного забезпечення, важливо також враховувати зміни у вимогах користувачів та ринкових умовах. Це може включати розширення функціональності, оптимізацію продуктивності та швидкодії, а також

адаптацію до нових технологічних трендів і вимог щодо безпеки та захисту даних. Налагодження ефективних процесів збору та аналізу зворотного зв'язку від користувачів допоможе ідентифікувати потреби та пріоритети для майбутнього розвитку продукту. Також важливо бути готовим до змін і вдосконалень, реагуючи на змінні умови ринку та вимоги користувачів для забезпечення стабільного та успішного майбутнього розвитку програмного забезпечення.

Крім того, для успішного майбутнього розвитку важливо підтримувати тісний контакт зі спільнотою користувачів та реагувати на їхні потреби та пропозиції. Це може включати організацію форумів, вебінарів або інших подій, де користувачі можуть обговорювати свої ідеї та відгуки щодо програмного забезпечення. Також важливо розглядати можливості співпраці з іншими компаніями чи розробниками для розширення функціональності та покращення якості продукту.

У контексті майбутнього розвитку програмного забезпечення важливо також враховувати зміни в індустрії та вимоги регуляторів. Наприклад, зростання уваги до кібербезпеки може вимагати впровадження додаткових заходів захисту даних та персональної інформації користувачів. Також можуть змінюватися стандарти та технологічні вимоги, що вимагатимуть адаптації програмного забезпечення до нових умов.

Загалом, успішний майбутній розвиток програмного забезпечення вимагає постійного моніторингу змін у вимогах ринку, відгуків користувачів та технологічних тенденцій, а також гнучкості та готовності до впровадження змін для забезпечення високої якості та конкурентоспроможності продукту.

Крім того, у контексті технічної підтримки важливо забезпечити ефективну систему зворотнього зв'язку для користувачів, де вони можуть повідомляти про проблеми, отримувати консультації та вирішувати технічні питання. Це може бути здійснено через електронну пошту, онлайн-чат, телефонну лінію підтримки або систему квитків для служби підтримки. Важливо забезпечити оперативну відповідь на запити користувачів та регулярно оновлювати їх щодо стану вирішення їхніх проблем.

Також важливо мати документовану базу знань або систему самообслуговування, де користувачі можуть знаходити відповіді на популярні запитання та інструкції щодо використання програмного забезпечення. Це дозволить зменшити навантаження на службу підтримки та забезпечити користувачам швидкий доступ до необхідної інформації.

Крім того, для забезпечення успішного майбутнього розвитку необхідно враховувати технологічні та індустріальні тенденції, вивчати їх потенційні застосування для покращення функціональності та безпеки програмного забезпечення. Також важливо активно реагувати на зміни в індустрії та конкуренцію, щоб забезпечити конкурентоспроможність продукту на ринку.

3 ТЕСТУВАННЯ СИСТЕМИ

3.1 Опис функціонування

Інтерфейс головного вікна представлено на рис. 3.1.

3 кнопки “Відкрити файл” “Опрацювати текст” “Зберегти результат” також можливість вставити текст з буфер обміну, або самостійно написати текст запиту.

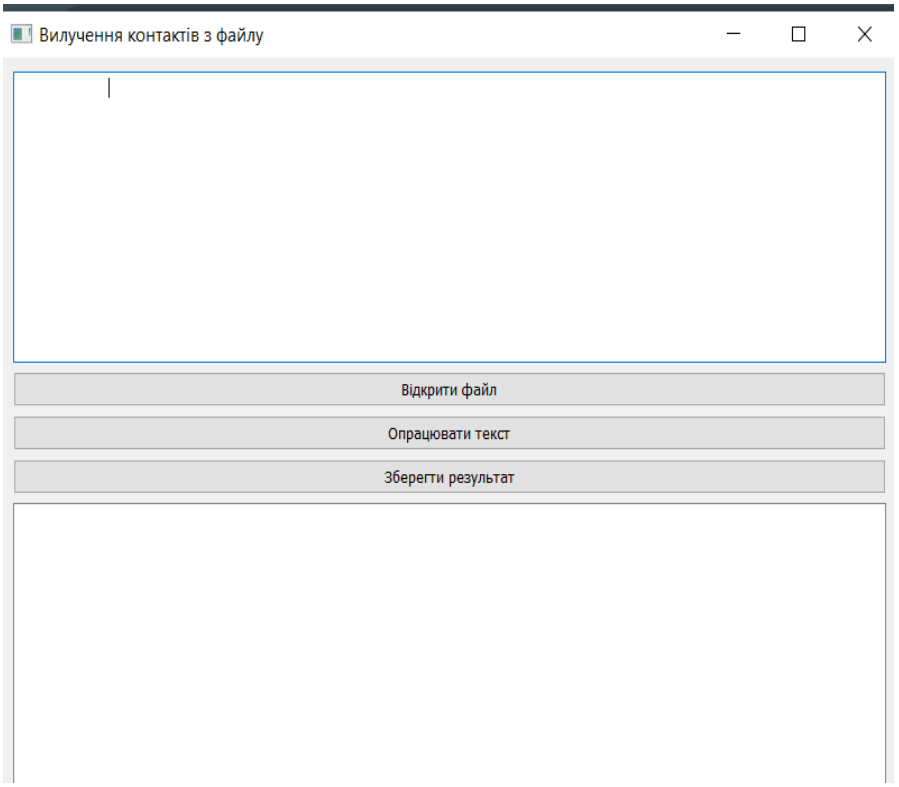


Рис. 3.1 Інтерфейс головного вікна

Для прикладу напишемо власноруч текст в системі - “Доброго дня, шановний Андрій Шевченко 19.09.2024 за адресою вулиця Миколи Алі 19,наш офіс, напишіть на електронну адресу johnsmith@gmail.com”

