НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації та управління

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)

Дипломний проєкт
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 126 «Інформаційні системи та технології»

на тему: «Інформаційна система добування фактів з україномовних текстів»

Виконав: студент *IV* курсу, групи *ІС-73*
Турко Микола Васильович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *доц., к.т.н., доц. Фіногенов Олексій Дмитрович* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з графічної документації *доц., к.т.н., доц. Сперкач Мая Олегівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2021 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації та управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім’я, прізвище)

ЗАВДАННЯ
на дипломний проєкт студенту

Турко Микола Васильович
(прізвище, ім’я, по батькові)

1. Тема проєкту «Інформаційна система добування фактів з україномовних текстів»
керівник проєкту *Фіногенов Олексій Дмитрович, к.т.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “*11*” *05* 2021 р. № 1139-с

2. Термін подання студентом проєкту “*04*”*червня* 2021 року

3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі
2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних
3. Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв’язання
4. Програмне та технічне забезпечення: засоби розробки, вимоги до

технічного забезпечення, архітектура програмного забезпечення, побудова звітів

5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1. Схема структурна варіантів використань

2. Схема структурна діяльності

3. Схема структурна компонентів програмного забезпечення

4. Схема структурна класів програмного забезпечення

5. Схема структурна послідовності

6. Креслення вигляду екранних форм

7. Рішення з математичного забезпечення

6. Консультанти розділів проєкту

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

7. Дата видачі завдання *«7» квітня 2021 року*

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*10.04.2021*	
2.	*Аналіз існуючих методів розв'язання задачі*	*12.04.2021*	
3.	*Постановка та формалізація задачі*	*13.04.2021*	
4.	*Розробка інформаційного забезпечення*	*18.04.2021*	
5.	*Алгоритмізація задачі*	*21.04.2021*	
6.	*Обґрунтування використовуваних технічних засобів*	*23.04.2021*	
7.	*Розробка програмного забезпечення*	*27.04.2021*	
8.	*Налагодження програми*	*29.04 2021*	
9.	*Виконання графічних документів*	*02.04.2021*	
10.	*Оформлення пояснювальної записки*	*12.05.2021*	
11.	*Подання ДП на попередній захист*	*14.05.2021*	
12.	*Подання ДП на основний захист*	*04.06.2021*	
13.	*Подання ДП рецензенту*	*07.06.2021*	

Студент Микола ТУРКО

Керівник Олексій ФІНОГЕНОВ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 7328.00.000 ПЗ	Пояснювальна записка	83	
3	А4	ДП 7328.01.000 ТЗ	Технічне завдання	11	
4	А3	ДП 7328.1139-с ССВ	Схема структурна варіантів використань	1	
5	А3	ДП 7328.1139-с ССД	Схема структурна діяльності системи	1	
6	А3	ДП 7328.1139-с ССД	Схема структурна компонентів	1	
			програмного забезпечення		
7	А3	ДП 7328.1139-с ССП	Схема структурна послідовності	1	
8	А3	ДП 7328.1139-с ССК	Схема структурна класів програмного	1	
			забезпечення		
9	А3	ДП 7328.1139-с КЕ	Креслення вигляду екранних форм	2	
10	А3		Рішення з математичного забезпечення	1	

Зм.	Арк.	ПІБ	Підп.	Дата	ДП 7328 00.000.00		
Розробн.		Турко М. В.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Фіногенов О.Д.					
Консульт.						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-73	
Н/контр.		Сперкач М.О.					
В.о.зав.каф.		Павлов О. А.					

ЗМІСТ

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з п’яти розділів, містить 23 рисунків, 19 таблиць, 1 додаток та 18 джерел.

Дипломний проект присвячений вирішенню задачі добування фактів з україномовних текстів. Метою створення системи є автоматизація процесу добування структурованої інформації з неструктурованих текстових джерел за допомогою розпізнавання іменованих сутностей для української мови.

У розділі загальних положень описано процес діяльності добування фактів з тексту, функціональну модель системи та її відмінності від наявних аналогів. Визначено мету розробки та встановлено задачі, які необхідно вирішити.

У розділі інформаційного забезпечення надано опис вхідних та вихідних даних, включно з детальним описом базу даних словника для проведення морфологічного аналізу.

Розділ математичного забезпечення присвячений обґрунтуванню вибраних методів розв’язання задачі та опису алгоритму, який розпізнає іменовані сутності.

Розділ програмного забезпечення описує засоби розробки програмного продукту та етапи проектування його архітектури. Описано специфікацію функцій та звіти, які генеруються в ході запуску програми.

У технологічному розділі визначено мету проведення випробувань програмного продукту та описано їх результати. Також наведені результати експериментальних досліджень.

ОБРОБКА ПРИРОДНОЇ МОВИ, ОБРОБКА ТЕКСТУ, РОЗПІЗНАВАННЯ ІМЕНОВАНИХ СУТНОСТЕЙ.

ANNOTATION

The structure and the scope of the work. Explanatory note of the diploma project consists of five sections, contains 27 drawings, 19 tables, 1 addition-paper and 11 sources.

The diploma project is devoted to solving the problem of extracting facts from Ukrainian-language texts. The purpose of the system is to automate the process of extracting structured information from unstructured text sources by recognizing named entities.

The general provisions section describes the process of extracting facts from the text, including the functional model of the system and its differences from available analogues. This section also describes the purpose of development and tasks that need to be solved.

The information providing section gives a description of the input and output data, including a detailed description of the dictionary database for morphological analysis.

The section of mathematical software is devoted to the substantiation of the chosen methods of solving the problem and the description of the algorithm that recognizes the named entities.

The software section describes the tools for developing a software product and the stages of designing its architecture. Describes the specification of functions and reports generated during protokenTag startup.

The technological section defines the purpose of software product testing and describes their results. The results of experimental studies are also presented.

NATURAL LANGUAGE PROCESSING, TEXT PROCESSING, NAMED ENTITY RECOGNITION.

ВСТУП

Ще у середині минулого століття людство почало намагатись навчити комп’ютерні машини розуміти людську мову, сформувавши цілу сферу наукових досліджень під назвою обробка природної мови (англ. Natural Language Processing)[1].

На початку свого становлення як наукової області, обробка природної мови зосереджувалась на вирішенні задачі машинного перекладу. Проте з плином часу стало зрозуміло, що повністю вирішити цю задачу є майже недосяжною метою, так почали з'являтись нові напрями та задачі.

У сьогоденні обробка і розуміння природної мови комп’ютерами стала навіть актуальнішою – розроблюються діалогові системи штучного інтелекту, відбувається постійний пошук нових способів зменшення інформаційних потоків у людському житті, створюються системи ідентифікацій фейків і так дал. Перелік практичних застосування можна продовжувати ще довго, тому що обробка природної мови становить широкий спектр різноманітних задач, рішення яких є комбінацією певних інструментів.

Однією з найпоширеніших задач серед дослідників є задача розпізнавання іменованих сутностей (англ. Named Entity Recognition)[2]. Ця задача являє собою знаходження та виділення фактів в тексті, що відповідають заздалегідь обраним типам. Фактами можуть бути: іменами людей, дати подій, локації, назви організацій, тощо.

Вирішення задачі добування іменованих сутностей допомагає прибирати інформаційний шум та загалом зменшує обсяг інформації завдяки попередній обробці та структуризації. Формально це інструмент для добування структурованої інформації з неструктурованого тексту.

Вирішення задачі розпізнавання іменованих сутностей у текстах в свою чергу може покращити розв’язки задач знаходження плагіату або зменшити розповсюдження фейків порівнюючи факти з різних текстів.

Для україномовних текстів й досі не існує сервісу чи системи у відкритому доступі, що дозволяє досліджувати вирішення задачі розпізнавання іменованих сутностей.

Завданням бакалаврської роботи є розробка інформаційної системи, що буде добувати факти з україномовних текстів. Розпізнавання іменованих сутностей у тексті є багатокроковим процесом, тому виділяються наступні завдання до виконання:

- токенізація вхідних даних;
- морфологічний аналіз токенів включно з лематизацією;
- реалізація добування різних типів фактів з тексту;
- розробка веб-додатку для користування системою.

Практичне значення одержаних результатів. Розроблено систему, що автоматизує добування структурованої інформації з неструктурованого тексту за допомогою розпізнавання іменованих сутностей.

Публікації. Результати роботи були опубліковані у тезі доповіді «Алгоритм добування фактів з україномовних текстів» на науково-технічній конференції [3].

Бакалаврський проєкт містить наступні розділи: вступ, базові розділи, висновки, список використаних джерел із 16 найменувань та 1 додатку. Графічна частина складається з 8 креслеників формату А3. Загальний обсяг – 83 сторінки.

ЗАГАЛЬНІ ПОЛОЖЕННЯ

Опис предметного середовища

Обробка природної мови (англ. Natural Language Processing)[1] є великою областю з різноманітними напрямами, що вирішує наукові та комерційні задачі:

- машинний переклад;
- класифікація текстів;
- сентиментний та семантичний аналіз;
- автореферування тексту та інші.

Однією з найпоширеніших серед дослідників у світі задач є задача розпізнавання іменованих сутностей (англ. Named Entity Recognition)[2]. Це процес знаходження та виділення заздалегідь обраних фактів в тексті. Типами фактів можуть бути: іменами людей, дати подій, локації, назви організацій, тощо.

Наведемо приклади галузей застосування:

1) пошук кандидатів для вакансії (англ. Human Resource) – компанії стикаються з потребою в оцінці великої кількості резюме для відбору кандидатів. Навіть для людини читання та сортування резюме не є простим завданням. Можна спростити процес оцінки резюме використовуючи розпізнавання сутностей.

   В цьому випадку фактами будуть: назви посади, вміння кандидата, адреси електронної пошти і так далі.

2) діалогові системи – у теперішніх умовах соціальної ізоляції кількість діалогових систем зростає. Яскравим прикладом є діалогова системи лікарні. Від користувачів надходить інформація про стан здоров'я, а відповідно система надає рекомендації.

   Іменованими сутностями в такому випадку можуть бути: симптоми.

3) алгоритми пошуку інформації в інтернеті – алгоритми пошуку опрацьовують величезні обсяги даних при знаходженні збігу до введеного запиту. Ефективніше опрацьовувати сайт чи статтю один раз і постійно зберігати пов'язані об'єкти. І тоді при пошуку можна порівняти ключові слова запиту з тегами. Також це може покращити процес пошуку плагіату або ж знаходження фейків.

Загалом, будь-яка сфера, де потрібно зменшувати обсяг інформації, прибрати інформаційний шум може використовувати фільтрацію за допомогою добування фактів.

Опис процесу діяльності

У простому вигляді розпізнавання іменованих сутностей полягає у послідовному виконанні двох дій:

– знаходження слова або послідовності слів, що є фактом;

– ідентифікація типу факту.

Не зважаючи на те, що на перший погляд це досить тривіальні для вирішення запитання, проте виникають труднощі з виконанням кожного етапу.

Для виконання першого кроку потрібно застосовувати токенізацію (іноді використовується тернім сегментація) вхідних даних. Існує два типи сегментації тексту:

– токенізація на речення;

– токенізація на самостійні слова-компоненти.

Для розпізнавання іменованих сутностей використовується токенізація на слова-компоненти, але це не просто розбиття на слова враховуючи «пробіл» чи розділові знаки. Токенізація – це процес попередньої обробки тексту для визначення об'єктів для подальшого аналізу.

Основною проблемою для виконання другого кроку, ідентифікації типу факту, є багатозначність слів, що пишуться однаковим чином. Виокремлюють два види багатозначності:

- полісемія – це вид багатозначності, коли слова мають спільний початковий контекст. Наприклад слово «Вашингтон» може бути містом, прізвище людини, навчальний заклад, тощо.
- омонімія – це багатозначність з абсолютно різними за значенням словами. Наприклад слово «ключ» може бути скрипковим, гайковим, або ж просто до квартири.

Ця проблема не має однозначного рішення, але найкращим методом є морфологічний аналіз слів. Пояснення до цього процесу полягає у тлумаченні терміну морфології в рамках науки про мовознавство: «Це розділ граматики, в якому вивчають явища, що характеризують граматичну природу слова як граматичної одиниці мови. Це вчення про будову та граматичні класи слів (частини мови), граматичні категорії і систему словозміни їх. Основною одиницею морфології є слово, але в аспекті граматичної будови, особливостей змінювання і творення, вираження властивих слову граматичних значень» [4]. Тобто, морфологічний аналіз – це надання граматичної характеристики опису слова. Так як морфологічний аналіз застосовується після виконаної сегментації вхідного тексту, то аналіз відбувається над токенами.

Важливим етапом для вирішення проблеми багатозначності слова є його приведення до нормальної форми (лематизація), в результаті чого отримується початкова форма слова(лема). Для прикметників та іменників «лема» – це слово в називному відмінку та у формі одними, для дієслова – слово-відповідь на «що робити?». Лематизації токена застосовується після морфологічного аналізу, тому що для цього процесу важливо знати до якої частини мови відноситься токен. Наприклад якщо пояснити, що «Зеленський» це не прикметник, а іменник, то лематизація не змінить форму слова. А від цього залежить подальше розпізнавання сутності «прізвище» в цьому токені.

Проте застосувавши морфологічний аналіз токену та знайшовши його лему комп'ютер все рівно не може добути факти з тексту, для цього потрібно його «навчити» розпізнавати іменовані сутності. Слово «навчання» найчастіше передбачає використання нейронних мереж, проте для того щоб досягти високої якості розпізнавання іменованих сутностей потрібно мати великий навчальний датасет. На жаль для української мови у відкритому доступі розмічені корпуси текстів наявні в обмеженій кількості. Окрім навчання нейронних мереж існує також спосіб надання системі власноруч зіставлених правил послідовності токенів та їх характеристик після морфологічного аналізу. В такому випадку опрацювання правил полягає у парсингу тексту.

Алгоритми парсингу тексту не є новітнім інструментом обробки текстової інформації. Ще у 80-их роках минулого століття такі алгоритми використовувались компіляторами для парсингу файлів різноманітних мов програмування. Нюансом для вибору алгоритму є те, що мови програмування мають досить чітку структуру своєї побудови, а ось українська мова займає провідні позиції у різноманітних рейтингах граматичної складності. Тому побудова правил для іменованих сутностей не є тривіальною задачею і алгоритм парсингу повинен опрацьовувати всі ці правила для вірної екстракції фактів.

Загалом діяльність системи можна уявити у послідовності виконання різних аналізів вхідного тексту, що в результаті добувають факти з нього.

У графічних матеріалах наведено схему структурну діяльності системи.

Опис функціональної моделі

У графічних матеріалах наведено схему структурну варіантів використання. Єдиним актором у функціональний моделі для розробленої системи є користувач. У рамках дипломного проєкту користувач – це фізична особа, що використовує систему для добування фактів.

У таблиці 1.1 наведено варіанти використання системи користувачем та опис до кожного з варіантів.

Таблиця 1.1 - Варіанти використання системи користувачем

Варіант використання	**Опис**
Надання вхідного тексту	Користувач надає вхідний текст двома варіантами: або введення власноруч, або завантажуючи текстовий файл
Вибір типів фактів	Користувач можете вільно обирати які саме іменовані сутності будуть розпізнаватись при аналізі тексту
Аналіз тексту для отримання фактів	У цьому варіанті використання користувач запускає аналіз тексту, який відбувається за допомогою наступних функцій системи: - токенізація тексту; - морфологічний аналіз (включно з отриманням леми токену); алгоритм парсингу мови, що містить правила для розпізнавань іменованих сутностей.
Отримання результатів парсингу	Після аналізу тексту користувач отримує добуті факти з тексту у вигляді звітів.

До вказаних варіантів використання системи сформовані функціональні вимоги опис яких наведено у таблиці 1.2.

Таблиця 1.2 – Функціональні вимоги

Варіант використання	Функціональна вимога	Пріоритет вимоги
Надання вхідного тексту	При надані вхідного тексту система повинна перевіряти наявність інформації для аналізу та при її відсутності повідомляти користувача про це. Для вхідних текстових файлів потрібна перевірка підтримуваних системою форматів.	Високий
Вибір типів фактів	Система повинна дозволяти користувачеві обирати різні комбінації типів фактів для добування. При відсутності вибору повинно відбуватись нотифікування користувача.	Високий
Отримання результатів парсингу	Система повинна надавати чіткі звіти для добутих фактів з україномовних текстів.	Високий

Після закінчення розробки програмного продукту будуть проводитись випробовування до вищезазначених функціональних вимог.

Огляд наявних аналогів

На жаль, для української мови немає загальнодоступних сервісів та систем, що дозволяють досліджувати процес добування фактів з текстів. Найбільш схожою на українську мову є російська, тому розглядаються існуючі аналоги для цієї мови. Загалом усі системи та сервіси для вирішення задачі розпізнавання іменованих сутностей можна розділити на дві групи:

1) рішення, що використовують методи машинного навчання – останньою тенденцією у всьому світі при обробці природної мови є застосування методів машинного навчання, а саме використання нейронних мереж. Такі системи спочатку «навчають» на тренувальних даних виконувати завдання розглядаючи множину вже вирішених прикладів.

   Машине навчання зазвичай застосовується, коли неможливо вирішити задачу розробивши звичайний по-кроковий алгоритм. Проблематикою таких систем є нестача даних для формування тренувальних мов. І якщо для найпопулярніших мов світу розроблюються відкриті корпуси з розміченими даними, то для української мови такі корпуси відсутні.

a) BERT-model [5] – у 2018 році компанія Google представила своє рішення для сфери обробки природної мови: **BERT** – **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. У алгоритмі використовується модель прихованої мови (masked language model), суть якої полягає в тому, що у вхідному тексті приховаються випадкові слова, а комп'ютер повинен по контексту інших слів зрозуміти, яке слово було приховано. Навчання моделі відбувається в обидві сторони – зліва-направо та зправо-наліво, таким чином будуючи контекст для кожного слова з вхідного набору тренувальних даних.

   BERT – це багатофункціональне рішення, що відразу зайняло лідерську позицію для використання різноманітних задачах обробки природної мови для найпоширеніших мов світу, наприклад суммаризації тексту чи проходженні тесту «запитання-відповідь». На всесвітньо відомому корпусі CoNLL-2003[6] для англійської мови даний алгоритм отримав якісну міру розпізнавання іменованих сутностей 96%, що є найкращим результатом на сьогоднішній день.

b) *SpaCy* [7] – це безкоштовна бібліотека з відкритим кодом різних інструментів для обробки природної мови на мові програмування Python. Це рішення має широке застосування у комерційних задачах завдяки простій імпліментації готових методів та має досконалу документації й підтримку користувачів. Для задачі розпізнавання іменованих сутностей spaCy використовує згорткові нейронні мережі, які щоправда показують гірший результат якості на корпусі CoNLL-2003 аніж BERT, а саме 93%.