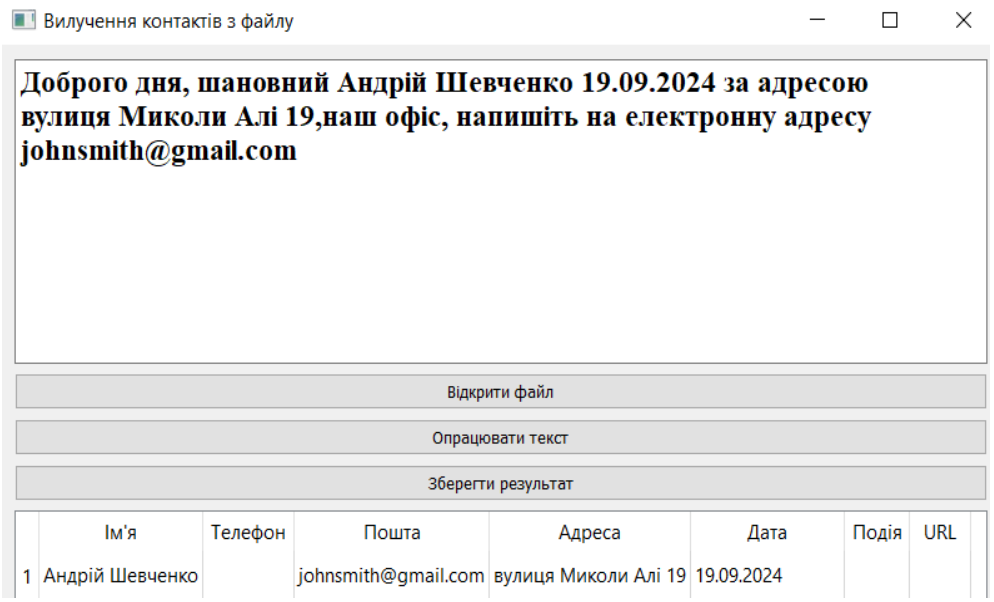


Рис. 3.2 Відповідь користувача

Щоб опрацювати файл потрібно вибрати потрібний файл, текстового формату TXT.

На рис. 3.3 наведено відповідь з файлом.

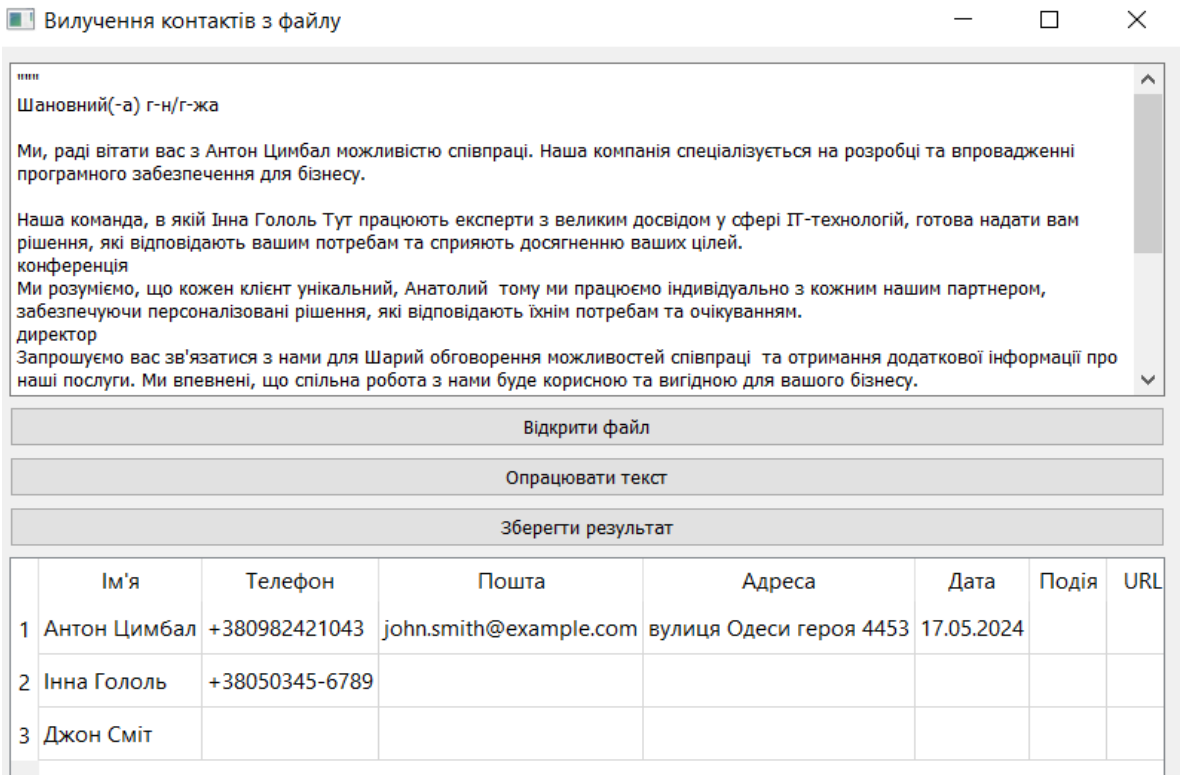


Рис. 3.3 Відповідь з файлом

3.2 Підготовка тестового середовища

Тестування програмного забезпечення - це процес перевірки відповідності вимог,

забезпеченні якості та надійності розробленого продукту. Тестування включає перевірку відповідності функціональності програмного забезпечення вимогам, виявлення можливих помилок та забезпечення стабільної роботи системи. Розглянемо основні етапи та методи спираючись на попередньо описані процеси розробки.

Етапи тестування:

планування тестування:

-визначення обсягу тестування, цілей та критеріїв успішності.