Переваги рішень основаних на використанні методів машинного навчання:

- для розробки та підтримки не потрібен фахівець-лінгвіст природної мови;
- висока якість розпізнавання іменованих сутностей при великому наборі тренувальних даних;
- успішно реалізовану модель можна застосовувати для іншої мови.

Недоліки рішень основаних на використанні методів машинного навчання:

- складність розробки;
- підготування тренувальних даних потребує багато часу;
- відсутність універсальності (якщо змінюється тематика вхідних даних, то потрібно перенавчити систему).

2) рішення, що використовують підготовлені знання про мову – база знань для таких рішень будуються або за допомогою онтології, або за зіставленими правилами для алгоритмів парсингу. Так як створення та підтримка онтології потребує часу, фінансування та фахівця-лінгвіста, такий спосіб бази знань зустрічається вкрай рідко. Рішення, що формуються на алгоритмах парсингу та відповідно на їх правилах є більш гнучкими та універсальними для застосунку.

a) *Томіта-парсер* [8] – російська компанія Яндекс відкрила світу свій інструмент для обробки російської мови в 2014 році. Томіта-парсер поширюється як виконуваний файл .exe, що працює на основі алгоритму парсингу GLR (англ. *Generalized Left-to-Right*), що приймає граматики та словники для розпізнавання іменованих сутностей. Окрім застосування у внутрішніх командах розробки Яндексу, Томіта-парсер не став поширеним інструментом для задач NER, навіть серед країн СНД.

b) *Yargy* [9] – це бібліотека з відкритим кодом для вирішення задачі розпізнавання іменованих сутностей для російської мови. На початку існування цього проекту в 2016, команда розробників, проаналізувавши помилки Яндексу, реалізувала проект на мові програмування Python, що значно полегшило взаємодію звичайних користувачів з інструментом. Yargy за час свого існування був значно модифікований.

   Спочатку це рішення добувало факти, як і Томіта-парсер, використовуючи GLR-алгоритм парсингу. Головною проблемою в цьому алгоритмі є його час роботи. Для зменшення часових витрат на етапі розпізнавання іменованих сутностей алгоритм парсингу було змінено на алгоритм-Ерлі.

   На сьогоднішній день Yargy поєднує в собі алгоритм парсингу мови та методи машинного навчання для добування фактів з текстів російською мовою. В межах російської мови це рішення є головним конкурентом для spaCy в комерційному застосуванні.

Переваги рішень основаних на алгоритмах парсингу:

- робоча модель є простішою та зрозумілішою для підтримки;
- завжди є можливість змінити параметри алгоритму при неправильному добуванні фактів;
- універсальність (може застосовуватись до будь-яких потоків інформації).

Недоліки рішень основаних на базах знань:

– бази знань потребують побудови перед імплементацією у систему, що іноді займає багато часу;
– погіршення якості розпізнавання іменованих сутностей;
– системи з базами знань застосовуються лише для конкретної природної мови.

У таблиці 1.3 продемонстровано порівняння різних систем з розробленою.

Таблиця 1.3 – Порівняння з аналогами

Назви / **Функції**	BERT	SpaCy	Томіта - парсер	Yargy /Natasha	Власна система
Токенізація	+	+	+	+	+
Морфологічний аналіз	-	-	+	+	+
Семантичний аналіз	+	+	-	-	-
Вибір типів фактів	-	+	-	-	+
Відкритий доступ до коду	+	+	-	+	+
Підтримка української мови	+-	-	-	+	+

Постановка задачі

Призначення розробки

Призначенням розробки є добування фактів з україномовних текстів.

Цілі та задачі розробки

Метою розробки є автоматизація процесу добування структурованої інформації з неструктурованих текстових джерел за допомогою розпізнавання іменованих сутностей. Для досягнення мети потрібно вирішити наступні задачі:

- реалізувати підсистему токенізації;
- реалізувати підсистему морфологічного аналізу;
- реалізувати підсистему алгоритм парсингу;
- реалізувати підсистему екстракторів іменованих сутностей.

Висновок до розділу

В даному розділі розглянуто актуальність поставленої задачі. Представлено порівняння з існуючими аналогами вирішення для російської мови. Описано функціональну модель, акторів та взаємодію з програмним продуктом. Сформульовано цілі розробки та задачі, які потрібно вирішити для досягнення поставленої мети.

ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

Вхідні дані

Вхідні дані до системи надає користувач. Насамперед це україномовний текст та вибір фактів для добування.

Вихідні дані

Вихідними даними є звіт добутих фактів з вхідного тексту та сам текст з позначеними іменованими сутностями всередині нього.

Опис структури бази даних

При реалізації підсистеми морфологічного аналізу було обрано Великий Електронний Словник Української Мови [10] (надалі ВЕСУМ), як інструмент для пошуку граматичної інформації про токен.

Команда Василя Старко[11] робить значний внесок для розвитку області обробки української мови, ВЕСУМ допомагає вирішувати головну проблему аналізу слова – багатозначність; окрім звичайного відображення приналежності до частини мови, для кожного слова є ряд інших ознак.

У таблиці 2.1 наведено перелік позначень слова та пояснення до кожного з них з прикладом.

Таблиця 2.1 – Опис тегів для словника ВЕСУМ

Частина мови	Ознака	Пояснення	Приклад
Іменник (noun)	anim	*істота*	Людина
	fname	*ім'я*	Микола
	lname	*прізвище*	Турко
	pname	*по батькові*	Васильович
	prop	*неістота*	Будівля
	geo	*топонім*	Україна

Продовження таблиці 2.1

Частина мови	Ознака	Пояснення	Приклад
Дієслово (verb)	imperf	*недоконаний вид*	Йти
	perf	*доконаний вид*	Пішов
	rev	*зворотна форма*	(-ться), (-йти)
	futr, past, pres	*часові ознаки*	Йшов, йду, піду
Прикметник (adj)	compb	*базова форма*	Хороший
	compc	*порівняння*	Кращий
	comps	*найвища форма*	Найкращий
Різні частин мови	m, f, n	*рід слова*	-
	p, s	*Однина, множина*	-
	abbr	*абревіатура*	-
	number	*Число*	-
	v_naz, v_rod, v_dav, … v_kly	*відмінок слова*	-

Приклад взаємодії зі словником продемонстровано на рисунку 2.1

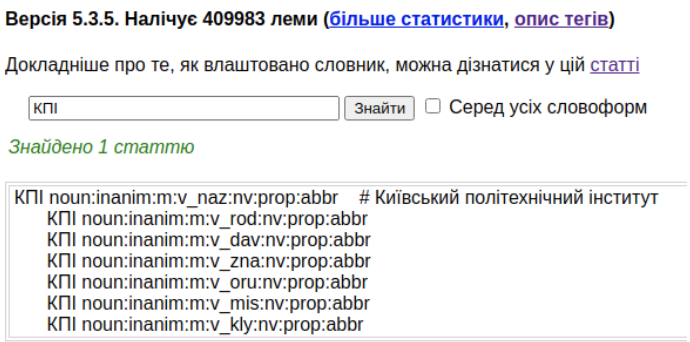


Рисунок 2.1 – Приклад запиту ВЕСУМ

Для імплементації словника в систему довелось витягнути усі слова та ознаки та зберігати інформацію у локальній базі даних. Також, було модифіковано початкову структуру ознак слова словника – було додано можливість пошуку слова за нормальною формою, таким чином реалізуючи процес лематизації.

У базі даних основною є лише одна таблиця – словник, а її структура зображена на рисунку 2.2.

Рисунок 2.2 – Діаграма сутностей бази даних

З рисунку видно, що у таблиці VESUM зберігається чотири колонки, де:

- id – індекс кожного слова для швидкодії пошуку;
- word – слово у будь-якій формі для пошуку;
- normForm – початкова форма слова або ж лема;
- tags – масив ознак для слова (перелік ознак див. у табл. 2.1).

Статистика бази даних наведена на рисунку 2.3, а на рисунку 2.4 приклад взаємодії відповідно.

```
> db.vesum.stats()
{
        "ns" : "NER-uk.vesum",
        "size" : 993839127,
        "count" : 6420462,
        "avgObjSize" : 154,
        "storageSize" : 293875712,
```

Рисунок 2.3 – Статистика ВЕСУМ

```
> db.vesum.find({'word':'Зеленському'}).pretty()
{
        " id" : ObjectId("609c7597524f637c44c7cfbf"),
        "word" : "Зеленському",
        "normForm" : "Зеленський",
        "tags" : [
                "noun",
                "anim",
                "m",
                "v dav",
                "prop",
                "lname"
        ]
}
{
        " id" : ObjectId("609c7597524f637c44c7cfc3"),
        "word" : "Зеленському",
        "normForm" : "Зеленський",
        "tags" : [
                "noun",
                "anim",
                "m",
                "v mis",
                "prop",
                "lname"
        ]
}
```

Рисунок 2.4 – Результати пошуку за запитом «Зеленському»

Висновок до розділу

У даному розділі визначено вхідні дані від користувачів системи добування фактів та вихідні дані після аналізу тексту. Представлені пояснення щодо морфологічного аналізу, а саме використання словника ВЕСУМ. У вигляді таблиць описані ознаки слів та наведено структуру таблиці-словника. При використанні даного алгоритму для інших мов можливе розширення бази даних з використанням іншомовних словників для близьких мов та додаткових таблиць суттєво відмінних від слов'янських.

МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

Змістовна постановка задачі

Після того як вхідний текст було токенізовано та проведено морфологічний аналіз, масив токенів надається до підсистеми алгоритму парсингу Ерлі.

Вхідними даними, окрім токенів, є також обрані користувачем типи фактів для добування.

Процес парсингу полягає у пошуку відповідності токенів шаблонам перетворення в іменовані сутності. Шаблони – це правила, що надаються за допомогою граматик.

Граматики становлять собою нетермінали (об’єкти або ж класи мови), термінали (протилежні за сенсом) та правила перетворення терміналів на нетермінали.

Математична постановка задачі

Контекстно-вільна граматика представлена у вигляді формули (3.1), що містить:

- множину нетермінальних символів за формулою (3.2);
- множину термінальних символів (токенів) за формулою (3.3);
- множина набору правил для перетворення в нетермінальні символи за формулою (3.4).

Вхідні дані до алгоритму:

$$G = \{V, E, R\} \quad (3.1)$$

$$V = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\} \quad (3.2)$$

$$E = \{a, b, c\} \quad (3.3)$$

$$R = \{A \to B; A \to a\ c\} \quad (3.4)$$

Обґрунтування методу розв’язання

Існує безліч алгоритмів для парсингу, проте більшість з них розроблена для аналізу мов програмування, тобто регулярних мов, які мають певні правила побудови. Алгоритм Ерлі[12] аналізує природні мови, використовуючи контекстно-вільні граматики. Це один з типів граматик за ієрархією Хомського, що дозволяє аналізувати термінальний символ без врахування контексту (сусідні термінальни символи, тощо).

У найгірших випадках працює алгоритм працює $O(n^3)$, де n - кількість токенів у вхідному потоці. Якщо граматика повністю детермінує мову, то час роботи алгоритму є лінійним.

Опис методу розв’язання

З отриманням вхідних даних будується матрицю станів S , де кожен стан – це поточний крок роботи парсеру, що характеризується правилами з граматиками. Нульовий крок відповідає нульовому елементу з тексту, і далі рухаючись зліва-направо по одному токену перебирає можливі правила для застосування, та при збігу додаючи до стану нове правило та індекс від якого стану це правило з’явилось. Тоді матриця станів має вигляд (3.5):

$$S = \begin{bmatrix} S_{00} & \cdots & S_{0j} \\ Z_{10} & \cdots & Z_{1j} \\ \vdots & \ddots & \vdots \\ Z_{i0} & \cdots & Z_{ij} \end{bmatrix}, i \in [0;\ k], j\ [0;\ n+1] \qquad (3.5)$$

Де:

n – кількість токенів у вхідному рядку;

k – кількість правил з вхідної граматики;

S_{ij} – це слова, що обробились на j-ому кроці;

Z_{ij} – це i-та множина з правила та індекса для j-го стану.

При ініціалізації елементи не нульового рядка матриці нульові. Приклад ініціалізації у вигляді псевдокоду:

```
function INIT(words, rules)
    S ← CREATE-MATRIX([LENGTH(rules)][LENGTH(words) + 1]//рядки-
стовпці
     for i ← from 0 to LENGTH(rules) do // цикл рядків
         for j ← from 0 to LENGTH(words) do // цикл стовпців
              S[i][j] ← EMPTY-ORDERED-SET //пусті кортежі Z
```

Застосовується три метода для зміни матриці станів.

1) сканування – якщо наступний елемент після поточного є терміналом, то парсер просувається на один токен праворуч, додаючи до наступного стану нову множину правила та індексу.

Робота методу у вигляді псевдокоду:

```
function SCANNER((A → α•aβ, j), k, words)
    if a is PARTS-OF-SPEECH(words[k]) then // перевірка на термінал
        ADD-TO-SET((A → αa•β, j), S[k+1])// додавання множини в стан
    end
```

2) передбачення – якщо наступний токен після крапки нетермінал, то шукається відповідність на правила в граматиці. Якщо така відповідність була знайдена, то до поточного стану додається нова множина правила та індексу.

Робота методу у вигляді псевдокоду:

```
function PREDICTOR((A → α•Bβ, j), k, tokenTagmar)
    for each (B → γ) in TOKENTAGMAR-RULES-FOR(B, tokenTagmar) do
                                                    //правила
                                                    перетворення
        ADD-TO-SET((B → •γ, k), S[k])
    end
```

3) завершення (згортки) – цей метод важливий для згортки після методу передбачень та сканування. Якщо в комірках стану є правило, що немає після себе символів та виконується за граматикою, то з множини комірок попереднього стану по індексу повертається вже оброблене нове

правило в поточний стан та відбувається просування по токену. В завершене правило замість індексу записується нуль.

Робота методу у вигляді псевдокоду:

```
function COMPLETER((B → γ•, i), k) //правило, що виконалось
    for each (A → α•Bβ, j) in S[i] do //пошук для згортки
        ADD-TO-SET((A → αB•β, j), S[k]) // згортка та просування
        ADD-TO-SET((A → αB•β, 0), S[j])
    end
```

Загальна робота алгоритму наведена у вигляді псевдокоду:

```
function EARLEY-PARSE(words, tokenTagmar)
    INIT(words, tokenTagmar)//ініціалізація матриці станів
    ADD-TO-SET((γ → •S, 0), S[0])//в нульовий стан відразу додається
                                 факт, що наступний елемент це термінал
    for k ← from 0 to LENGTH(words) do // цикл для кожного стану
        for each state in S[k] do
            if not FINISHED(state) then //перевірка на можливість дії
                if NEXT-ELEMENT-OF(state) is nonterminal then
                    PREDICTOR(state, k, tokenTagmar)// виклик методу
                                                    передбачення
                else do
                    SCANNER(state, k, words)// виклик сканування
            else do
                COMPLETER(state, k) //виклик методу згортки
        end
    end
    return chart
```

Тестовий приклад парсингу

Нехай вхідними даними будуть:

Граматика з правилами:

```
1.    P → S
2-3.  S → S "+" T | T
4.    T → [1-4]
```

Рядок: «2 + 3»

Маємо 3 токенів та 4 правил, отже після виклику методу ініціалізації маємо матрицю станів *S* наступного вигляду:

$$\begin{bmatrix} S0 & S1\text{:}"2" & S2\text{:}"2\ +" & S3\text{:}"2\ +\ 3" \\ \{P \to \cdot\, S\,;S_{00}\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset \end{bmatrix}$$

Для стану S0 відразу додали перше правило з граматики. Далі робота парсеру Ерлі починається саме з цього стану. Так як наступний елемент S – нетермінал, то запускається метод передбачення і застосовуються правила № 2-3. У випадку третього правила знову виконається метод передбачення для нетерміналу T. Матриця станів приймає вигляд:

$$\begin{bmatrix} S0 & S1\text{:}"2" & S2\text{:}"2\ +" & S3\text{:}"2\ +\ 3" \\ \{P \to \cdot\, S\,;S_{00}\} & \emptyset & \emptyset & \emptyset \\ \{S \to \cdot\, S + T;S_{01}\} & \emptyset & \emptyset & \emptyset \\ \{S \to \cdot\, T\,;S_{01}\} & \emptyset & \emptyset & \emptyset \\ \{T \to \cdot\, [1-4]\,;S_{03}\} & \emptyset & \emptyset & \emptyset \end{bmatrix}$$

Після нетермінального символу T можуть йти лише термінали 1-4, отже виконується метод сканування, і в стан S1 додається множина, що приводить матрицю до наступного вигляду.

$$\begin{bmatrix} S0 & S1\text{:}"2" & S2\text{:}"2\ +" & S3\text{:}"2\ +\ 3" \\ \{P \to \cdot\, S\,;S_{00}\} & \{T \to\ [1-4]\,\cdot\,;S_{05}\} & \emptyset & \emptyset \\ \{S \to \cdot\, S + T;S_{01}\} & \emptyset & \emptyset & \emptyset \\ \{S \to \cdot\, T\,;S_{01}\} & \emptyset & \emptyset & \emptyset \\ \{T \to \cdot\, [1-4]\,;S_{03}\} & \emptyset & \emptyset & \emptyset \end{bmatrix}$$

Стан S_{11} виконує метод згортки для комірки S_{04}, яка в свою чергу виконує цей метод для S_{03} та S_{02} та S_{00} породжуючи нові правила для стану S2: “2”.

$$\begin{bmatrix} S0 & S1\text{:}"2" & S2\text{:}"2\ +" & S3\text{:}"2\ +\ 3" \\ \{P \to \cdot\, S\,;0\} & \{T \to\ [1-4]\,\cdot\,;0\} & \emptyset & \emptyset \\ \{S \to \cdot\, S + T;S_{01}\} & \{S \to\ T\,\cdot\,;0\} & \emptyset & \emptyset \\ \{S \to \cdot\, T\,;0\} & \{S \to\ S\,\cdot + T;S_{11}\} & \emptyset & \emptyset \\ \{T \to \cdot\, [1-4]\,;0\} & \{P \to S\,\cdot\,;0\} & \emptyset & \emptyset \end{bmatrix}$$

З поточної матриці станів лише комірка S_{13} не є завершеною, в ній наступним токеном для аналізу є термінал «+» тому спрацьовує метод сканування.

$$\begin{bmatrix} S0 & S1\text{:}"2" & S2\text{:}"2\ +" & S3\text{:}"2\ +\ 3" \\ \{P \to \cdot S\,;0\} & \{T \to\ [1-4]\cdot;S_{05}\} & \{S \to\ S+\ \cdot T;S_{13}\} & \emptyset \\ \{S \to \cdot S+T;S_{01}\} & \{S \to\ T\cdot;S_{11}\} & \emptyset & \emptyset \\ \{S \to \cdot T\,;0\} & \{S \to\ S\cdot+T;S_{11}\} & \emptyset & \emptyset \\ \{T \to \cdot\,[1-4]\,;0\} & \{P \to S\cdot;S_{11}\} & \emptyset & \emptyset \end{bmatrix}$$

У комірці S_{21} за правилом є наступний символ нетермінал, тому виконується метод передбачення.

$$\begin{bmatrix} S0 & S1\text{:}"2" & S2\text{:}"2\ +" & S3\text{:}"2\ +\ 3" \\ \{P \to \cdot S\,;0\} & \{T \to\ [1-4]\cdot;S_{05}\} & \{S \to\ S+\ \cdot T;S_{13}\} & \emptyset \\ \{S \to \cdot S+T;S_{01}\} & \{S \to\ T\cdot;S_{11}\} & \{T \to \cdot\,[1-4]\,;S_{21}\} & \emptyset \\ \{S \to \cdot T\,;0\} & \{S \to\ S\cdot+T;S_{11}\} & \emptyset & \emptyset \\ \{T \to \cdot\,[1-4]\,;0\} & \{P \to S\cdot;S_{11}\} & \emptyset & \emptyset \end{bmatrix}$$

Комірка S_{21} з нетерміналом T в свою чергу виконує метод сканування, створюючи нову комірку для стану S3: “2 + 3”. Після сканування одразу буде виконано метод завершення, що повністю завершить процес парсингу для рядка. Кінцевий вигляд матриці станів має вигляд:

$$\begin{bmatrix} S0 & S1\text{:}"2" & S2\text{:}"2\ +" & S3\text{:}"2\ +\ 3" \\ \{P \to \cdot S\,;0\} & \{T \to\ [1-4]\cdot;0\} & \{S \to\ S+\ \cdot T;0\} & \{T \to\ [1-4]\cdot;0\} \\ \{S \to \cdot S+T;0\} & \{S \to\ T\cdot;0\} & \{T \to \cdot\,[1-4];0\} & \{S \to\ T\cdot;0\} \\ \{S \to \cdot T\,;0\} & \{S \to\ S\cdot+T;0\} & \emptyset & \{S \to\ S+T\cdot;0\} \\ \{T \to \cdot\,[1-4];0\} & \{P \to S\cdot;0\} & \emptyset & \{P \to\ S\cdot;0\} \end{bmatrix}$$

В результаті вхідний рядок звівся до нетерміналу P за коміркою S_{34}.