-підготовка тестового середовища та вибір необхідних інструментів для автоматизованого та ручного тестування.

-розробка тестових сценаріїв і тест-кейсів.

розробка тестових сценаріїв:

-складання детальних сценаріїв для перевірки функціональності кожного модуля програмного забезпечення.

-розробка позитивних та негативних тест-кейсів для перевірки різних аспектів роботи системи.

виконання тестів:

-запуск тестових сценаріїв вручну або за допомогою автоматизованих інструментів.

-виконання модульних, інтеграційних, системних та приймальних тестів для перевірки окремих компонентів та їх взаємодії.

аналіз результатів:

-оцінка результатів тестування, виявлення помилок та несправностей.

-документування знайдених дефектів та їх класифікація за пріоритетом.

виправлення помилок:

-внесення змін до коду для усунення виявлених помилок.

-перевірка виправлень шляхом повторного тестування.

реліз та підтримка:

-остаточна перевірка програмного забезпечення перед випуском.

-моніторинг та підтримка програмного забезпечення після релізу для виявлення та виправлення нових помилок.

методи тестування:

-системи на відповідність специфікаціям.

-використання бібліотек для модульного тестування, таких як unittest або pytest в Python.

-тестування інтерфейсів та даних, що передаються між модулями.

системне тестування:

-повна перевірка всієї системи

-на відповідність вимогам кінцевого користувача.

-виконується з залученням представників замовника або кінцевих користувачів.

Тестування з використанням тест-кейсів:

-ID тест-кейсу: Унікальний ідентифікатор для кожного тесту.

-опис: Короткий опис функціональності, що перевіряється.

-передумови: Умови, які повинні бути виконані перед виконанням тесту.

-кроки виконання: Детальний перелік дій, які необхідно виконати.

-очікуваний результат: Результат, який повинен бути отриманий при правильній роботі системи.

-фактичний результат: Результат, отриманий після виконання тесту.

-статус: Відмітка про успішне або неуспішне проходження тесту.

Після виконання всіх запланованих тест-кейсів було отримано наступні результати:

-успішно пройдені тести: Всі тести пройдені успішно, що підтверджує правильність реалізації функціональності.

-виявлені дефекти: В процесі тестування були виявлені та виправлені декілька незначних дефектів.

-оцінка стабільності: Програмне забезпечення показало високу стабільність та надійність при виконанні всіх тестових сценаріїв.

Таким чином, проведене тестування підтверджує готовність програмного забезпечення до використання в реальних умовах та взаємодії з кінцевими користувачами.

Крім основних етапів та методів тестування, варто розглянути додаткові аспекти, що допоможуть забезпечити ще вищий рівень якості програмного забезпечення. До таких аспектів можна віднести автоматизацію тестування, безперервну інтеграцію та безперервне тестування, а також забезпечення тестового покриття.

Автоматизація тестування дозволяє значно прискорити процес перевірки програмного забезпечення, особливо при повторюваних завданнях або великих обсягах тестування. Вона включає створення скриптів для автоматичного виконання тестових сценаріїв та порівняння результатів з очікуваними значеннями. Серед популярних інструментів для автоматизації тестування можна виділити Selenium для тестування веб-застосунків, pytest для автоматизації тестування Python, а також JUnit для тестування Java-застосунків.

Впровадження практик безперервної інтеграції та безперервного тестування дозволяє забезпечити стабільність та якість програмного забезпечення на всіх етапах розробки.

Система CI/CT автоматично виконує всі необхідні тести щоразу, коли відбувається зміна у коді (наприклад, при злитті змін з гілками розробки). Це дозволяє вчасно виявляти та виправляти помилки, знижуючи ризик виникнення критичних дефектів на етапі впровадження.

Тестове покриття – це показник, який визначає, яка частина коду була перевірена тестами. Високий рівень тестового покриття допомагає впевнитися, що більшість можливих помилок були виявлені на ранніх етапах розробки. Для

оцінки та підвищення тестового покриття можна використовувати спеціальні інструменти, такі як Coverage.py для Python або JaCoCo для Java.

Важливо не лише досягти високого відсотка покриття, але й забезпечити якість самих тестів, що включає створення різноманітних сценаріїв та виявлення граничних випадків.

Регресійне тестування – це процес повторного тестування програмного забезпечення після внесення змін, щоб переконатися, що нові зміни не викликали помилок у раніше перевіреній функціональності. Це критично важливо для підтримки стабільності системи, особливо при внесенні значних змін або додаванні нових функцій. Автоматизовані тести значно спрощують проведення регресійного тестування, дозволяючи швидко перевіряти великий обсяг сценаріїв.

Окрім функціонального тестування, важливо також перевіряти продуктивність та поведінку системи під навантаженням. Це допомагає визначити, як програмне забезпечення працюватиме у реальних умовах з великим числом користувачів або великим обсягом даних. Для цього використовуються спеціальні інструменти, такі як JMeter або LoadRunner, які дозволяють моделювати навантаження та аналізувати результати.

Безпекове тестування є невід'ємною частиною процесу забезпечення якості програмного забезпечення, особливо для систем, що обробляють конфіденційну або критичну інформацію. Це включає перевірку системи на наявність вразливостей, таких як SQL-ін'єкції, XSS (міжсайтовий скриптинг) та допомагає виявляти та усувати потенційні ризики безпеки.

Зручність та інтуїтивність користувацького інтерфейсу відіграють важливу роль у загальному враженні від програмного забезпечення. Тестування UI включає перевірку взаємодії користувача з системою, забезпечення відповідності інтерфейсу вимогам до дизайну та зручності використання.

Це може бути досягнуто як вручну, так і за допомогою автоматизованих інструментів, таких як Selenium або Appium для мобільних застосунків.

Загалом, тестування програмного забезпечення – це аспекти функціональності, продуктивності, безпеки та зручності використання. Ретельне

планування, використання сучасних інструментів та методологій, а також безперервне вдосконалення процесів тестування допомагають створити надійне та якісне програмне забезпечення, що відповідає всім вимогам користувачів та замовників.

Невід'ємною частиною процесу тестування є документування результатів. Це включає створення детальних звітів про проведені тести, виявлені помилки та їх виправлення. Документація забезпечує прозорість процесу розробки, полегшує комунікацію між командами та служить як історичний запис для подальшого аналізу і вдосконалення продукту.

Тестування із залученням реальних користувачів є важливим етапом, який допомагає оцінити, як продукт буде використовуватися у реальних умовах. Це може включати альфа- та бета-тестування:

-альфа-тестування: проводиться всередині організації розробника з метою виявлення очевидних дефектів на ранніх етапах. В цьому процесі можуть брати участь працівники компанії або спеціально запрошені користувачі.

-бета-тестування: здійснюється за участі зовнішніх користувачів, які використовують продукт у своїх реальних умовах. Це дозволяє отримати цінний зворотний зв'язок, виявити проблеми, які не були помічені на попередніх етапах тестування, та оцінити задоволеність користувачів.

Тестування в різних середовищах допомагає забезпечити, що продукт буде стабільно працювати у різних конфігураціях системи. Це включає тестування в середовищах, що імітують різні операційні системи, браузери, мобільні пристрої тощо.

Використання віртуальних машин та контейнерів (наприклад, Docker) дозволяє швидко створювати та налаштовувати різні тестові середовища.

Після завершення кожного циклу тестування важливо проводити аналіз результатів та визначати, що можна вдосконалити в майбутньому. Це може включати:

-аналіз дефектів: Розбір виявлених дефектів, їхніх причин та шляхів усунення, щоб уникнути подібних помилок у майбутньому.

-навчання команди: Проведення тренінгів та семінарів для команди тестувальників, обмін знаннями та досвідом між учасниками проекту.

Важливо, щоб тестувальники були залучені до процесу з самого початку, мали доступ до вимог та технічної документації, а також брали участь у плануванні та оцінці ризиків.

В таблиці 3.1 наведено результати тестування.

Таблиця 3.1

Перелік очікувань користувачів

Тест-кейс	Очікуваний результат	Результат тестування
Натиснути кнопку «Відкрити файл», вибрати файл	Вибраний файл завантажується в текстове поле	Passed
Натиснути кнопку «Опрацювати текст»	Дані з текстового поля обробляються і заповнюють таблицю	Passed
Натиснути кнопку «Зберегти результат» та вибрати місце збереження	Таблиця зберігається у вибране місце як текстовий файл	Passed
Ввести текст з різними типами даних та натиснути кнопку «Опрацювати текст»	Дані коректно обробляються та відображаються у відповідних колонках таблиці	Passed
Натиснути кнопку «Опрацювати текст» декілька разів поспіль	Таблиця оновлюється без дублювання даних	Passed
Натиснути кнопку «Зберегти результат» без попереднього завантаження файлу та обробки тексту	Відображається повідомлення про помилку або збереження не відбувається	Passed
Ввести текст з URL, подіями, компанією	Відповідні дані коректно відображаються у таблиці	Passed

ВИСНОВКИ

В процесі виконання кваліфікаційної роботи, було досягнуто наступних результатів:

1. Проведено теоретичні дослідження в сфері програмного забезпечення, щоб зрозуміти основні принципи та підходи, які використовуються в сучасних програмних засобах.

2. Проаналізовано теоретичні методи, що сприяють вилученню цінної інформації з комерційних документів за допомогою Python.

3. Проаналізовано сучасний стан розробки програмного забезпечення для вилучення цінної інформації з комерційних документів.

4. Розроблено функціональні та нефункціональні вимоги до застосунку.

5. Спроектувано архітектуру, визначити класи та методи для створення застосунку, що дозволяє вилучати цінну інформацію з комерційних документів за допомогою мови програмування Python.

6. Проведено модульне та інтеграційне тестування програмного забезпечення.

7. Розроблено користувацький інтерфейс для взаємодії студентів з програмним забезпеченням.

На основі отриманих даних та досліджень, розроблено застосунок для підтримки самостійної роботи студентів, який враховує всі проаналізовані потреби та можливості, щоб стати ефективним інструментом у навчанні.

8. Робота пройшла апробацію:

Мельник В.В., Аверічев І.М. Програмна реалізація автоматизованого вилучення інформації з використанням мови Python. V Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IOT», 18.04.2024, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С.43-45.

Мельник В.В., Аверічев І.М. Розробка програмного забезпечення для автоматизованого вилучення корисної інформації за допомогою Python. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ», 24.04.2024, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С.48-49.

ПЕРЕЛІК ПОСИЛАНЬ

1. Електронний архів [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%95%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B8%D0%B9_%D0%B0%D1%80%D1%85%D1%96%D0%B2

2. Top 10 Python IDEs [Електронний ресурс]. — Режим доступу : https://intellipaat.com/blog/top-10-python-ides/?US — Дата доступу : січень 2023. — Назва з екрана.

3. A Concise Guide of 10+ Awesome Python Editors and How To Choose Which Editor Suits You The Best: With Pros and Cons.— Режим доступу : https://towardsdatascience.com/a-concise-guide-of-10-awesome-python-editors-and-how-to-choose-which-editor-suits-you-the-best-465c9b232afd.

4. 12 BEST Python IDE & Code Editors For Mac & Windows In 2023 [Електронний ресурс]. — Режим доступу : https://www.softwaretestinghelp.com/python-ide-code-editors

5. PyCharm Features [Електронний ресурс] // JetBrains. – 2023. – Режим доступу до ресурсу: https://www.jetbrains.com/pycharm/features/.