Висновок до розділу

У даному розділі розглянута математична постановка задачі. Наведено переваги використаного алгоритму парсингу мови та його обґрунтування. Описані методи та етапи роботи алгоритму у вигляді псевдокоду, та наведено тестовий приклад парсингу з покроковим поясненням дій.

ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

Засоби розробки

Для реалізації програмного продукту було використано мову програмування Python[13], фреймворк Streamlit[14] для розробки веб-додатку і база даних MongoDB[15] та середовище розробки Visual Studio Code[16].

Python

Пайтон – це інтерпретована, високорівнена мова програмування, що має динамічну типізацію. Головною перевагою для використання саме цієї мови є її відносна синтаксична простота, тобто абсолютно не знайомій з синтаксисом людині потрібно небагато часу, щоб зрозуміти, що відбувається у скрипті файлу. Пайтон є мультипарадигмовою мовою програмування, тобто дозволяє застосовувати структурне, об`єктно-орієнтоване та функціональне програмування. Основні архітектурні особливості: автоматичне управління пам`яттю, механізм обробки виключень, високорівневі структури даних.

Метою є створення універсального інструменту розпізнавання іменованих сутностей, який буде у відкритому доступі для всіх охочих модифікувати алгоритм. На прикладі Yargy (опис див. у розд. 1.5) видно результат застосування Python, як основної мови програмування, що поширило інструмент саме завдяки читабельності програмного продукту.

Streamlit

Бібліотека для розробки веб-додатків для Python. Цей фреймворк дозволяє швидко будувати клієнтську частину використовуючи вбудовані методи відображення результатів та роботи алгоритмів програми.

MongoDB

MongoDB – це інструмент створення та підтримки нереляційних баз даних для програмних продуктів. У другому розділі (див. розд. 2.1.3) було розглянуто структуру таблиці ВЕСУМ.

Visual Studio Code

VS Code – це інтегроване середовище розробки від компанії Microsoft. Даний інструмент було обрано через можливість користуватись ним на операційних системах з різними ядрами, будь то Unix, Mac OS або ж Windows. Не зважаючи на те, що це просто редактор коду, а не компілятор, VS Code все рівно дозволяє відслідковувати процес виконання коду і повідомляє про помилки.

Вимоги технічного забезпечення

Даний програмний продукт представлений у вигляді веб-додатку. У склад комплексу повинен входити персональний комп'ютер користувача. Система повинні відповідати пред'явленим в ТЗ вимогам:

- процесор з тактовою частотою не нижче 2 ГГц;
- об'єм оперативної пам'яті (не менше 4 ГБ);
- жорсткий диск (не менше 40 ГБ);
- операційна система Ubuntu 18.04 АБО Windows 10 (та вище);
- версія Python 3.6-3.8;
- версія Streamlit 0.80.0;
- версія MongoDB 4.2;
- Версія VS Code не нижче 1.13.

Архітектура програмного забезпечення

Так як вирішення задачі розпізнавання іменованих сутностей є багатокрокових процесом і включає в себе дії, що застосовуються і в інших задачах обробки природної мови, то система містить різні самостійні підсистеми. На рисунку 4.1 зображено взаємодія усіх підсистем.

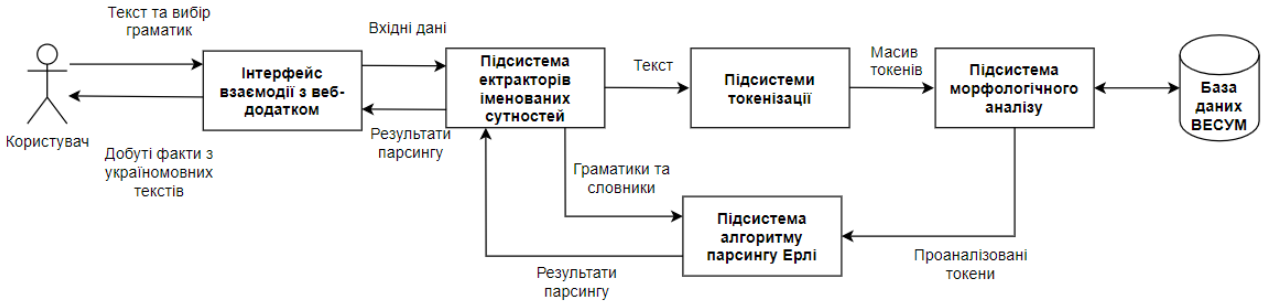


Рисунок 4.1 – Взаємодія підсистем

Ключовою для добування різних фактів є підсистема екстракторів іменованих сутностей.

Діаграма класів

У графічних матерілах наведено схему структурну класів програмного забезпечення. Система складається з ряду підсистем (див. рис. 4.1), що реалізуються як об’єктні моделі. Для кожної підсистеми розглядається своя діаграма класів в імплементаційній перспективі.

Підсистема морфологічного аналізу

Підсистема морфологічного аналізу потрібна в системі для інкапсуляції роботи із базою даних, а саме виконання наступних дій: створення зв’язку додатку з утилітою бази даних, ініціалізація бази даних зі словником (якщо екземпляру немає) та виконання запитів пошуку слів. Діаграма класів зображена на рисунку 4.2.

Рисунок 4.2 – Діаграма класів підсистеми морфологічного аналізу

Зв’язок між іншими підсистема та цією забезпечує клас-агрегатор Vesum.

Підсистема токенізації

Токенізація – це не просто розподіл речення на слова, це насамперед виокремлення самостійний частин з вхідного тексту. Речення не завжди закінчуються крапкою, і велика літера не позначає його початок, тому процес токенізації також будується на правилах для правильного сегментування вхідного тексту. Наприклад токен типу «цифра» не містить ніякої граматичної характеристики, тому застосування морфологічного аналізу не потребує. Діаграма класів зображена на рисунку 4.3.

Рисунок 4.3 – Діаграма класів підсистеми токенізації

Таким чином у токена є початковий тег після процесу сегментації, і у випадку наявності позначки токену «український» застосовується аналіз за словником ВЕСУМ.

Підсистема алгоритму парсингу Ерлі

Для реалізації алгоритму парсеру Ерлі окрім методів заповнення комірок (див. розд. 3.3) було також розроблено додаткові класи:

– клас правил граматики;

– клас колонок станів;

– клас стану відповідно;

– клас перетворення «терміналу» в «нетермінал».

На рисунку 4.4 зображено діаграму класів для цієї підсистеми.

Рисунок 4.4 – Діаграма класів підсистеми парсеру

Приклад граматики з правилами написаними для мови програмування Python зображено на рисунку 4.5.

```
NAME = and_(
    or_(
        gram('Name'),
        vesum_tag('fname')
    ),
    not_(ABBR)
)
```

Рисунок. 4.5 – Приклад правила граматики

У цьому прикладі нетермінал NAME задається предикатом and_ для одного токену в якому необхідно задовільними обидві умови, що токен має позначку ім’я :«fname» або «FName» та не є нетерміналом «ABBREVIATION».

Підсистема екстракторів іменованих сутностей

Ця підсистема являє собою вже готові граматики для різних типів фактів. У поточний момент розробки готові наступні граматики:

– імена людей;

– локації;

– дати;

– персони (посада/звання та людина).

Діаграма класів для даної підсистеми зображена на рисунку 4.6.

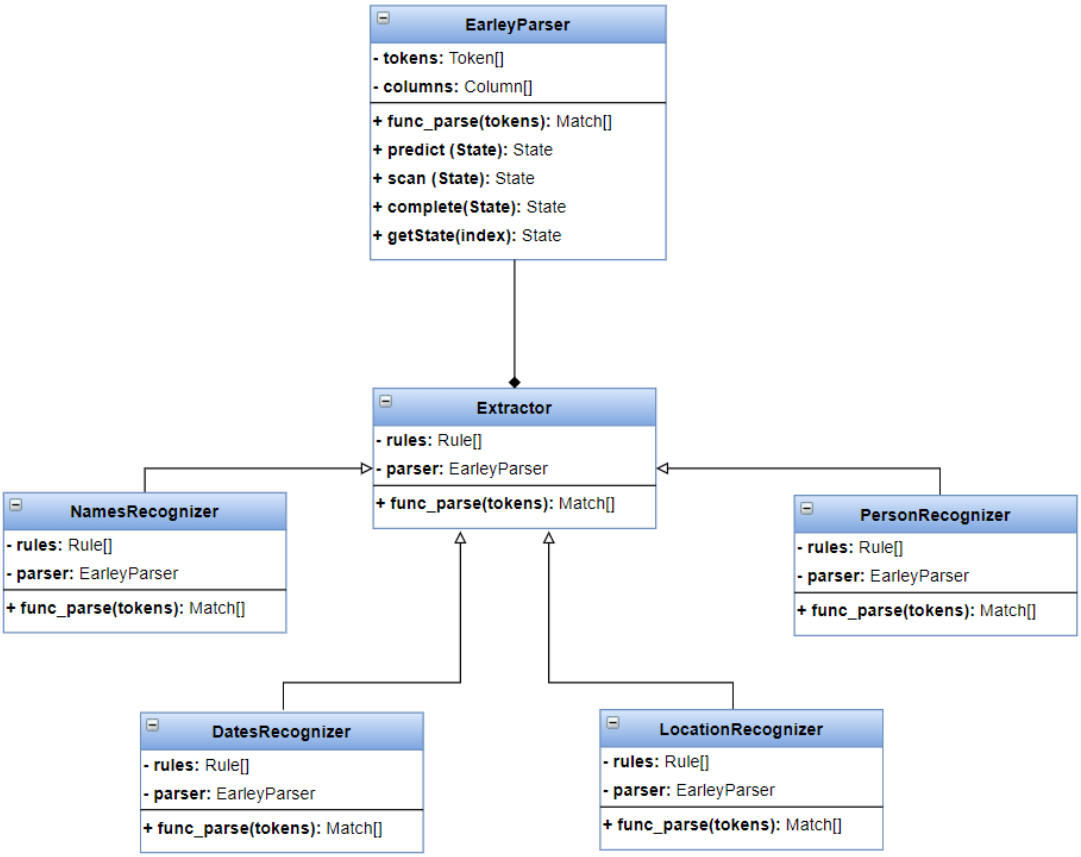


Рисунок 4.6 – Діаграма класів для підсистеми екстракторів іменованих сутностей

Підсистема веб-додатку

Підсистема веб-додатку – це реалізація інтерфейсу через який користувач надає вхідні дані (див. розд. 2.1) до системи добування фактів з україномовних текстів. Діаграма класів наведена у рисунку 4.7.

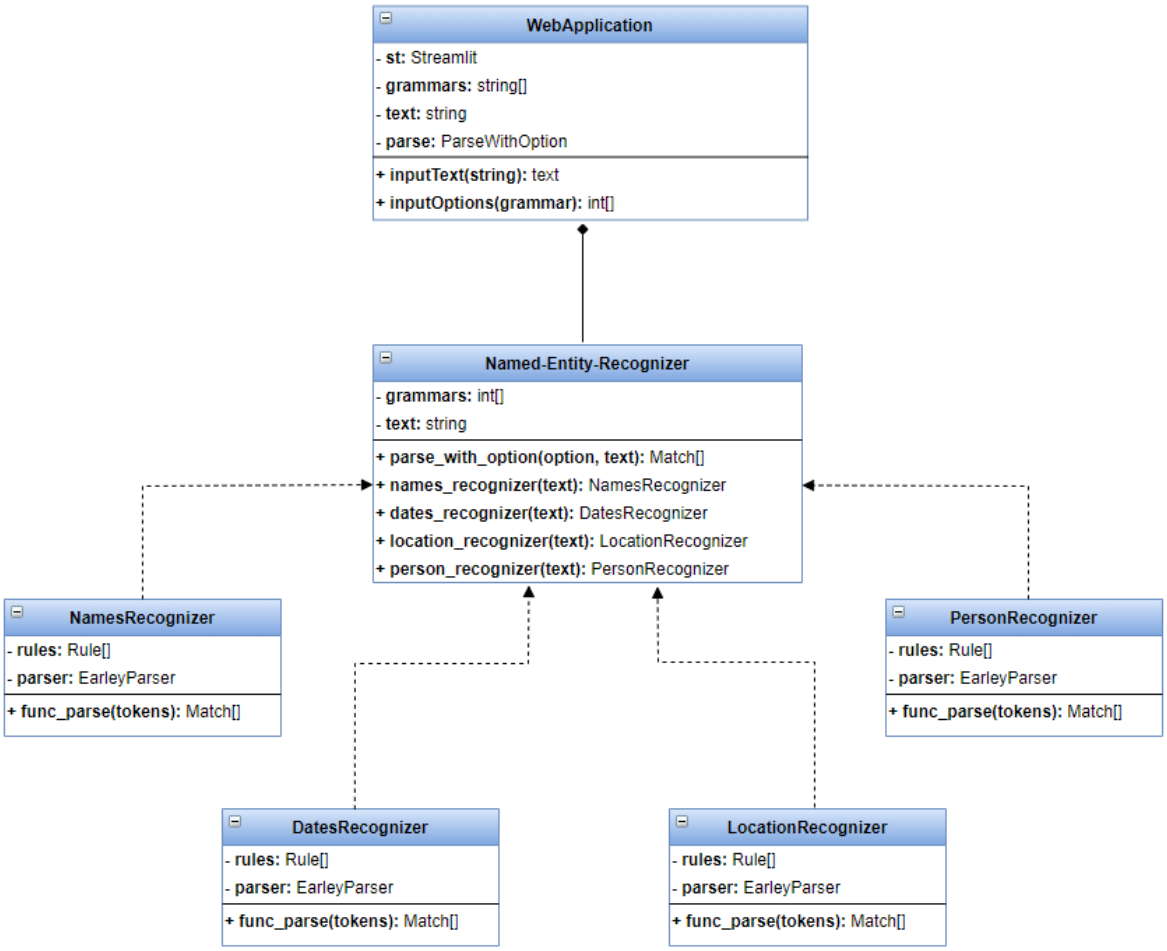


Рисунок 4.7 – Діаграма класів для підсистеми веб-додатку

Клас NameEntityRecognizer реалізує парсинг за граматиками:

– NamesRecognizer – імена;

– DatesRecognizer – дати;

– LocationRecognizer – локації;

– PersonRecognizer – персони(посади та імена людей).

Діаграма послідовностей

Схема структурна послідовності наведена у графічних матеріалах до дипломного проєкту. Користувач надає вхідні дані до системи за допомогою методів інтерфейсу веб-додатку inputText() та inputOptions(), після чого запускається аналіз вхідного тексту методом parse_with_option(). Перший крок аналізу виконується підсистемою Tokenizer та методом tokenize(), що сегментує речення на самостійні частини. У випадку, якщо токен відповідає типу «український» відбувається морфологічний аналіз з інкапсульованим словником VESUM та методом find_by_word(). Після отримання характеристик до кожного токену відбувається пошук за збігами шаблонів-правил підсистемою Earley Parser та методом func_parse() відповідно. В результаті роботи алгоритму парсингу повертається матриця (див. розд. 3) chart. З цієї матриці добуваються іменовані сутності типу match, що формуються звіт для користувача з методом text_markup(), що виділяє іменовані сутності у вхідному тексті. Загалом, користувач чекає на виконання аналізу кожною підсистемою, через це час очікування отримання результатів добування фактів становить більше однієї хвилини в середньому.

Діаграма компонентів

У графічних матеріалах наведена схема структурна компонентів програмного забезпечення, що складається з двох основних блоків:

– інтерфейс веб-додатку для взаємодії користувача та системи;

– серверна частина, що реалізує функції підсистем.

Компонент блоку веб-додатку Web-application надає вхідні дані від користувача до компоненту NER-component блоку серверу. Від компоненту NER-component відбувається делегування готових граматик для різних типів фактів. Компонент Tokenizer функціонально залежить від компоненту Vesum.

Специфікація функцій

У таблиці 4.1 наведено опис методів, що використовуються класами.