6. The Python Tutorial [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/tutorial/

7. Documentation v6.5.1 >> Introduction [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.riverbankcomputing.com/static/Docs/PyQt6/introduction.html.

8. Utsav M. 10 Top NLP Algorithms [Електронний ресурс] / Mishra Utsav. – 2022. – Режим доступу до ресурсу: https://www.analyticssteps.com/blogs/topnlp-algorithms.

9. PyQt5 vs PyQt6 What are the differences, and is it time to upgrade? [Електронний ресурс] // PythonGUIS. – 2023. – Режим доступу до ресурсу: https://www.pythonguis.com/faq/pyqt5-vs-pyqt6/.

10.Li, H.; Gupta, A.; Zhang, J. [Електронний ресурс] ; Flor, N. Who will use augmented reality? An integrated approach based on text analytics and field survey. Eur. J. Oper. Res. 2020, 281, 502–516.

11.Stanza – A Python NLP Package for Many Human Languages [Електронний ресурс] // Stanza. – 2023. – Режим доступу до ресурсу: https://stanfordnlp.github.io/stanza/.

12. Available Models & Languages [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://stanfordnlp.github.io/stanza/available_models.html.

13. Rothman, D. Transformers for Natural Language Processing: Build Innovative Deep Neural Network Architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and More; Packt Publishing Ltd. Birmingham Mumbai: Birmingham, UK, 2021.

14. Vectorization Techniques in NLP [Guide] [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://neptune.ai/blog/vectorization-techniques-innlp-guide.

15. Marshall C. Named Entity Recognition [Електронний ресурс] / Christopher Marshall. – 2019. – Режим доступу до ресурсу: https://medium.com/mysuperai/what-is-named-entity-recognition-ner-and-howcan-i-use-it-2b68cf6f545d.

ДОДАТОК А
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення для вилучення з комерційних документів цінної для користувача інформації за допомогою Python

Виконав студент 4 курсу
групи ПД-41
Мельник Владислав Вікторович
Керівник роботи
К.е.н., доцент кафедри ІПЗ Аверічев Ігор Миколайович

Київ – 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи:** спрощення процесу вилучення з комерційних документів цінної для користувача інформації за допомогою застосунку мовою Python.
- **Об'єкт дослідження:** процес вилучення з комерційних документів цінної для користувача інформації.
- **Предмет дослідження:** програмне забезпечення для вилучення з комерційних документів цінної для користувача інформації.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Дослідити особливості вирішення задачі вилучення інформації з текстових документів комерційного спрямування.
2. Проаналізувати цифрові рішення для вилучення з комерційних документів цінної для користувача інформації.
3. Визначити функціональні та нефункціональні вимоги для вилучення з комерційних документів цінної для користувача інформації.
4. Спроектувати схему функціонування програми
5. Розробка інструментальних засобів для вирішення задачі вилучення з комерційних документів цінної для користувача інформації.
6. Провести тестування застосунку.

3

АНАЛІЗ АНАЛОГІВ

Функціональна можливість	Textract	Pytesseract	Apache Tika	Rossum	Розроблений застосунок
Вилучення номер телефону	+	+	–	–	+
Вилучення електронної пошти	+	–	+	–	+
Українська локалізація	–	–	–	–	+
Збереження результатів обробки	–	–	–	–	+
Вилучення дат заходів	–	+	+	+	+
Вилучення назв компанії	+	–	+	+	+
Вилучення адрес	–	–	–	–	+
Завантаження файлу TXT	–	–	–	+	+
Користувач повинен робити додаткові налаштування	+	+	+	+	–
Орієнтація на комерційні документи	–	–	–	–	+

4

ВИМОГИ ДО ЗАСТОСУНКУ

Функціональні вимоги:

1. Можливість вилучення наступних видів цінної інформації для користувача:
   - Вилучення номер телефону.
   - Вилучення електронної пошти.
   - Вилучення дат заходів.
   - Вилучення назв компанії.
2. Завантажити дані з файлу.
3. Переглянути результати обробки в застосунку.
4. Зберегти результати обробки в файл.

Нефункціональні вимоги:

1. Підтримка різних ОС (Windows, Mac, Linux).
2. Завантаження форматів документів TXT.
3. Українська локалізація інтерфейсу .

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

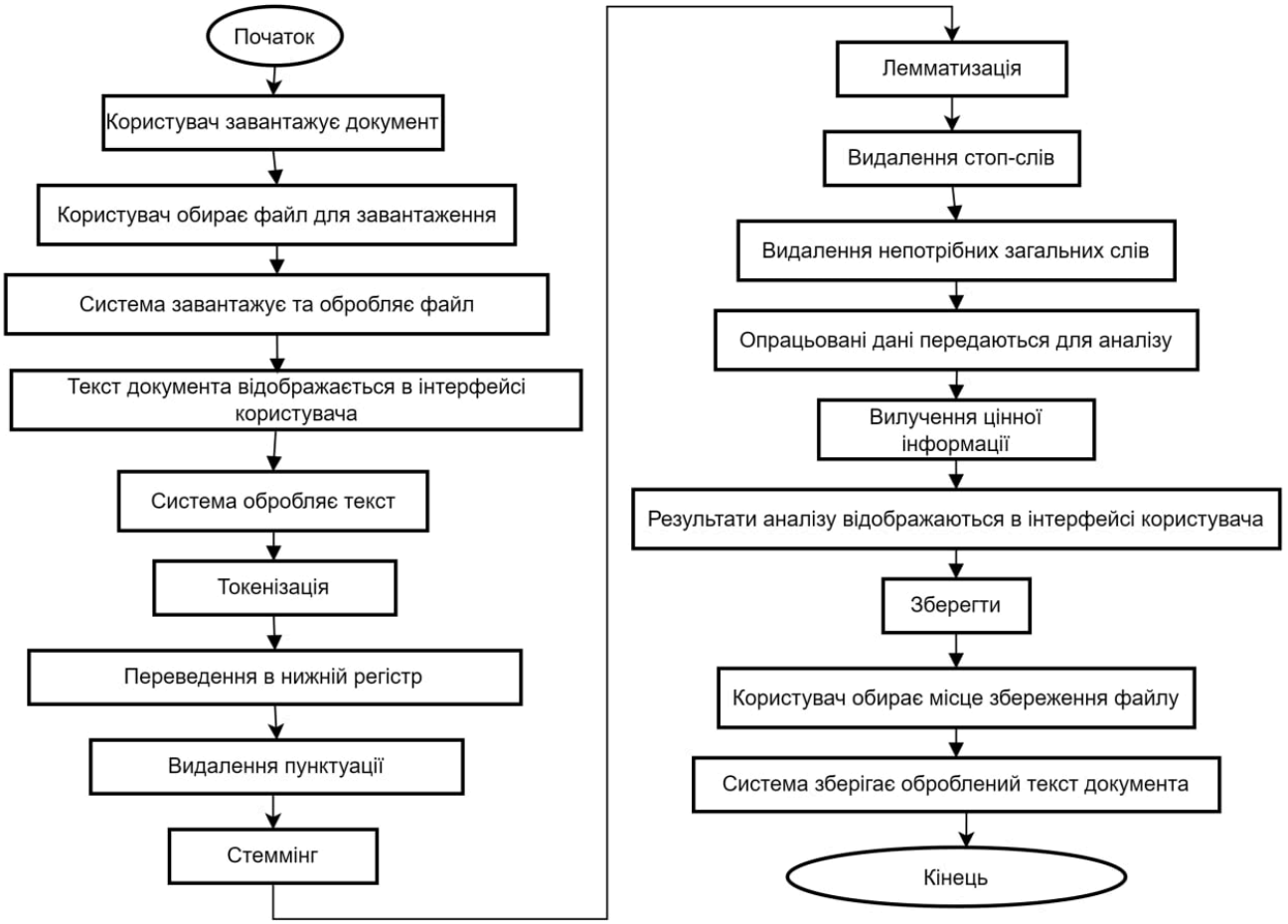
Python

PyQt6

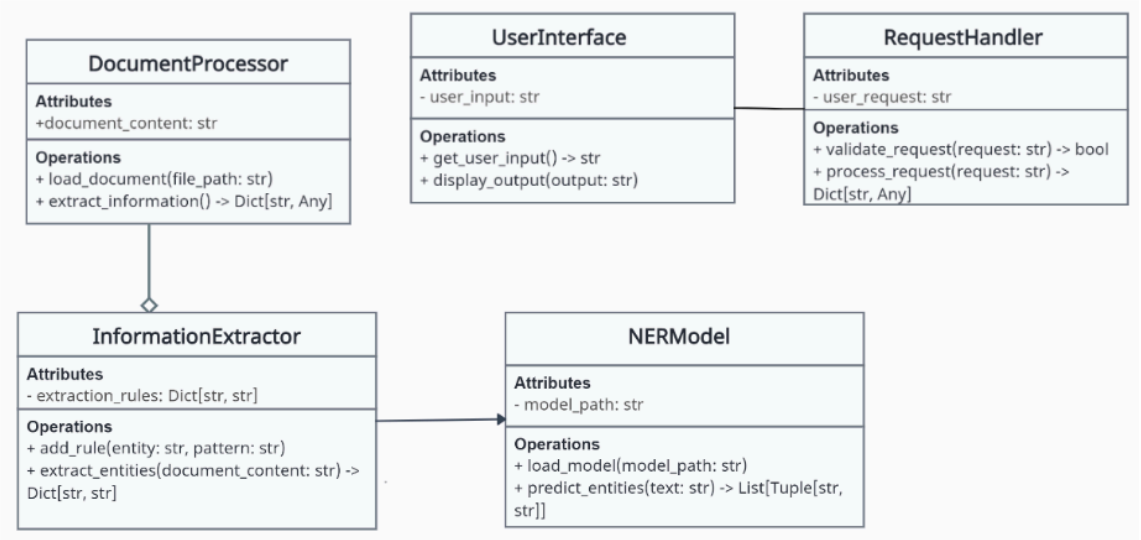
6


Діаграма варіантів використання
Відкрити файл
Надіслати запит на опрацювання
Переглянути вилучену інформацію
Зберегти вилучену інформацію
Користувач
7