Таблиця 4.1 – Опис методів класів системи

Назва класу	Назва методу	Вхідні дані	Вихідні дані	Пояснення до дії
VesumDB	find_by_word()	string	vesumWord	Метод для пошуку слова у словнику ВЕСУМ. У результаті повертається екземпляр слова зі словника.
	createVesumDB()	void	void	Ініціалізація бази даних зі словником ВЕСУМ
MongoClient	connect()	root: string	void	Метод встановлення зв’язку з базою даних MongoDB
FileReader	open()	file:string	void	Відкриття текстового файлу за назвою
	readByLines()	Iterator<string>	string	Зчитування файлу, в результаті ітеративного виконання повертаються рядки файлу

Продовження таблиці 4.1

Назва класу	Назва методу	Вхідні дані	Вихідні дані	Пояснення до дії
Tokenizer	tokenize()	string	Token[]	Основний метод токенізації. Приймає текст та за правилами сегментує на самостійні частини. В результаті виконання повертається масив типу токен
Token	typeUK()		boolean	Якщо токен типу української мови, то відбувається пошук за словником ВЕСУМ
	normalize()	void	string	Метод зведення токену до нормальної форми. Лема знаходиться за словником ВЕСУМ
MorphToken	findByWord()	string	vesumArray	Цей метод успадковується від класу Vesum.

Продовження таблиці 4.1

Назва класу	Назва методу	Вхідні дані	Вихідні дані	Пояснення до дії
EarleyParser	func_parse()	tokens: Token[]	Match[]	Метод парсингу попередньо токенізованого тексту
	predict()	State	State	Метод передбачення для зміни комірки стану парсингу
	scan()	State	State	Метод сканування терміналу для стану
	complete ()	State	State	Метод завершення парсингу та згортки комірки стану
Column	getState()	Index:int	State	Отримання комірки стану за вказаним індексом
State	completed()	void	boolean	Булевий метод перевірки стану на закінчення парсингу
Rule	matchRule()	token: Token	boolean	Булевий метод перевірки збігу з правилом граматики

Продовження таблиці 4.1

Назва класу	Назва методу	Вхідні дані	Вихідні дані	Пояснення до дії
Rule	or_()	token:Token, rule: string	boolean	Метод предикату «або» для перевірки збігу токену за набором правил. Повертає істину у випадку хоча б одного збігу з набору
	hasMorphTag()	token: Token	boolean	Перевірка чи виконувався морфологічний аналіз для токену
	eq_()	token:Token, rule: string	boolean	Перевірка «збігу» для нетерміналу та терміналу
	not_()	token:Token, rule: string	boolean	Метод предикату «ніколи» для перевірки збігу токену за набором правил. Повертає істину у випадку невиконання всіх правил з вказаного набору

Продовження таблиці 4.1

Назва класу	Назва методу	Вхідні дані	Вихідні дані	Пояснення до дії
Rule	and_()	token:Token, rule: string	boolean	Метод предикату «лише тоді» для перевірки збігу токену за набором правил. Повертає істину у випадку виконання всіх правил перетворення для нетерміналу
Extractor (та його нащадки ...)	func_parse()	tokens	Match	Метод від агрегованого класу EarleyParser.
WebApplication	inputText()	string	text:string	Метод надання користувачем тексту в систему для парсингу
	inputOptions()	tokenTagmars: string[]	int[]	Метод вибору типів фактів для добування системою. Вибір інтерпретується за значенням в булевому масиві, де певний індекс відповідає типу факту.

Кінець таблиці 4.1

Назва класу	Назва методу	Вхідні дані	Вихідні дані	Пояснення до дії
NameEntity Recognizer	parse_with_ _option()	option:int[], text:string	Match[]	Метод, що запускає парсинг відповідно до обраних типів фактів. Реалізує класи-нащадки Extractor. В результаті повертаються розпізнанні іменовані сутності

Для найпоширеніших типів іменованих сутностей, що зустрічаються на конференціях з обробки природної мови та комерційного застосування у світі було розроблено граматики з правилами. У таблиці 4.2 наведено правила та пояснення правил для фактів типу «Імена».

Таблиця 4.2 – Правила граматики для факту «Імена»

Правило у коді	Пояснення	Приклад

`ABBREV = or_(`
`  tokenTag('Abbrev'),`
`  vesumTag('abbr')`
`)`
	Нетермінал «абревіатура» – токен має тег “abbr” або аналогічний тег за ВЕСУМ	*оон*

`NAME = and_(`
`  or_(`
`  tokenTag('Name'),`
`  vesumTag('fname')`
`  ),`
`  not_(ABBR)`
`). .interpretation(`
`  Name.first.inflected()`
`)`
	Нетермінал «ім’я» – *лише тоді,* коли токен містить тег ВЕСУМ “fname” та токен *ніколи* не є нетерміналом «абревіатура». Ім’я на відмінно від абревіатури може бути в різних відмінках, тому потребує подальшої лемантизації	*Миколою*

Продовження таблиці 4.2

Правило у коді	Пояснення	Приклад

`PARNAME = and_(`
`  or_(`
`  tokenTag('Patr'),`
`  vesumTag('pname')`
`  ),`
`  not_(ABBR)`
`)`
	Нетермінал «ім'я батьків» – *лише тоді,* коли токен містить тег ВЕСУМ "pname" та токен *ніколи* не є «абревіатурою»	*Васильович*

`LAST = eq_(`
`  vesumTag('lname')`
`).interpretation(`
`  Name.last.inflected()`
`)`
	Нетермінал «прізвище» - це рівність токена за тегом ВЕСУМ "lname" який буде лемантизовано	*Зеленському*

`NAME_ABBREV = and_(`
`  ABBREV,`
`  CAPITALIZED`
`).interpretation(`
`  Name.first`
`)`
	Нетермінал «абревіатурне ім'я» - *лише тоді,* коли токен перетворився в нетермінал «абревіатура» та записаний прописними літерами	*І.*

`LAST_FIRST = rule(`
`  LAST,`
`  FIRST )`
	Вище описане пояснення для зміни порядку	*Турко Микола*

`ABBREV_FIRST_LAST = rule(`
`  NAME_ABBREV,`
`  '.',`
`  LAST`
`)`
	Нетермінал комбінації «абревіатурне ім'я» та «прізвище» в заданому порядку	*М. Турко*

`LAST_ABBREV_FIRST = rule(`
`  LAST,`
`  NAME_ABBREV,`
`  '.',)`
	Вище описане пояснення для зміни порядку	*Турко М.*

`ABBREV_FIRST_MIDDLE_LAST =`
`  rule(`
`  NAME_ABBREV,`
`  '.',`
`  NAME_ABBREV,`
`  '.',`
`  LAST)`
	Нетермінал комбінації двох нетерміналів «абревіатурного імені» та нетерміналу «прізвище» в заданому порядку	*М. В. Турко*

`LAST_ABBREV_FIRST_MIDDLE =`
`  rule(`
`  LAST,`
`  NAME _ABBREV,`
`  '.',`
`  NAME _ABBREV,`
`  '.')`
	Вище описане пояснення для зміни порядку	*Турко М. В.*

Кінець таблиці 4.2

Правило у коді	Пояснення	Приклад

`FIRST_MIDDLE = rule(`
`  FIRST,`
`  MIDDLE )`
	Нетермінал комбінації нетерміналів «ім'я» та «по-батькові» в заданому порядку	*Микола Висильович*

`FIRST_MIDDLE_LAST = rule(`
`  FIRST,`
`  MIDDLE,`
`  LAST`
`)`
	Повноцінна комбінація з нетерміналів «ім'я», «по-батькові» та «прізвище»	*Микола Васильович Турко*

`LAST_FIRST_MIDDLE = rule(`
`  LAST,`
`  FIRST,`
`  MIDDLE`
`)`
	Зміна порядку для попереднього нетерміналу	*Турко Микола Васильович*

`JUST_FIRST = FIRST`	Тільки нетермінал «ім'я»	*Микола*
`JUST_LAST = LAST`	Тільки нетермінал «прізвище»	*Турко*

`NAME = or_(`
`  FIRST_LAST,`
`  LAST_FIRST,`
 	
`  ABBREV_FIRST_LAST,`
`  LAST_ABBREV_FIRST,`
`  ABBREV_FIRST_MIDDLE_LAST,`
`  LAST_ABBREV_FIRST_MIDDLE,`
 	
`  FIRST_MIDDLE,`
`  FIRST_MIDDLE_LAST,`
`  LAST_FIRST_MIDDLE,`
 	
`  JUST_FIRST,`
`  JUST_LAST,`
`).interpretation(`
`  Name)`
	Кореневе правило граматики, що містить усі можливі комбінації для розпізнавання іменованої сутності «Імена»	*Усі приклади нетерміналів вище*

У таблиці 4.3 наведено правила та пояснення правил для факту типу «Дати».

Таблиця 4.3 – Правила граматики для факту «Дати»

Правило у коді	Пояснення	Приклад
`MONTH_NAME = dictionary(MONTHS).interpretation( Date.month.normalized() .custom(MONTHS.__getitem__) )`	Нетермінал «назва місяцю» перетворює прописну назву в число за допомогою відновідностей у вказаній структурі даних.	*Травень → 5*
`MONTH = and_( gte(1), lte(12) ).interpretation( Date.month.custom(int) )`	Нетермінал «місяць» – *лише тоді* коли токен заданий числом, що знаходиться в інтервалі від 1 до 12	*Число в інтервалі від 1 до 12*
`YEAR_WORD = or_( rule('р', eq('.').optional()), rule(normalized('рік')) )`	Нетермінал «слово рік» – коли токен заданий лемою «рік» *або* токен «р.»	*Рік, р, р.*
`YEAR = and_( gte(1), lte(2100) ).interpretation( Date.year.custom(int) )`	Нетермінал «рік» – *лише тоді* коли токен заданий числом, що знаходиться в інтервалі від 1 до 2100	*Число в інтервалі від 1 до 2100*
`YEAR_SHORT = and_( length_eq(2), gte(0), lte(99) ).interpretation( Date.year.custom (lambda _: 1900 + int(_)) )`	Нетермінал «скорочений рік» – *лише тоді* коли токен складається з двох чисел, які обидва знаходяться в інтервалі від 0 до 99	*80-ті*
`ERA_YEAR = and_( gte(1), lte(100000) ).interpretation( Date.year.custom(int) )`	Нетермінал «рік ери» – *лише тоді* коли токен заданий числом, що знаходиться в інтервалі від 1 до 100000	*Число в інтервалі від 1 до 100000*

Продовження таблиці 4.3

Правило у коді	Пояснення	Приклад
ERA_WORD = rule(eq('до'), or_(rule('н', eq('.'), 'е', eq('.').optional()), rule(normalized('наша'), normalized('ера')))).interpretation(Date.current_era.const(False))	Нетермінал «слово ера» – три токени, що комбінують в заданому порядку термінал «до», лему «наша ера» *або* «н. е.»	*До н. е., до нашої ери*
DATE = or_(rule(DAY,'.',MONTH,'.', or_(YEAR,YEAR_SHORT), YEAR_WORD.optional()), rule(YEAR, YEAR_WORD), rule(DAY, MONTH_NAME), rule(MONTH_NAME, YEAR, YEAR_WORD.optional()), rule(DAY, MONTH_NAME, YEAR, YEAR_WORD.optional()), rule(ERA_YEAR, YEAR_WORD, ERA_WORD,)). interpretation(Date)	Кореневе правило граматики, що містить усі можливі комбінації для розпізнавання іменованої сутності «Дати».	*Усі приклади нетерміналів вище*

У таблиці 4.4 наведено правила та пояснення правил для факту типу «Локація».

Таблиця 4.4 – Правила граматики для факту «Локація»

Правило у коді	Пояснення	Приклад

`ADJECTIVE_TAG = or_(`
`  tokenTag('ADJF'),`
`  vesumTag('adj')`
`)`
	Нетермінал «Позначка прикметнику» коли токен має тег прикметнику. Назви локацій часто містять більше одного слова і одне з слів часто є прикметником. Для прикладу село *Великі* Кошарища	*Великі*

`REGION = rule(`
`  ADJECTIVE_TAG,`
`  dictionary({`
`  'край',`
`  'район',`
`  'область',`
`  'губернія',`
`  }),`
`).interpretation`
`(Location.name.inflected())`
	Нетермінал «Регіон» – з нетерміналу «прикметник» та власноруч заданою лемою.	*Київська область*

`AUTONOMOUS_DISTRICT = rule(`
`  ADJECTIVE_TAG.repeatable(),`
`  or_(`
`  rule(`
`  dictionary`
`  ({'автономний'}),`
`  Dictionary`
`  ({'округ'}),`
`  ),`
`  rule('АО'),`
`  ),`
`).interpretation`
`(Location.name.inflected())`
	Нетермінал «Автономний округ» – з нетерміналу «прикметник» та лем «автономний округ» *або* «а.о.»	*Чукотський а. о.*

`FEDERATION = rule(`
`  ADJECTIVE_TAG.repeatable(),`
`  dictionary({`
`  'республіка',`
`  'федерація',`
`  })`
`).interpretation`
`(Location.name.inflected())`
	Нетермінал «Федерація» – з довільного повторень нетерміналу «прикметник» та лем «республіка» або «федерація»	*Російська Федерація*

Продовження таблиці 4.4

Правило у коді	Пояснення	Приклад
STATE = rule(dictionary({ 'графство', 'штат', }), ADJECTIVE_TAG.optional(), or_(tokenTag('NOUN'), vesumTag('noun')),).interpretation (Location.name.inflected())	Нетермінал «штат» – з лем «республіка» або «федерація» та необов'язкового терміналу «прикметник» та токену з тегом ВЕСУМ «noun»	*Штат Алабама*
LOCALITY = rule(and_(dictionary({ 'місто', 'село', 'селище', }),).optional(), and_(ADJECTIVE_TAG,).optional(), and_(or_(vesumTag('geo')))).interpretation (Location.name.inflected())	Нетермінал «місцевість» – з необов'язкових лем: «місто»; або «село»; або «селище». необов'язкового терміналу «прикметник» та токену з тегом ВЕСУМ «geo».	*Село Великі Кошарища, місто Київ.*
LOCATION = or_(REGION, … LOCALITY,).interpretation(Location)	Кореневе правило граматики, що містить усі можливі комбінації для розпізнавання іменованої сутності «Дати».	*Усі приклади нетерміналів вище*

У таблиці 4.5 наведено правила та пояснення правил для факту типу «Персони».

Таблиця 4.5 – Правила граматики для факту «Персони»

Правило у коді	Пояснення	Приклад
`ROLE = morph_pipeline([])`	Нетермінал «Роль» – власноруч введені леми для можливих посад та позицій, що може займати особа у суспільстві	*Адвокат*

`SUBJECT = or_(`
`  tokenTag('gent'),`
`  vesumTag('v_rod')`
`)`
	Нетермінал «Суб'єкт» – токен з тегом ВЕСУМ "v_rod". Тобто це іменник в родовому відмінку, тому що нетермінал «Роль» часто відноситься до певного «Суб'єкту»	*Президента*

`MANY_SUBBJECT = or_(`
`  rule(SUBJECT),`
`  …`
`  rule(SUBJECT…SUBJECT)`
`  )`
	Нетермінал «Багато суб'єктів» – правило на довільний порядок з нетерміналів «Суб'єкт»	*Президента України*

`PERSON_ROLE = or_(`
`  ROLE,`
`  rule(ROLE, MANY_SUBEJCT)`
`).interpretation(`
`  Person.position`
`)`
	Нетермінал «Роль особи» – або нетермінал «роль» або комбінація нетерміналів: «роль» та «багато суб'єктів»	*Адвокат, адвокат президента України*

`PERSON = and_(`
`  PERSON_ROLE,`
`  NAME)`
`  .interpretation(Person`
`)`
	Корневе правило граматики, що декларує факт «Персона». Комбінує нетермінали: «Роль особи» та імені за граматикою «Імена» (див. табл. 4.2)	*Президент Володимир Зеленський*

Опис звітів

Якщо користувач обрав усі чотири наявні типи іменованих сутностей, то після того як відбувся аналіз вхідного тексту система надає користувачеві звіт по кожному типу добутого факту. У таблиці 4.6 наведено пояснення до звітів системи.

Таблиця 4.6 – Опис звітів

Тип факту	**Складова факту**	**Пояснення**	**Приклад**
Name (імена)	first	ім’я	*Микола*
	middle	по-батькові	*Васильович*
	last	прізвище	*Турко*
Location (локації)	name	власна назва локації	*Київ*
Date (дати)	year	рік	*2021*
	month	місяць	*Квітень*
	day	день	*Понеділок*
	current_era	булеве значення показнику відношення дати до нашої ери	*True*
Person (персони)	position	звання чи посада людини	*Президент*
	name	додатково людина, за звітом описаним вище	*Володимир Олександрович Зеленський*

При виборі певного типу факту звіт буде надаватись лише для цього типу. Користувач може комбінувати різни типи фактів своїм вибором.

Висновок до розділу

В даному розділі розглянуто технічні засоби для розробки програмного продукту. Наведено пояснення до обраної мови програмування та інструментів для розробки, діаграми класів для підсистем розробленого програмного продукту та загальна діаграма компонентів, опис головних методів для кожного розробленого класу, його вхідні та вихідні дані.

Розроблено структурну схему діаграми послідовностей, на якій було показано послідовність взаємодій між користувачем та підсистемами з демонстрацією послідовності часового виконання до отримання результату.

Наведено правила граматик для визначення різних типів іменованих сутностей.

ТЕХНОЛОГІЧНИЙ РОЗДІЛ

Керівництво користувача

Для користування системою потрібно локально завантажити програмний продукт. Якщо виконанні всі технічні вимоги (див. розд. 4.2) то для запуску системи користувач повинен виконати наступну послідовність дій:

1) Корневу теку з проєктом відкрити за допомогою VS Code або іншого середовища, що реалізує роботу з терміналом операційної системи. На рисунку 5.1 зображено файловий вигляд програмного забезпечення.

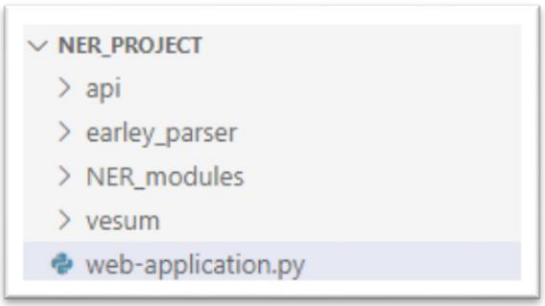


Рисунок 5.1 – Файлова структура проекту

2) у VS Code натиснути комбінацію клавіш “Ctrl” + “Shift” + “`” або відкрити термінал власноруч.
3) у терміналі ввести команду *“streamlit run web-application.py”* та натиснути клавішу “Enter”. На рисунку 5.2 приклад вигляду вікна введення команди в термінал.

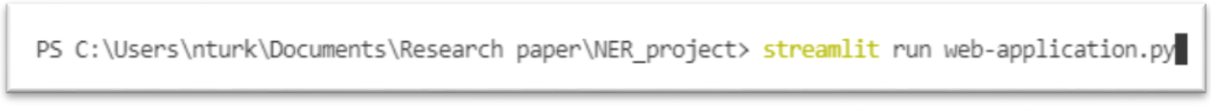


Рисунок 5.2 – Внутрішній термінал VS Code

4) зайти у веб-браузер за посиланням localhost:8501

Після покрокового виконання цих дій система працює у форматі веб-додатку. На рисунку 5.3 зображена стартова веб-сторінка системи.

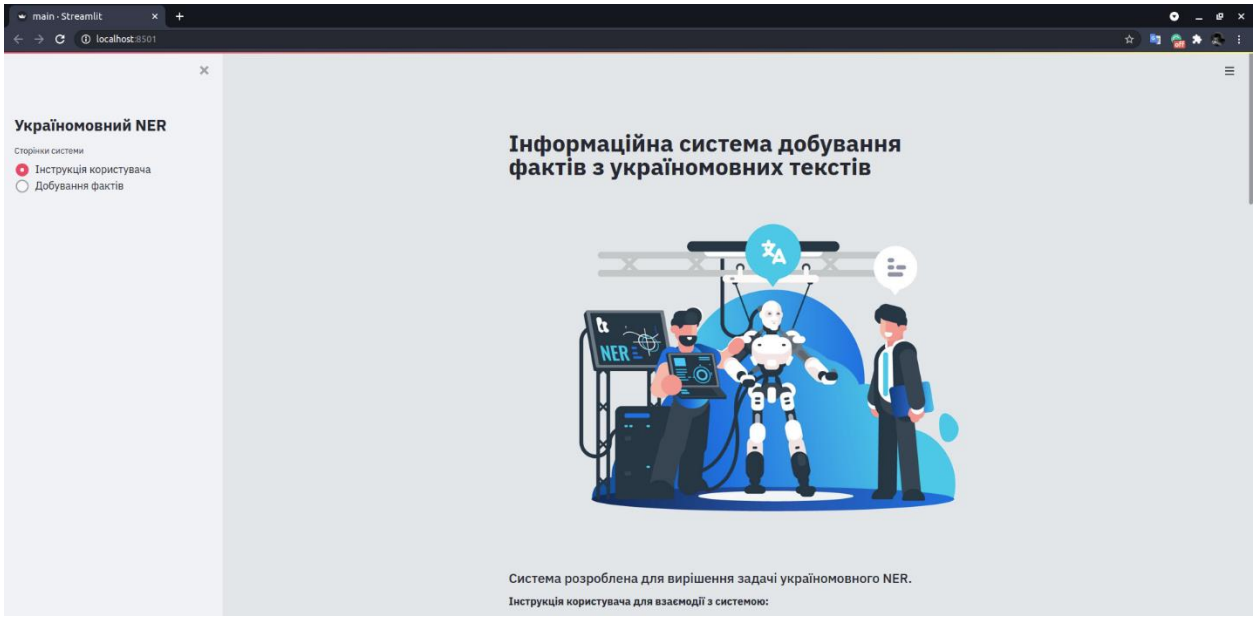


Рисунок 5.3 – Початковий вигляд системи

У системі загалом реалізовано дві веб-сторінки:

1) інструкція для взаємодії користувача з системою;
2) форма внесення даних для добування фактів.

Інструкція взаємодії користувача та системи

Після вибору сторінки добування фактів є вибір для введення тексту:

– власноруч ([1]) ;
– обравши файл ([2]).

На рисунку 5.4 зображена екранна форма для вибору користувачем

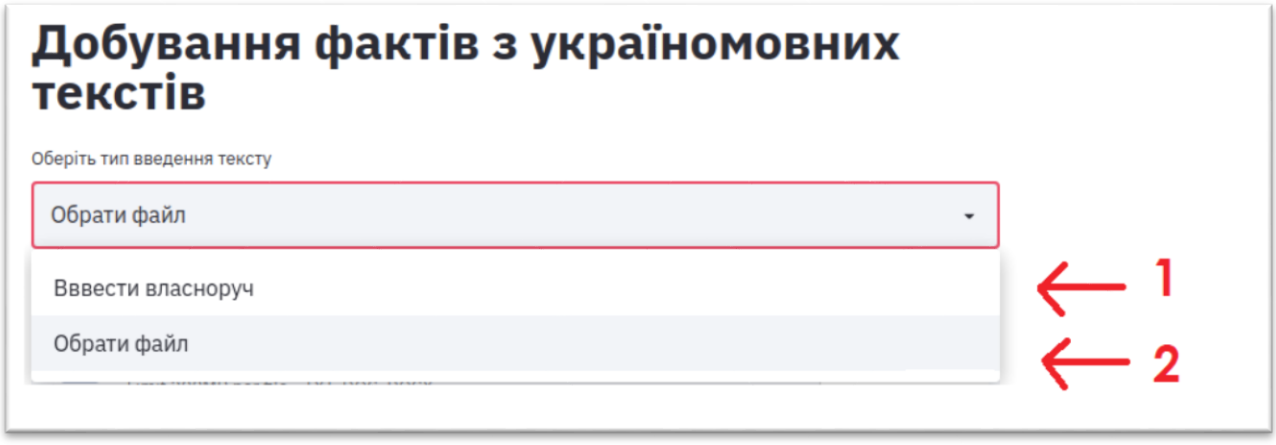


Рисунок 5.4 – Типи вибору введення файлу

При виборі введення власноруч з'являється відповідне поле для заповнення текстом ([1.1]), при виборі завантаження тексту з файлу з'являється кнопка, що відкриває провідник для локального вибору файлів ([2.1]). Система дозволяє завантажувати файли формату *.txt, *.doc, *.docx. На рисунках 5.5 та 5.6 зображено відповідні екранні форми.

Рисунок 5.5 – При виборі введення власноруч

Рисунок 5.6 – При виборі завантаження файлу

За вибір типів фактів для добування відповідають чек-бокси ([3]). Після надання вхідних даних для аналізу натискається кнопка запуску алгоритму ([4]). Процес аналізу тексту зображується за допомогою діалогового вікна очікування ([5]). На рисунку 5.7 та 5.8 наведено вигляд екранних форм вищеописаних елементів.

Рисунок 5.7 – При виборі завантаження файлу

Рисунок 5.8 – Вигляд діалогового вікна очікування результатів парсингу

Користувача повідомляють відповідним діалоговим вікном про завершення аналізу([6]) та надають звіт ([7]) добутих фактів. Екранна форма на рисунку 5.9.

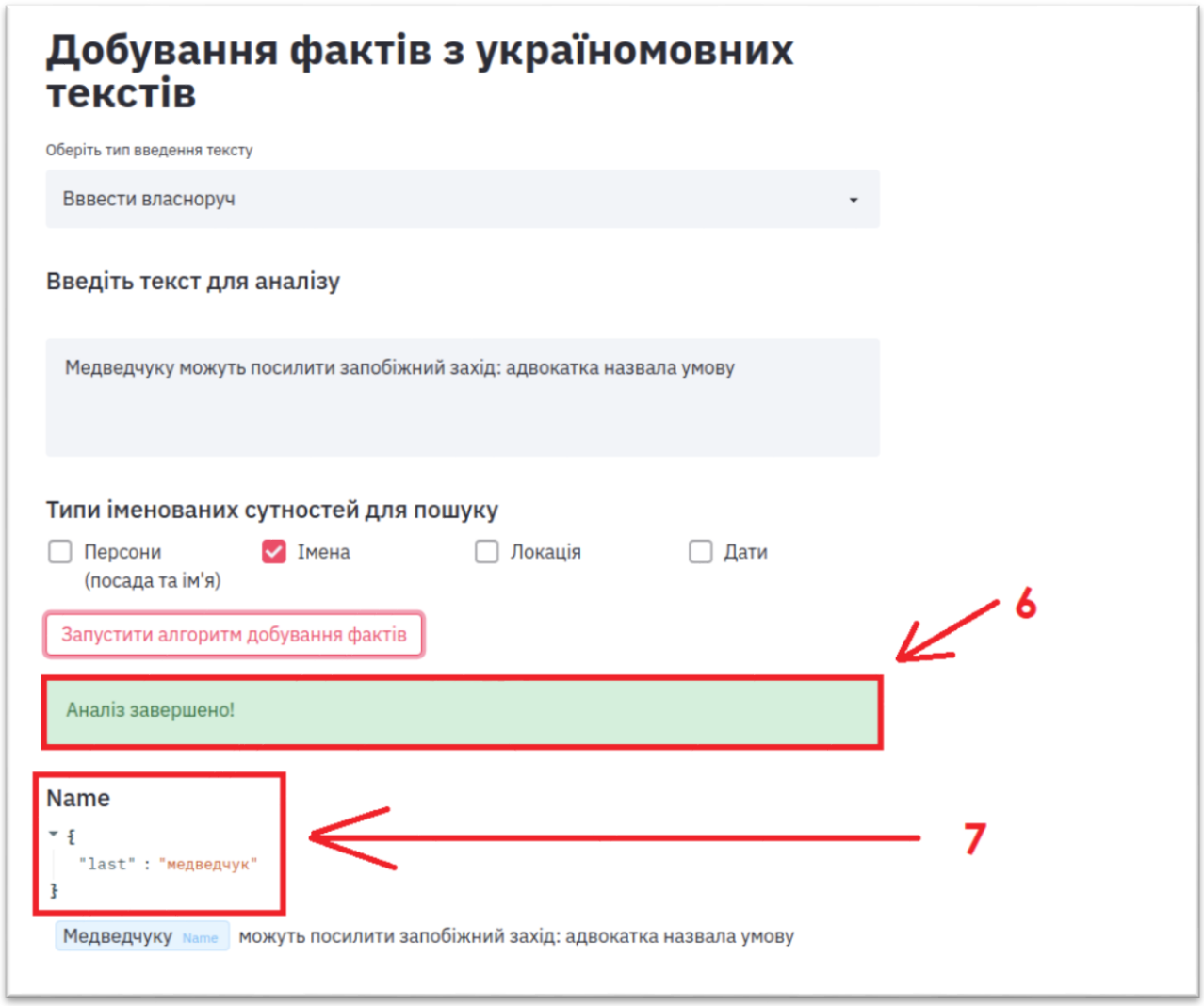


Рисунок 5.9 – Вигляд екранної форми результату парсингу

На рисунку 5.10 та 5.11 продемонстовано повноцінний результат добування фактів з україномовного тексту новини, взятої зі стрічки сайту «Україна 24».

Рисунок 5.10 – Джерело новини

Рисунок 5.11 – Результат парсингу для новини порталу «Україна 24»

Випробовування програмного продукту

Випробування програмного продукту можна розділити на дві частини:

- перевірка виконання функціональних вимог (див. розд. 1.3);
- перевірка підсистеми екстракторів іменованих сутностей.

Мета випробувань

Метою випробувань являється перевірка відповідності функцій інформаційної системи добування фактів з україномовних текстів вимогам технічного завдання. Для кожного тесту необхідно виконувати функцію та оцінити достовірність отриманих результатів.

Загальні положення

Випробування проводяться на основі наступних документів:

- ГОСТ 34.603 92. Інформаційна технологія. Види випробувань автоматизованих систем;
- ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту.

Результати випробувань

У таблицях 5.1-5.5 наведено тестові сценарії та їх виконання для випробовування функціональних вимог системи. За результатами тестів була підтверджена повноцінна правильність функціональної модель системи.

Таблиця 5.1 – Тестування вхідного тексту при власноручному введені

Тест:	Поле тексту при власноручному введені не пусте перед початком його аналізу
Початковий стан системи:	Відкрита сторінка добування фактів та обрано власноручний тип введення тексту
Дія:	Запуск аналізу тексту при його відсутності
Очікуваний результат:	Діалогове вікно для користувача про відсутність тексту
Фактичний результат:	*Відповідає очікуваному*

Таблиця 5.2 – Тестування на вірний формат файлу

Тест:	Тестування формату файлу, що завантажує користувач на відповідність до одного з вказаних форматів
Початковий стан системи:	Відкрита сторінка добування фактів, обрано завантаження файлу у типі введення тексту
Дія:	Спроба завантажити файл невказаного типу
Очікуваний результат:	Діалогове вікно для користувача про неправильний формат файлу
Фактичний результат:	*Відповідає очікуваному*

Таблиця 5.3 –Тестування вмісту вхідного текстового файлу

Тест:	Тестування вхідного текстового файлу на вміст тексту в ньому
Початковий стан системи:	Відкрита сторінка добування фактів, обрано завантаження файлу у типі введення тексту
Дія:	Спроба завантажити пустий текстовий файл
Очікуваний результат:	Діалогове вікно для користувача про відсутність тексту для аналізу
Фактичний результат:	*Відповідає очікуваному*

Таблиця 5.4 – Тестування відсутності вибору типів фактів

Тест:	Тестування відсутності вибору типів фактів для добування системою
Початковий стан системи:	Відкрита сторінка добування фактів та надано вхідний текст для аналізу
Дія:	Запуск аналізу тексту при відсутності вибору типів факту для добування
Очікуваний результат:	Діалогове вікно для користувача про відсутність вибору типів фактів
Фактичний результат:	*Відповідає очікуваному*

Таблиця 5.5 – Тестування вибору різних комбінацій типів фактів

Тест:	Тестування аналізу тексту при різних комбінаціях вибору типів фактів
Початковий стан системи:	Відкрита сторінка добування фактів та надано вхідний текст для аналізу
Дія:	Запуск аналізу тексту при бажаному типі факту для добування
Очікуваний результат:	Надання звіту до наявності факті у тексті
Фактичний результат:	*Відповідає очікуваному*

В таблиця 5.6-5.10 наведено сценарії та результати модульного тестування підсистеми екстракторів іменованих сутностей. Очікуваний та фактичний результат описується згідно звітам системи (див. розд. 4.8).

Таблиця 5.5 – Тестування розпізнавання сутностей типу «Імена»

Вхідний текст	Очікуваний результат добування	Фактичний результат добування
Нагорода президенту Володимиру Зеленському	{"first": "володимир", "last":" зеленський"}	*Відповідає очікуваному*
Нагорода президенту Зеленському Володимиру	{"first": "володимир", "last":" зеленський"}	*Відповідає очікуваному*
Нагорода президенту Зеленському В.	{"first": "в.", "last":" зеленський"}	*Відповідає очікуваному*
Нагорода президенту В. Зеленському	{"first": "в.", "last":" зеленський"}	*Відповідає очікуваному*
Нагорода президенту В. О. Зеленському	{"first": "в.", "middle": "о.", "last":" зеленський"}	*Відповідає очікуваному*
Нагорода президенту Зеленському В. О.	{"first": "в.", "middle": "о.", "last":" зеленський"}	*Відповідає очікуваному*
Нагорода президенту Володимиру Олександровичу	{"first": "володимир", "middle":"олександрович"}	*Відповідає очікуваному*
Нагорода президенту Володимиру Олександровичу Зеленському	{"first": "володимир", "middle":"олександрович", "last":" зеленський"}	*Відповідає очікуваному*

Продовження таблиці 5.4

Вхідний текст	Очікуваний результат добування	Фактичний результат добування

Нагорода президенту Володимиру Олександровичу Зеленському	{
	"first": "володимир",
	"middle":"олександрович",
	"last":" зеленський"
	}
		Відповідає очікуваному

Нагорода президенту Володимиру	{
	"first": "володимир"
	}
		Відповідає очікуваному

Нагорода президенту Зеленському	{
	"last":" зеленський"
	}
		Відповідає очікуваному

Таблиця 5.6 – Тестування розпізнавання сутностей типу «Персони»

Вхідний текст	Очікуваний результат добування	Фактичний результат добування
Нагорода президенту Володимиру Зеленському	{"position": "президент", name{ "first": "володимир", "last":" зеленський" }}	*Відповідає очікуваному*
Нагорода президенту України Зеленському Володимиру	{"position": "президент", name{ "first": "володимир", "last":" зеленський" }}	*Відповідає очікуваному*

Таблиця 5.7 – Тестування розпізнавання сутностей типу «Дати»

Вхідний текст	**Очікуваний результат добування**	**Фактичний результат добування**
Нагороджували 18.05.2021	{"year": 2021, "month": 05, "day": 18, "current_era": true}	*Відповідає очікуваному*
Нагороджували 18 травня	{"month": 05, "day": 18, "current_era": true}	*Відповідає очікуваному*
Нагороджували у 2021 р.	{"year": 2021, "current_era": true}	*Відповідає очікуваному*
Нагороджували 18 травня 2021 року	{"year": 2021, "month": 05, "day": 18, "current_era": true}	*Відповідає очікуваному*
Нагороджували у травні 2021 року	{"year": 2021, "month": 05, "current_era": true}	*Відповідає очікуваному*
У 459 році до нашої ери	{"year": 459, "current_era": false}	*Відповідає очікуваному*
У 459 році до н. е.	{"year": 459, "current_era": false}	*Відповідає очікуваному*

Таблиця 5.9 – Тестування розпізнавання сутностей типу «Локації»

Вхідний текст	Очікуваний результат добування	Фактичний результат добування
Нагороджували у місті Київ	{"name" : "місто Київ"}	*Відповідає очікуваному*
Полетів до штату Алабама	{"name" : "штат Алабама"}	*Відповідає очікуваному*
Сенсація у Білорусі	{"name" : "Білорусь"}	*Відповідає очікуваному*

В результаті випробовування підсистеми екстракторів іменованих сутностей перевірено точність роботи алгоритму парсингу відповідно до правил граматики.

Експериментальні дослідження

Також були проведені експериментальні дослідження щодо якості добування різних типів іменованих сутностей. Суть дослідження полягає у порівнянні попередньо підготовленого розміченого тексту з результатами добування фактів системою.

Для якості використовуються дві метрики [17-18]:

- точність – показує наскільки вірно відносно еталону була розпізнана іменована сутність;
- повнота – кількісний показник добування фактів з тексту відносно еталону.

Як було вже неодноразово написано, для української мови немає корпусів з розміченими текстами у відкритому доступі. Для проведення експериментальних досліджень було взято 100 речень з різних сайтів новин. Результати досліджень наведено на рисунку 5.12. В таблиці 5.10 наведені статистичні дані експериментальних та пояснення отриманих результатів.