Блок Схема

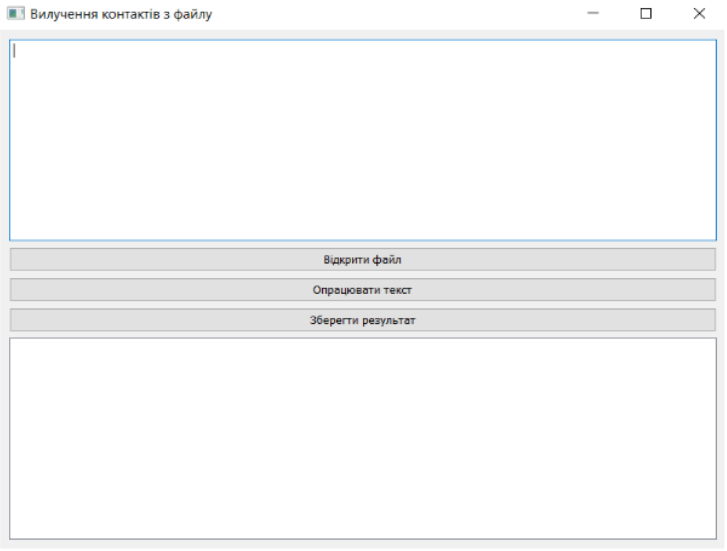

Початок
Користувач завантажує документ
Користувач обирає файл для завантаження
Система завантажує та обробляє файл
Текст документа відображається в інтерфейсі користувача
Система обробляє текст
Токенізація
Переведення в нижній регістр
Видалення пунктуації
Стеммінг
Лемматизація
Видалення стоп-слів
Видалення непотрібних загальних слів
Опрацьовані дані передаються для аналізу
Вилучення цінної інформації
Результати аналізу відображаються в інтерфейсі користувача
Зберегти
Користувач обирає місце збереження файлу
Система зберігає оброблений текст документа
Кінець

Діаграма класів



9

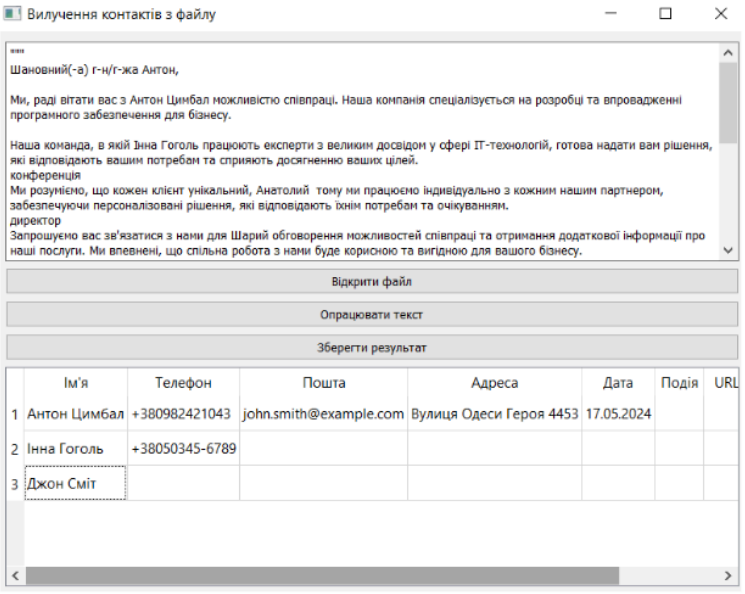
ЕКРАННІ ФОРМИ



Головний екран застосунку

10

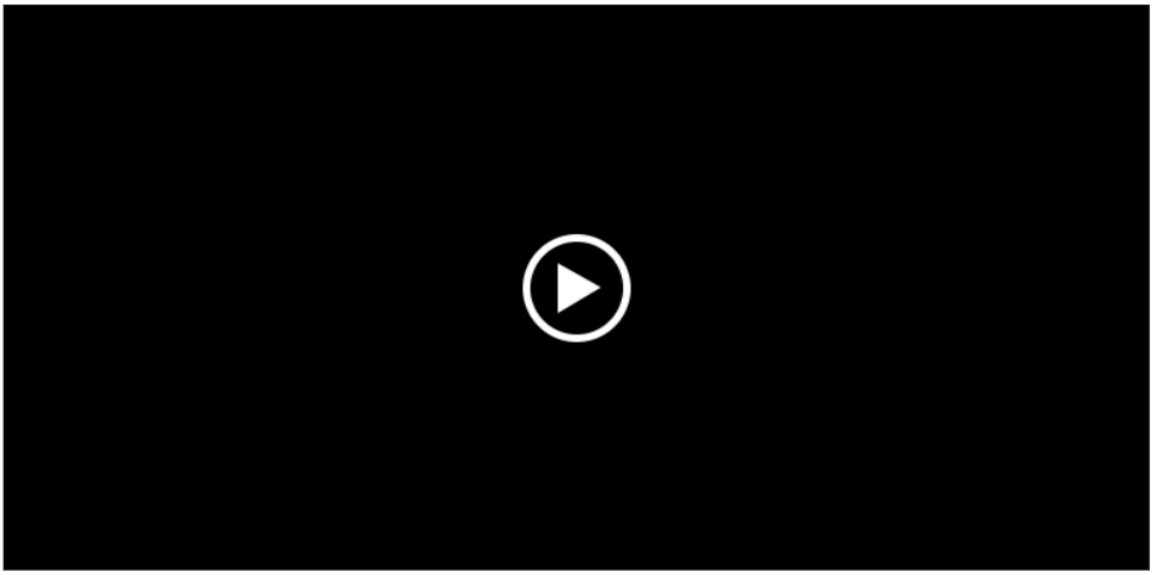
ЕКРАННІ ФОРМИ



Екран виводу результатів

11

Демонстрація застосунку



12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Аверічев І.М., Мельник В.В. Програмна реалізація автоматизованого вилучення інформації з використанням мови Python // V Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IOT» - Київ. 18.04.24, ДУІКТ. С.43-45.
2. Аверічев І.М., Мельник В.В. Розробка програмного забезпечення для автоматизованого вилучення корисної інформації за допомогою Python // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ» - Київ. 24.04.24, ДУІКТ. С.48-49.

13

ВИСНОВКИ

1. Досліджено особливості вирішення задачі вилучення інформації з текстових документів комерційного спрямування
2. Проаналізовано цифрові рішення для вилучення з комерційних документів цінної для користувача інформації.
3. Визначено функціональні та нефункціональні вимоги для вилучення з комерційних документів цінної для користувача інформації.
4. БЛок
5. Розроблено інструментальні засоби для вирішення задачі вилучення з комерційних документів цінної для користувача інформації.
6. Проведено тестування застосунку.