Рисунок 5.12 – Дослідження точності добування фактів

Таблиця 5.10 – результаті експериментальних досліджень

Тип іменованої сутності	Точність	Повнота	Пояснення результатів
Імена	87%	94%	Зустрічаються імена які відсутні в словнику ВЕСУМА, тому це погіршує повноту.
Персони	91%	59%	Потрібно розширювати варіанти ролі для персон для підвищення якості розпізнавання. Зустрічаються випадки, коли позиція розділена з іменем людини іншими токенами.
Дати	98%	83%	Точність розпізнавання висока, проте повнота низька через вказування дат в новинах словами.
Локації	93%	82%	Багато власних імен локацій не додано у ВЕСУМ.

Висновок до розділу

У даному розділі наведена інструкція користувача для взаємодії з системою; визначена мета тестування та проведення тестування функціональної моделі та підсистеми екстракторів іменованих сутностей. Описано порядок проведення та результати експериментальних досліджень.

За отриманими результатами можна зробити висновок, що прості правила граматики мають погану точність, але покривають велику кількість фактів, як наприклад «Імена» – 94% повноти. І навпаки – складні правила, як наприклад «Персони» чи «Локації» мають велику точність – 91% та 93%, проте багато фактів пропускаються.

Вочевидь, що повнота визначення «Персони» має значно гірші показники якості розпізнавання, поліпшення яких є напрямком подальших досліджень та опрацювань алгоритму.

ЗАГАЛЬНІ ВИСНОВКИ

У пояснювальній записці розглянуто детальний опис дипломного проекту, що присвячений розробці інформаційної системи добування фактів з україномовних текстів.

Визначено мету та призначення розробки, що описана у розділі загальних положень. У розділі опису предметного середовища описано предметну область дипломного проекту. Для вирішення задачі розпізнавання іменованих сутностей представлений опис підсистем, що реалізуються. Було розроблено підсистеми токенізації, морфологічного аналізу та алгоритму пошуку розпізнавання різних іменованих сутностей. Користувач взаємодіє з системою через інтерфейс веб-додатку.

Розглянуто процес діяльності користувача та описано функціональну модель за допомогою схеми варіантів використання системи користувачем, на основі якої відображено функціональні вимоги до системи. Для української мови відсутні сервіси та системи вирішення задачі розпізнавання іменованих сутностей у відкритому доступі, тому порівняння системи з іншими рішеннями було виконано для російськомовних аналогів.

Розділ інформаційного забезпечення призначений для опису даних, якими оперує система та структур зберігання даних. Описано вхідні дані системи – україномовний текст та вибір типів фактів для добування. Також у цьому розділі описано використання словника для української мови ВЕСУМ, що використовувався для підсистеми морфологічного аналізу.

Математичне забезпечення – розділ, що містить змістовну та математичну постановку задачі, а також описує обраний алгоритми парсингу та його роботу, що вирішує задачу розпізнавання іменованих сутностей у тексті.

Опис та обґрунтування вибору засобів розробки наведено у розділі технічне забезпечення. У даному розділі описано архітектуру програмного забезпечення: представлено детальний опис підсистем, що були створені для розв’язання задачі, описано схему компонентів. Основні процеси у системі та

демонстрація внутрішньої реалізації та взаємодії окремих підсистем представлена за допомогою схем послідовності процесу добування фактів з україномовних текстів. Наведений опис до основних специфікацій програмного продукту та надані технічні вимоги для користування системою.

У технологічному розділі наведено керівництво користувача та екранні форми застосування. Також були проведені випробовування на відповідність функціональним вимогам системи та перевірка підсистеми розпізнавань іменованих сутностей. Наведені результати проведених експериментальних досліджень, в рамках системи реалізовано розпізнавання іменованих сутностей типу «імена», «дати», «локації» та «персони».

ПЕРЕЛІК ПОСИЛАНЬ

1. Madeleine B. Models of natural language understanding. *Proc. Natl. Acad. Sci. USA* Vol. 92, pp. 9977-9982, October 1995. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC40721/pdf/pnas01500-0075.pdf (дата звернення: 10.04.2021).

2. Rubtsova J. Custom Named Entity Recognition (NER) URL: https://medium.com/product-ai/custom-named-entity-recognition-ner-329e73cf909b (дата звернення: 12.04.2021).

3. Турко М.В. Алгоритм добування іменованих сутностей з україномовних текстів / М. В. Турко // Матеріали VI всеукраїнської науково-практичної конференції молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2021) – м. Київ.: НТУУ «КПІ ім. Ігоря Сікорського», 22-23 квітня 2021 р.

4. Вихованець І. Р. Морфологія. *Українська мова: енциклопедія*. / ред. В. М. Русанівський [та ін.]. — Київ, 2000. 750 с.

5. Toutanova BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding / Jacob Devlin Ming-Wei Chang Kenton Lee Kristina // URL: https://arxiv.org/pdf/1810.04805.pdf (дата звернення: 11.04.2021).

6. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition / Erik F., Tjong Kim Sang, Fien De Meulder // Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003. URL: https://www.aclweb.org/anthology/W03-0419.pdf (дата звернення: 18.04.2021).

7. SpaCy Architecture URL: https://spacy.io/api (дата звернення: 07.04.2021).

8. Что такое Томита-парсер, как Яндекс с его помощью понимает естественный язык, и как вы с его помощью сможете

извлекать факты из текстов. URL: https://habr.com/ru/company/yandex/blog/219311/ (дата звернення: 08.04.2021).

9. Yargy Набор качественных открытых инструментов для обработки естественного русского языка (NLP) URL: https://habr.com/ru/post/516098/ (дата звернення: 08.04.2021).

10. Trevor Jim Yitzhak Mandelbaum Efficient Earley Parsing with Regular Right-hand Sides / Electronic Notes in Theoretical Computer Science 253 (2010). 135–148 P. URL: https://www.sciencedirect.com/science/article/pii/S1571066110001167 (дата звернення: 12.04.2021).

11. Старко В. Ф. Великий електронний словник української мови (ВЕСУМ) як засіб NLP для української мови / В.Ф. Старко, А. Рисін // *Галактика Слова. Галині Макарівні Гнатюк.* Київ, 2020. — С. 135–141. URL: https://www.researchgate.net/publication/344842033_Velikij_elektronnij_slovnik_ukrainskoi_movi_VESUM_ak_zasib_NLP_dla_ukrainskoi_movi_Galaktika_Slova_Galini_Makarivni_Gnatuk (дата звернення: 14.04.2021).

12. Старко В. Комп'ютерні лінгвістичні проекти гурту r2u: стан та застосування / В. Старко // Українська мова. -2017. -№ 3. -С. 86-100.

13. Allen D. Think Python: An Introduction to Software Design. O'Reilly Media, Inc. 2012. 244 p. URL: https://greenteapress.com/thinkpython2/thinkpython2.pdf (дата звернення: 25.04.2021)

14. Streamlit - Introduction to beginners. URL: https://docs.streamlit.io/en/stable/(дата звернення: 26.04.2021)

15. MongoDB Documentation. URL: https://docs.mongodb.com/ (дата звернення: 28.04.2021)

16. VS Code User Guide. URL: https://code.visualstudio.com/docs/editor/codebasics (дата звернення: 28.04.2021)

17. NLP. Основы. Техники. Саморазвитие. Часть 1. URL: https://habr.com/ru/company/abbyy/blog/437008/ (дата звернення: 09.04.2021)

18. NLP. Основы. Техники. Саморазвитие. Часть 2: NER. URL: https://habr.com/ru/company/abbyy/blog/449514/ (дата звернення: 09.04.2021)

ДОДАТОК А

Тексти програмного коду

Інформаційна система добування фактів з україномовних текстів

(Найменування програми (документа))

DVD-R

(Вид носія даних)

7 арк., 25000 Кб

(Обсяг програми (документа), арк.,) Кб)

Київ – 2021 року

Web-aplications.py

```
import streamlit as st
from api import parsing_with_option as parse
from NER_modules import markup_text
from io import StringIO

def find_matches(text, options):
    text = text.strip()
    if text:
        if any(options):
            results_list = []
            spans = []
            with st.spinner('Відбувається аналіз тексту...'):
                for index_option in range(len(options)):
                    if options[index_option]:
                        parsing_result = parse(index_option, text)
                        results_list.append(parsing_result)

            st.success('Аналіз завершено!')
            for result in results_list:
                for match in result.matches:
                    res = str(match.fact).split(sep='(')
                    st.subheader(res[0])
                    st.write(match.fact.as_json)
                    spans.append([match.span[0], match.span[1], res[0]])
            st.write(markup_text(text, spans),unsafe_allow_html=True)


        elif not any(options):
            st.error('Оберіть типи іменованих сутностей')

    elif not text:
        st.error('Введіть текст для аналізу')

if __name__ == '__main__':

    st.sidebar.title('NER UA')

    user_choice = st.sidebar.radio('Оберіть сценарій роботи',
    ['Інструкція користувача', 'Добування фактів', 'Аналізатор інформаційних пото
ків'])
```

```
    if user_choice == 'Інструкція користувача':
        st.title('blank')

    elif user_choice == 'Добування фактів':
        text = False
        st.title('Добування фактів з україномовних текстів')
        input_choice = st.selectbox('Оберіть тип введення тексту',
                                    ['Ввести власноруч','Обрати файл'])
        if input_choice == 'Ввести власноруч':
            st.subheader('Введіть текст для аналізу')
            text = st.text_area(' ')
        elif input_choice == 'Обрати файл':
            uploaded_file = st.file_uploader(label="Завантажте локальний файл", t
ype=['txt','doc','docx'])
            if uploaded_file:
                text = StringIO(uploaded_file.getvalue().decode("utf-8")).read()
                st.subheader("Текст з файлу")
                st.markdown("*{}*".format(text))

        st.subheader('Типи іменованих сутностей для пошуку')
        cols = st.beta_columns(4)
        person_col = cols[0].checkbox("Персони (посада та ім'я)")
        name_col = cols[1].checkbox('Імена')
        location_col = cols[2].checkbox('Локація')
        date_col = cols[3].checkbox('Дати')
        options = [person_col, name_col, location_col, date_col]
        start = st.button('Запустити алгоритм добування фактів')

        if start and text:
            find_matches(text, options)
        elif start and text != True:
            st.error('Введіть текст для аналізу')
    elif user_choice == 'Аналізатор інформаційних потоків':
        st.subheader('Аналіз інформаційних потоків')
```

vesumDB.py

```
from pymongo import MongoClient
from string import whitespace

client= MongoClient('localhost:27017')
db = client["NER-uk"]
collection = db["vesum"]

text = open("dict_corp_vis.txt", 'r').readlines()
norm_form = ''
for line in text:
    line = line.replace('\n', '')
    if line[0] not in whitespace:
        word_entity_arr = line.split(' ')
        norm_form = word_entity_arr[0]
    else:
        line = line.strip()
        word_entity_arr = line.split(' ')

    word_entity = {
        'word': word_entity_arr[0],
        'normForm': norm_form,
        'tags': word_entity_arr[1].split(':')
        }
    collection.insert_one(word_entity)

def find_by_word(word):
    return collection.find({'word': word})
```

NER_Extractors.py

```
from NER_modules import extractors

def name_extracting_service(text):
    articles_matches = extractors.NamesExtractor()
    result = articles_matches(text)
    return result

def location_extracting_service(text):
    articles_matches = extractors.LocationExtractor()
    result = articles_matches(text)
    return result

def date_extracting_service(text):
    articles_matches = extractors.DatesExtractor()
    result = articles_matches(text)
```

```
    return result

def person_extracting_service(text):
    articles_matches = extractors.PersonExtractor()
    result = articles_matches(text)
    return result


def parsing_with_option (option, text):
    if option == 0:
        parser_result = person_extracting_service(text)
    elif option == 1:
        parser_result = name_extracting_service(text)
    elif option == 2:
        parser_result = location_extracting_service(text)
    elif option == 3:
        parser_result = date_extracting_service(text)
    return parser_result
```

Tokenizer.py

```
from __future__ import unicode_literals

import re

from .utils import Record, assert_type
from .span import Span
from .token import Token


class TokenRule(Record):
    __attributes__ = ['type', 'pattern']

    def __init__(self, type, pattern):
        self.type = type
        self.pattern = pattern


UKRAINIAN = 'UK'
LATIN = 'LATIN'
INT = 'INT'
PUNCT = 'PUNCT'
EOL = 'EOL'
OTHER = 'OTHER'

EMAIL_RULE = TokenRule(
```

```
    'EMAIL',
    r'[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+'
)
# https://toster.ru/answer?answer_id=852265#answers_list_answer
PHONE_RULE = TokenRule(
    'PHONE',
    r'(\+)?([-\s_()]?\d[-\s_()]?){10,14}'
)

GENERAL_QUOTES = '"\'”'
LEFT_QUOTES = '«„’'
RIGHT_QUOTES = '»“‘'
QUOTES = LEFT_QUOTES + GENERAL_QUOTES + RIGHT_QUOTES
RULES = [

    TokenRule(UKRAINIAN, r'[а-яіїйє]+'),
    TokenRule(LATIN, r'[a-z]+'),
    TokenRule(INT, r'\d+'),
    TokenRule(
        PUNCT,
        r'[-\\/!#$%&()\[\]\*\+,\.:;<=>?@^_`{|}~№…"\'«»„“’‘”]'
    ),
    TokenRule(EOL, r'[\n\r]+'),
    TokenRule(OTHER, r'\S'),
]


class Tokenizer(object):
    def __init__(self, rules=RULES):
        self.reset(rules)

    def reset(self, rules):
        for rule in rules:
            assert_type(rule, TokenRule)
        self.rules = rules
        self.regexp, self.mapping, self.types = self.compile(rules)

    def add_rules(self, *rules):
        self.reset(list(rules) + self.rules)
        return self

    def remove_types(self, *types):
        for type in types:
            self.check_type(type)
        self.reset([
            _ for _ in self.rules
            if _.type not in types
        ])
        return self
```

```
    def check_type(self, type):
        if type not in self.types:
            raise ValueError(type)


      def compile(self, rules):

        types = set()
        mapping = {}
        patterns = []
        for rule in rules:
            type, pattern = rule
            name = 'rule_{id}'.format(id=id(rule))
            pattern = r'(?P<{name}>{pattern})'.format(
                name=name,
                pattern=pattern
            )
            mapping[name] = type
            types.add(type)
            patterns.append(pattern)
        pattern = '|'.join(patterns)
        regexp = re.compile(pattern, re.UNICODE | re.IGNORECASE)
        return regexp, mapping, types

    def __call__(self, text):
        for match in re.finditer(self.regexp, text):
            name = match.lastgroup
            value = match.group(0)
            start, stop = match.span()
            type = self.mapping[name]
            token = Token(value, Span(start, stop), type)
            yield token

    def split(self, text):
        return [_.value for _ in self(text)]



class MorphTokenizer(Tokenizer):
    def __init__(self, rules=RULES, morph=None):
        super(MorphTokenizer, self).__init__(rules)
        if not morph:
            from .morph import CachedMorphAnalyzer
            morph = CachedMorphAnalyzer()
        self.morph = morph
```

```
    def __call__(self, text):
        tokens = Tokenizer.__call__(self, text)
        for token in tokens:
            if token.type == UKRAINIAN:
                forms = self.morph(token.value)

                yield token.morphed(forms)
            else:
                yield token

# coding: utf-8
from __future__ import unicode_literals, print_function


def assert_ipymarkup():
    try:
        import ipymarkup
    except ImportError:
        raise ImportError('pip install ipymarkup')


def get_markup_notebook(text, spans):
    assert_ipymarkup()
    from ipymarkup import BoxMarkup, Span
    from IPython.display import display

    spans = [Span(start, stop) for start, stop in spans]
    return BoxMarkup(text, spans)


def show_markup_notebook(text, spans):
    markup = get_markup_notebook(text, spans)
    display(markup)


def show_markup(text, spans):
    assert_ipymarkup()
    from ipymarkup import show_span_box_markup
    show_markup(text, spans)


def format_json(data):
    import json

    return json.dumps(data, indent=2, ensure_ascii=False)
```

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації та управління

УЗГОДЖЕНО	**ЗАТВЕРДЖУЮ**
Керівник проєкту	**В.о. завідувача кафедри**
_____ *Олексій ФІНОГЕНОВ*	_____ *Олександр ПАВЛОВ*
(підпис) (вл. ім’я, прізвище)	(підпис) (вл. ім’я, прізвище)
“5” квітня 2021 р.	“6” квітня 2021 р.

«Інформаційна система добування фактів з україномовних текстів»

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП 7328.01.000 ТЗ*

на 11 сторінках

Київ – 2021 року

ЗМІСТ

ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи та її умовне позначення

Повне найменування системи: «Інформаційна система добування фактів з україномовних текстів».

Коротке найменування: «Система».

1.2 Найменування організації-замовника та організацій-учасників робіт

Замовником системи є кафедру Автоматизованих систем обробки інформації та управління факультету інформатики та обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського». Адреса замовника: м. Київ, пр. Перемоги 37, корп.18.

Розробником системи є студент гр. ІС-73 кафедри Автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» Турко Микола Васильович.

1.3 Перелік документів, на підставі яких створюється система

Завдання на дипломний проєкт є підставою для розробки системи. При розробці системи за написанні документації учасники робіт повинні керуватися вимогами наступних нормативних документів:

- ДСТУ 19.201-78. Технічне завдання. Вимоги до змісту і оформлення;
- ДСТУ 34.601-90. Комплекс стандартів на автоматизовані системи. Автоматизовані системи. Стадії створення;
- ДСТУ 34.201-89. Інформаційні технології. Комплекс стандартів на автоматизовані системи. Види, комплектність і позначення документів при створенні автоматизованих систем.

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку роботи: 15 лютого 2021 року.

Плановий термін закінчення роботи: 30 травня 2021 року.

ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ

1.5 Призначення системи

Призначенням розробки є добування фактів з україномовних текстів.

1.6 Цілі створення системи

Метою розробки є автоматизація процесу добування структурованої інформації з неструктурованих текстових джерел за допомогою розпізнавання іменованих сутностей. Для досягнення мети потрібно вирішити наступні задачі:

- реалізувати підсистему токенізації;
- реалізувати підсистему морфологічного аналізу;
- реалізувати підсистему алгоритм парсингу;
- реалізувати підсистему екстракторів іменованих сутностей;

ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Об’єктом автоматизації є процес добування фактів з україномовних текстів.

Цей процес полягає у послідовному виконанні двох дій:

1) знаходження слова або послідовності слів, що є фактом;
2) ідентифікація типу факту.

Для виконання першого кроку потрібно застосовувати токенізацію (іноді використовується тернім сегментація) вхідних даних. Існує два типи сегментації тексту:

- токенізація на речення;
- токенізація на самостійні слова-компоненти.

Для розпізнавання іменованих сутностей використовується токенізація на слова-компоненти, але це не просто розбиття на слова враховуючи «пробіл» чи розділові знаки.

Проблемою для виконання другого кроку є багатозначність слів, що пишуться однаковим чином. Виокремлюють два види багатозначності:

a) полісемія – це вид багатозначності, коли слова мають спільний початковий контекст. Наприклад слово «Вашингтон» може бути містом, прізвище людини, навчальний заклад, тощо.
b) омонімія – це багатозначність з абсолютно різними за значенням словами. Наприклад слово «ключ» може бути скрипковим, гайковим, або ж просто до квартири.

Ця проблема не має однозначного вирішення, але найкращим методом є морфологічний аналіз слів. Морфологічний аналіз – це надання граматичної характеристики опису слова. Так як морфологічний аналіз застосовується після виконаної сегментації вхідного тексту, то аналіз відбувається над токенами. Також застосовується лематизація токенів.

Проте застосувавши морфологічний аналіз токену та знайшовши його лему комп’ютер все рівно не може добути факти з тексту, для цього потрібно його «навчити» розпізнавати іменовані сутності. Слово «навчання» відразу наштовхує на використання нейронних мереж, проте для того щоб досягти високої якості розпізнавання іменованих сутностей потрібно мати великий навчальний датасет. На жаль для української мови у відкритому доступі розмічених корпусів тексту немає. Окрім навчання нейронних мереж існує також спосіб надання системі власноруч зіставлених правил послідовності токенів та їх характеристик після морфологічного аналізу. В такому випадку опрацювання правил полягає у парсингу тексту. Нюансом для вибору алгоритму є те, що мови програмування мають досить чітку структуру своєї побудови, а ось українська мова займає провідні позиції у різноманітних рейтингах граматичної складності. Тому побудова правил для іменованих сутностей не є тривіальною задачею і алгоритм парсингу повинен опрацьовувати всі ці правила для вірної екстракції фактів.

Автоматизація буде успішно виконаною, якщо будуть взаємодіяти усі підсистеми.

ВИМОГИ ДО СИСТЕМИ

1.7 Вимоги до системи в цілому

Для успішного функціонування системи добування фактів потрібно, щоб працювали наступні підсистеми:

- підсистема токенізації;
- підсистема морфологічного аналізу включно з лематизацією;
- підсистема алгоритму парсингу Ерлі.

1.8 Вимоги до функціональних характеристик

Функціональні вимоги ставляться відповідно до варіантів використання системи користувачем:

- можливість надавати текст різними шляхами(введення власноруч та завантажувати файл) та повідомлення про неочікувану поведінку;
- можливість обирання типів фактів для добування, система повинна надавати користувачеві можливість комбінувати різні типи фактів для добування та повідомляти, що не було обрано жодного факту;

1.9 Вимоги до видів забезпечення

- процесор з тактовою частотою не нижче 2 ГГц;
- об’єм оперативної пам’яті (не менше 2 ГБ);
- жорсткий диск (не менше 40 ГБ);
- операційна система Ubuntu 18.04 АБО Windows 10 (та вище);
- Версія Python 3.6-3.8;
- Версія Streamlit 0.80.0;
- Версія MongoDB 4.2;
- Будь-яка версія VS Code або ж іншого текстового редактору.

СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

У таблиці 5.1 наведено календарний план робіт та терміни їх виконання.

Таблиця 5.1 – Календарний план робіт

№	Назва етапу розробки	Термін виконання
1	Визначення мети та задач розробки	10.04.2021
2	Вивчення рекомендованої літератури	12.04.2021
3	Пошук існуючих аналогів	13.04.2021
4	Аналіз існуючих методів розв’язання задач	18.04.2021
5	Постановка та формалізація задач	21.04.2021
6	Вибір технічних засобів	23.04.2021
7	Розробка системи	27.04.2021
8	Налагодження системи	29.04 2021
9	Оформлення пояснювальної записки	02.04.2021

ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

1.10 Види випробувань

Для забезпечення якості реалізації системи, необхідно провести наступні випробування:

- випробування згідно функціональних вимог, поставлених до системи.
- випробовування модульного тестування для розпізнавань різних типів іменованих сутностей.

1.10.1 Випробовування функціональних вимог

Сценарії для випробовування функціональних вимог після завершення розробки наведено у таблиці 6.1.

Таблиця 6.1 – Тестування функціональних вимог системи

Тест	Дія користувача	Очікуваний результат
Поле тексту при власноручному введені не пусте перед початком його аналізу	Запуск аналізу тексту при його відсутності	Діалогове вікно для користувача про відсутність тексту
Тестування формату файлу, що завантажує користувач на відповідність до одного з вказаних форматів	Спроба завантажити файл невказаного типу	Діалогове вікно для користувача про неправильний формат файлу
Тестування вхідного текстового файлу на вміст тексту в ньому	Спроба завантажити пустий текстовий файл	Діалогове вікно для користувача про відсутність тексту для аналізу

Продовження таблиці 6.1

Тест	Дія користувача	Очікуваний результат
Тестування відсутності вибору типів фактів для добування системою	Запуск аналізу тексту при відсутності вибору типів факту для добування	Діалогове вікно для користувача про відсутність вибору типів фактів
Тестування аналізу тексту при різних комбінаціях вибору типів фактів	Запуск аналізу тексту при бажаному типі факту для добування	Надання звіту до наявності факті у тексті

1.10.1 Випробовування на розпізнавання різних іменованих сутностей

Це випробовування буде проводитись відповідно до розроблених правил граматик системи.

UNICHECK

Ім'я користувача:
Попенко Володимир Дмитрович

ID перевірки:
1008082019

Дата перевірки:
30.05.2021 01:35:40 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
31.05.2021 00:28:49 EEST

ID користувача:
77149

Назва документа: **Turko_bachelor_is71**

Кількість сторінок: **74** Кількість слів: **10476** Кількість символів: **83781** Розмір файлу: **2.76 MB** ID файлу: **1008167672**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

8.8%
Схожість

Найбільша схожість: **2.9%** з Інтернет-джерелом (https://ela.kpi.ua/bitstream/123456789/39827/1/Khomenko_bakalavr.pd.

6.07% Джерела з Інтернету	120	Сторінка 76
7.63% Джерела з Бібліотеки	317	Сторінка 77

4.67% Цитат

Цитати	6	Сторінка 78

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

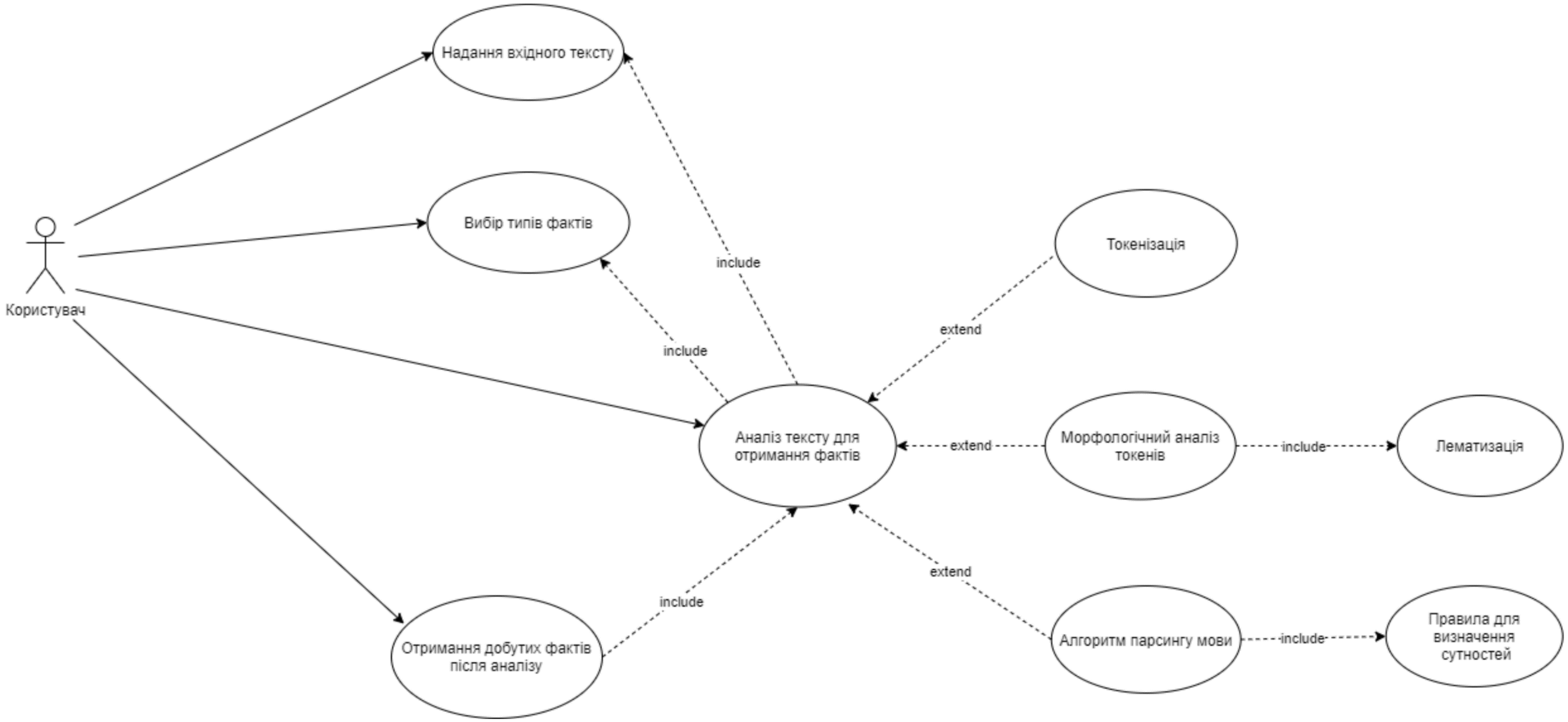
Замінені символи	12
Підозріле форматування	20 сторінок

Графічний матеріал
до дипломного проєкту

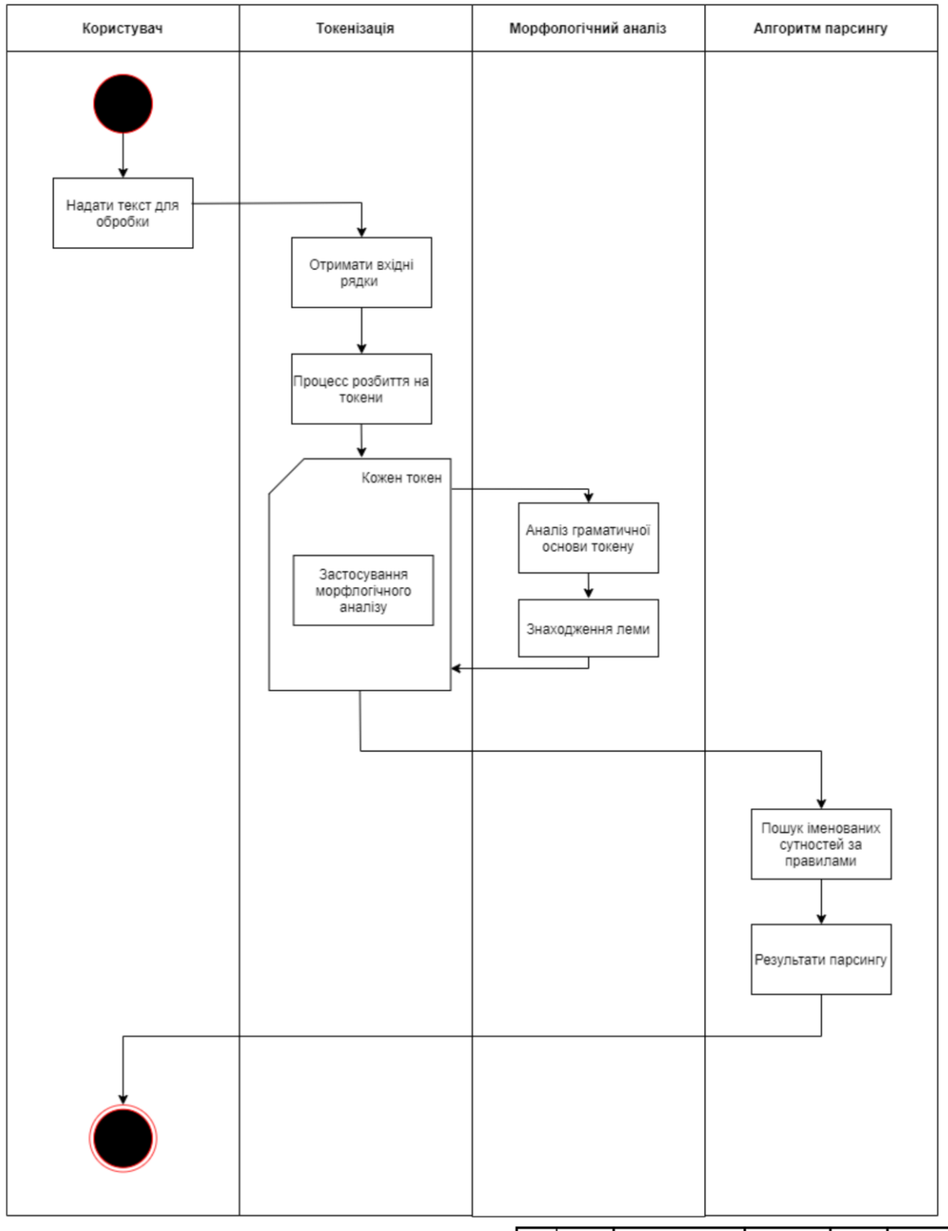
на тему: «Інформаційна система добування фактів з україномовних текстів»

Київ - 2021 року

ДП ІС-7328.1139-с.ССВ

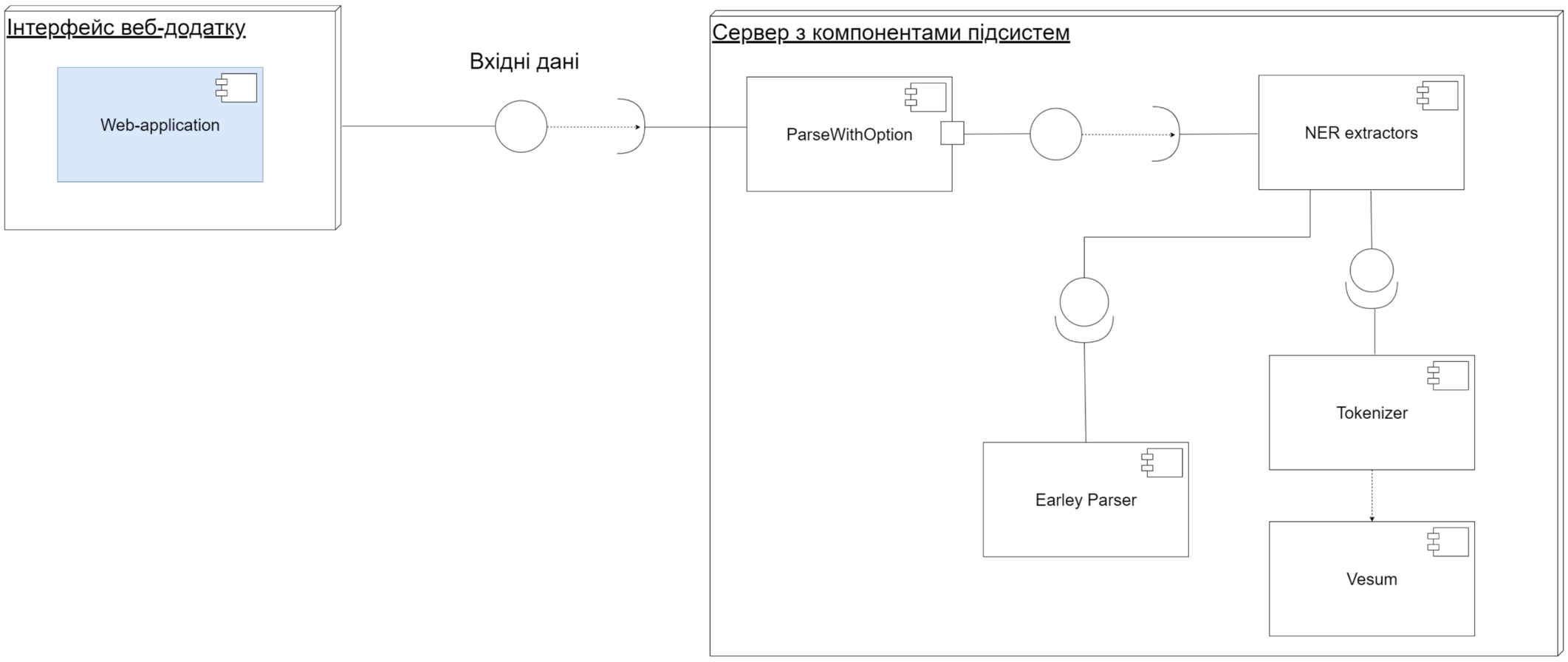


					ДП ІС-7328.1139-с.ССВ			
Зм.	Арк.	№ документа	Підпис	Дата		Літера	Маса	Масштаб
Розробив		Турко М. В.			Схема структурна варіантів використання			
Перевірив		Фіногенов О. Д.						
Т. кон.						Аркуш 1		Аркушів 1
					Інформаційна система добування фактів з україномовних текстів	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-73		
Н. кон.		Сперкач М. О.						
Затвердив		Фіногенов О. Д.						