14

ДОДАТОК Б
ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
import sys
from PyQt5.QtWidgets import QApplication,
QMainWindow, QPushButton, QVBoxLayout, QWidget,
QTextEdit, QFileDialog, \
  QTableWidget, QTableWidgetItem
import re
import nltk
class MainWindow(QMainWindow):
  def __init__(self):
    super().__init__()

    self.setWindowTitle("Вилучення контактів з
файлу")
    self.setGeometry(100, 100, 800, 600)

    layout = QVBoxLayout()

    self.text_edit = QTextEdit()
    layout.addWidget(self.text_edit)

    load_button = QPushButton("Відкрити файл")
    load_button.clicked.connect(self.load_file)
    layout.addWidget(load_button)

    process_button = QPushButton("Опрацювати
текст")
    process_button.clicked.connect(self.process_text)
    layout.addWidget(process_button)

    save_button = QPushButton("Зберегти результат")
    save_button.clicked.connect(self.save_result)
    layout.addWidget(save_button)

    self.table_widget = QTableWidget()
    layout.addWidget(self.table_widget)

    central_widget = QWidget()
    central_widget.setLayout(layout)
    self.setCentralWidget(central_widget)

  def load_file(self):
    file_dialog = QFileDialog()
    file_path, _ = file_dialog.getOpenFileName(self,
"Вибрати файл", "", "Текстові файли (*.txt)")

    if file_path:
      with open(file_path, "r", encoding="utf-8") as file:
        text = file.read()
        self.text_edit.setPlainText(text)

  def process_text(self):
    text = self.text_edit.toPlainText()
    ім_фам, телефони, пошти, адреси, дати, події,
url_адреси = вилучити_контакти(text)

    headers = ["Ім'я", "Телефон", "Пошта", "Адреса",
"Дата", "Подія", "URL"]
    self.table_widget.setColumnCount(len(headers))

self.table_widget.setHorizontalHeaderLabels(headers)

    row_count = max(len(ім_фам), len(телефони),
len(пошти), len(адреси), len(дати), len(події),
len(url_адреси))
    self.table_widget.setRowCount(row_count)

    for i, ім_фамілія in enumerate(ім_фам):
      item = QTableWidgetItem(ім_фамілія)
      self.table_widget.setItem(i, 0, item)

    for i, телефон in enumerate(телефони):
      item = QTableWidgetItem(телефон)
      self.table_widget.setItem(i, 1, item)

    for i, пошта in enumerate(пошти):
      item = QTableWidgetItem(пошта)
      self.table_widget.setItem(i, 2, item)

    for i, адреса in enumerate(адреси):
      item = QTableWidgetItem(адреса)
      self.table_widget.setItem(i, 3, item)

    for i, дата in enumerate(дати):
      item = QTableWidgetItem(дата)
      self.table_widget.setItem(i, 4, item)

    for i, подія in enumerate(події):
      item = QTableWidgetItem(подія)
      self.table_widget.setItem(i, 5, item)

    for i, url in enumerate(url_адреси):
      item = QTableWidgetItem(url)
      self.table_widget.setItem(i, 6, item)

    self.table_widget.resizeColumnsToContents()

  def save_result(self):
    file_dialog = QFileDialog()
    file_path, _ = file_dialog.getSaveFileName(self,
"Зберегти результат", "", "Текстові файли (*.txt)")

    if file_path:
      with open(file_path, "w", encoding="utf-8") as
file:
```

```
            headers =
"\t".join([self.table_widget.horizontalHeaderItem(i).text(
) for i in range(self.table_widget.columnCount())])
            file.write(headers + "\n")
            for row in range(self.table_widget.rowCount()):
                row_data = []
                for column in
range(self.table_widget.columnCount()):
                    item = self.table_widget.item(row,
column)
                    if item is not None:
                        row_data.append(item.text())
                    else:
                        row_data.append("")
                file.write("\t".join(row_data) + "\n")

def вилучити_контакти(text):
    # Вирази для пошуку різних типів даних
    # Вираз для пошуку імені та прізвища
    pattern_ім_фам =
r'\b[А-ЯІЇЄҐ][а-яіїєґ]+\s[А-ЯІЇЄҐ][а-яіїєґ]+\b'
    # Вираз для пошуку номеру телефону
    pattern_телефон = r'\+380\d{2}[ -]?\d{3}[ -]?\d{2}[
-]?\d{2}|\b0\d{9}\b'
    # Вираз для пошуку електронної пошти
    pattern_пошта =
r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b
'


    pattern_адреса =
r'\bвулиця\s[А-ЩЬЮЯҐЄІЇа-щьюяґєії]+\s[А-ЩЬЮЯҐ
ЄІЇа-щьюяґєії]+\s\d{1,5}\b'

    # Пошук в тексті за допомогою регулярних виразів
    pattern_дата =
r'\b\d{1,2}[-/\.]\d{1,2}[-/\.]\d{2,4}\b|\b\d{4}-\d{2}-\d{2}
\b|\b\d{1,2}\s[А-ЩЮЯІЇЄҐа-щьюяіїєґ]+\s\d{4}\b'


    pattern_подія =
r'\б(конференція|збори|зустріч|вебінар|свято|подія|сем
інар)\б'
    pattern_url = r'\бhttps?://[^\c]+'

    # Пошук в тексті за допомогою регулярних виразів
    ім_фамілії = re.findall(pattern_ім_фам, text)
    телефони = re.findall(pattern_телефон, text)
    пошти = re.findall(pattern_пошта, text)
    адреси = re.findall(pattern_адреса, text)
    дати = re.findall(pattern_дата, text)


    події = re.findall(pattern_подія, text)
    url_адреси = re.findall(pattern_url, text)

    return ім_фамілії, телефони, пошти, адреси, дати,
події, url_адреси

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
```