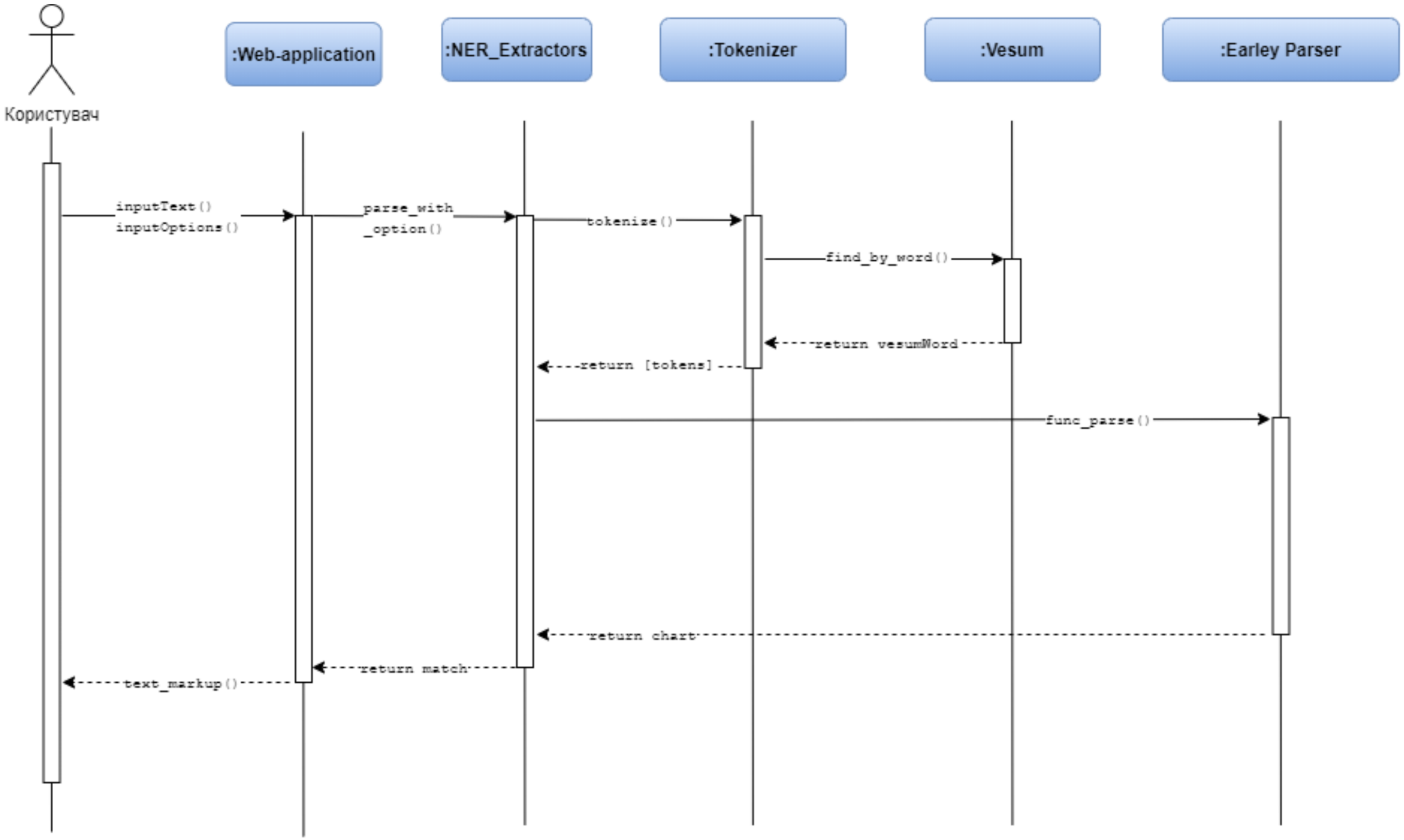
					ДП ІС-7328.1139-с.ССД	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна діяльності системи	Літера / Маса / Масштаб
Розробив		Турко М. В.				
Перевірив		Фіногенов О. Д.				Аркуш 1 / Аркушів 1
Т. кон.						
Н. кон.		Сперкач М. О.			*Інформаційна система добування фактів з україномовних текстів*	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-73
Затвердив		Фіногенов О. Д.				

ДП ІС-7328.1139-с.ССК



					ДП ІС-7328.1139-с.ССК		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна компонентів програмного забезпечення	Літера	Маса / Масштаб
Розробив		Турко М. В.					
Перевірив		Фіногенов О. Д.				Аркуш 1	Аркушів 1
Т. кон.							
Н. кон.		Сперкач М. О.			*Інформаційна система добування фактів з україномовних текстів*	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-73	
Затвердив		Фіногенов О. Д.					

ДП ІС-7328.1139-с.ССП

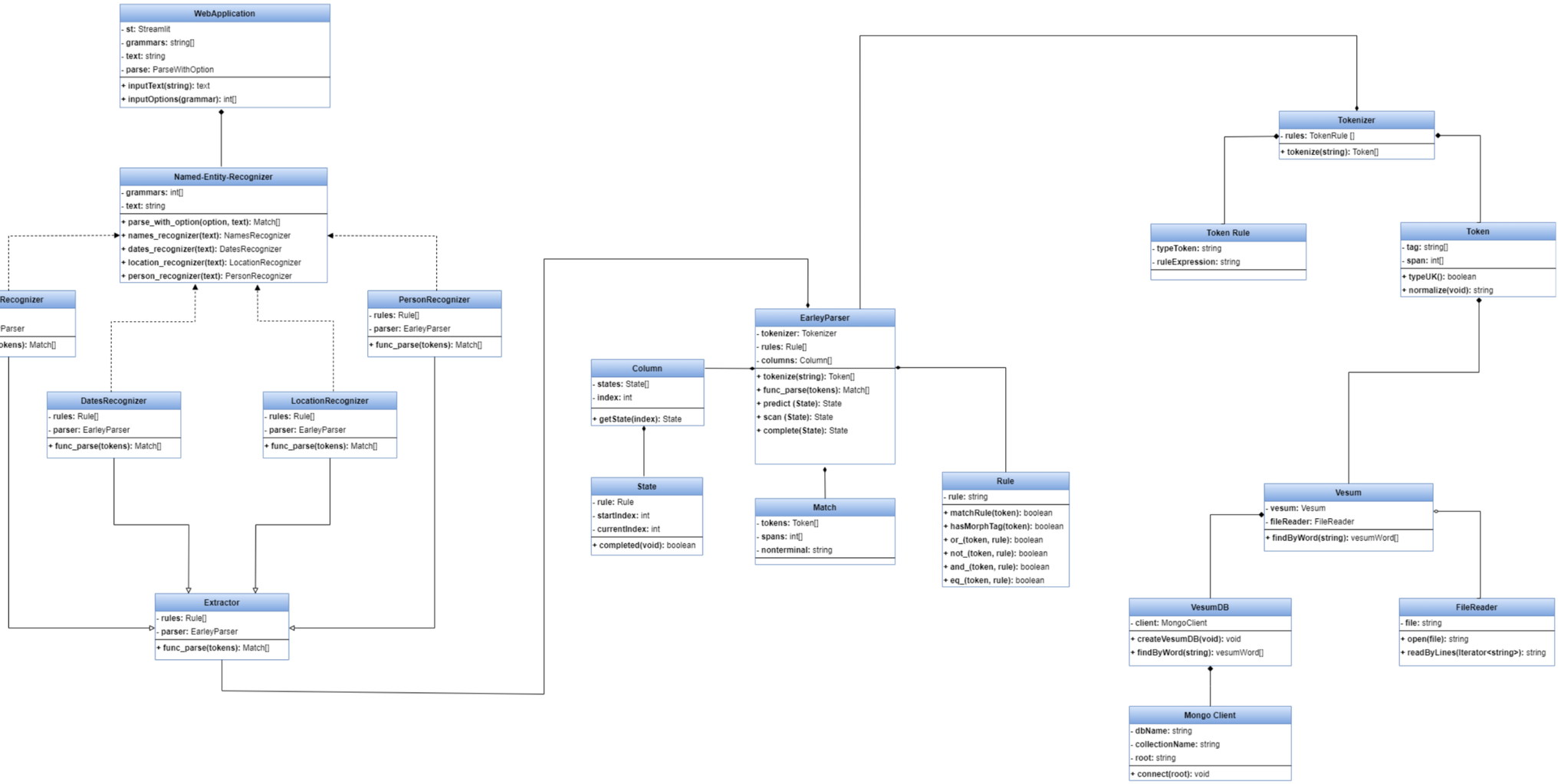


Зм.	Арк.	№ документа	Підпис	Дата	ДП ІС-7328.1139-с.ССП		
Розробив		Турко М. В.			Схема структурна послідовності		
Перевірив		Фіногенов О. Д.					
Т. кон.							
Н. кон.		Сперкач М. О.			*Інформаційна система добування фактів з україномовних текстів*		
Затвердив		Фіногенов О. Д.					

Літера	Маса	Масштаб
Аркуш 1		Аркушів 1

КПІ ім. Ігоря Сікорського
кафедра АСОІУ гр. ІС-73

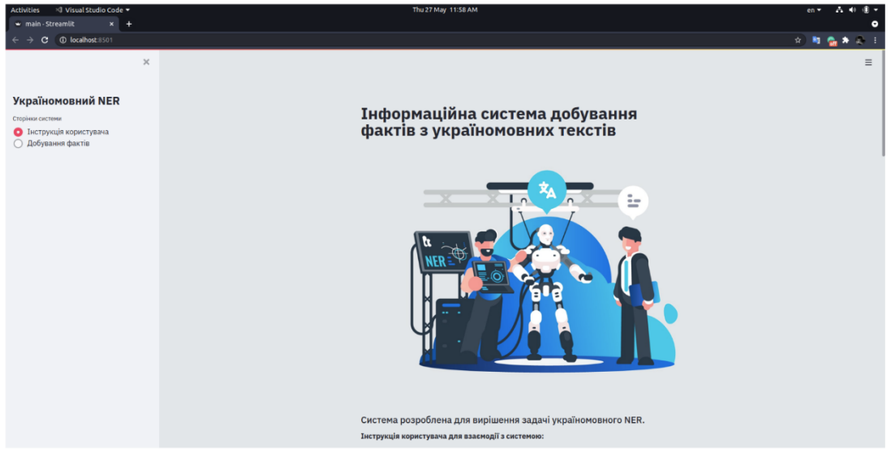
ДП ІС-7328.1139-с.ССК



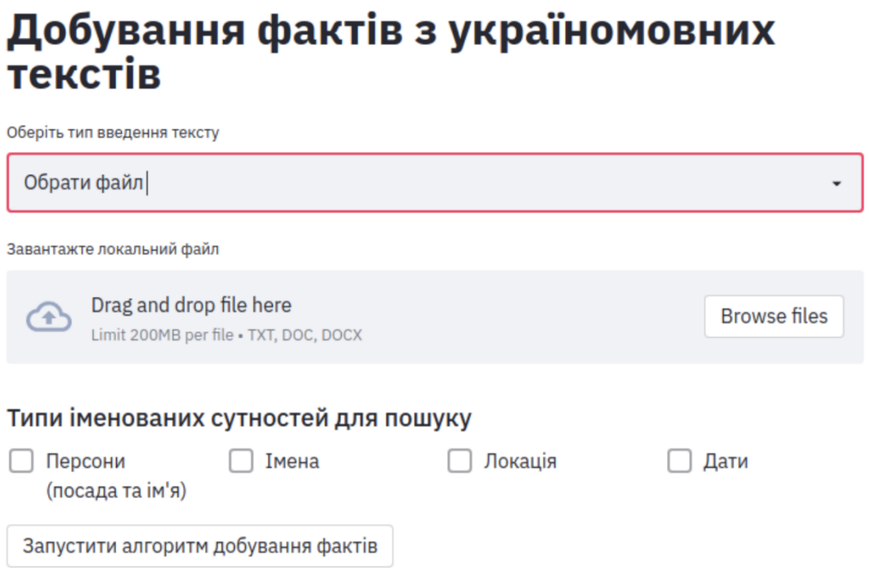
					ДП ІС-7328.1139-с.ССК	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна класів програмного забезпечення	Літера / Маса / Масштаб
Розробив		Турко М. В.				
Перевірив		Фіногенов О. Д.				Аркуш 1 / Аркушів 1
Т. кон.						
					Інформаційна система добування фактів з україномовних текстів	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-73
Н. кон.		Сперкач М. О.				
Затвердив		Фіногенов О. Д.				

ДП ІС-7328.1139-с.КЕ

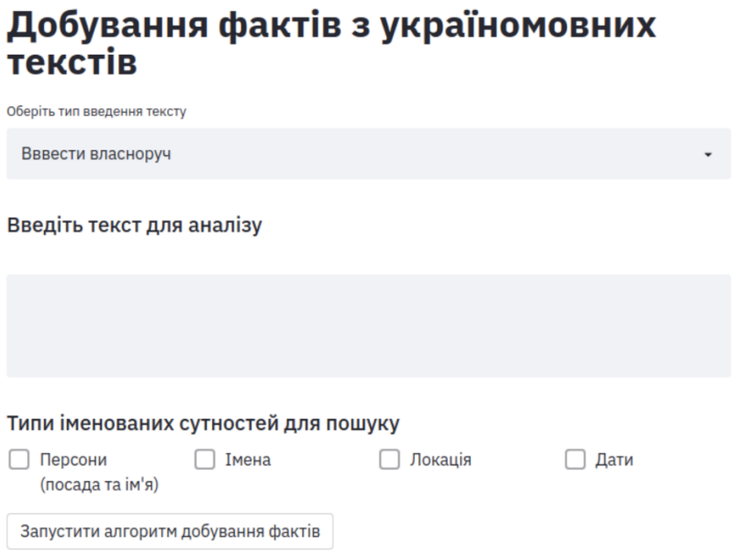
Екранна форма початкової сторінки з інструкцією для користувача



Екрана форма при виборі завантаженні файлу



Екрана форма при виборі введення тексту власноруч



					ДП ІС-7328.1139-с.КЕ	
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм	Літера / Маса / Масштаб
Розробив		Турко М. В.				
Перевірив		Фіногенов О. Д.				Аркуш 1 / Аркушів 2
Т. кон.						
Н. кон.		Сперкач М. О.			*Інформаційна система добування фактів з україномовних текстів*	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-73
Затвердив		Фіногенов О. Д.				

Суд залишив депутата від ОПЗЖ Віктора Медведчука, який підозрюється у держзраді, під домашнім арештом. Адвокатка Тетяна Козаченко розповіла в програмі "Сьогодні День" на телеканалі "Україна 24", як довго може розглядатися справа проти політика.

Типи іменованих сутностей для пошуку

☐ Персони (посада та ім'я) ☑ Імена ☑ Локація ☑ Дати

Запустити алгоритм добування фактів

Аналіз завершено!

Name

```
{
  "first" : "віктор"
  "last" : "медведчук"
}
```

Name

```
{
  "first" : "тетяна"
  "last" : "козаченко"
}
```

Location

```
{
  "name" : "україна"
}
```

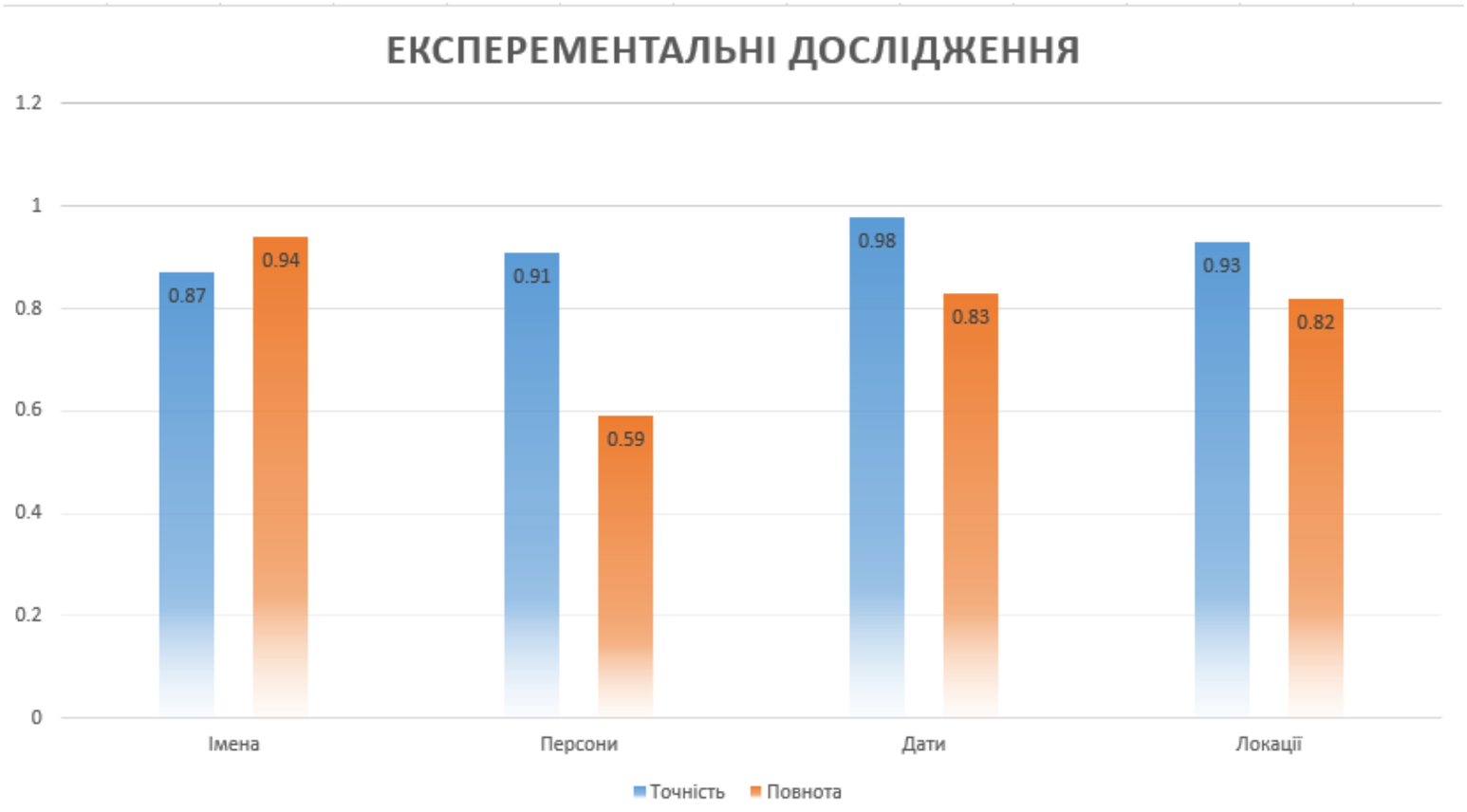
Суд залишив депутата від ОПЗЖ Віктора Медведчука Name , який підозрюється у держзраді, під домашнім арештом. Адвокатка Тетяна Козаченко Name розповіла в програмі "Сьогодні День" на телеканалі " Україна Location 24", як довго може розглядатися справа проти політика.

Екранна форма завершеного аналізу тексту та звіту для добутих фактів

					ДП ІС-7328.1139-с.КЕ	
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм	Літера / Маса / Масштаб
Розробив		Турко М. В.				
Перевірив		Фіногенов О. Д.				Аркуш 2 / Аркушів 2
Т. кон.						
Н. кон.		Сперкач М. О.			Інформаційна система добування фактів з україномовних текстів	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-73
Затвердив		Фіногенов О. Д.				

Рішення з математичного забезпечення

Дослідження точності та повноти
розпізнавання різних типів іменованих сутностей



Демонстраційний плакат до дипломного проекту
«Інформаційна система добування фактів з україномовних текстів»

Виконав студент гр. ІС-73 Турко М. В.

Керівник ДП Фіногенов О. Д.