Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

Проектування та розробка системи парсингу текстових документів

Текстова частина до курсової роботи

за спеціальністю «Інженерія програмного забезпечення»- 121

Керівник курсової роботи

Асистент
Корнійчук Максим Анатолійович

_____/підпис/

“_____” _____ 2021 р.

Виконав студент ІПЗ-4:

Мероник Тетяна Юріївна

_____/підпис/

“_____” _____ 2021 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем,
к. ф.-м. н. О. П. Жежерун

_____

(підпис)

“____” _____ 2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Мероник Тетяні Юріївні факультету інформатики 4-го курсу

ТЕМА Проектування та розробка системи парсингу текстових документів

Вихідні дані:

Аналіз особливостей та етапів побудови парсера для текстових документів

Застосунок, що виконує парсинг судових документів

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

Розділ 1: Парсинг даних

Розділ 2: Етапи побудови парсера для текстових документів

Розділ 3: Реалізація парсера судових документів

Висновки

Список літератури

Дата видачі „___” _____ 2021 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Зміст

Тема: Ігровий штучний інтелект – підходи до побудови

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	09.10.2020	
2.	Пошук літератури за темою роботи	10.10.2020	
3.	Ознайомлення з науковою літературою	11.10.2020	
4.	Ознайомлення зі структурою та особливостями вхідних даних для практичної частини	21.10.2020	
5.	Проектування результуючих структур парсингу	25.10.2020	
6.	Розробка парсера	08.12.2020	
7.	Написання першої частини курсової роботи	08.02.2021	
8.	Написання другої частини курсової роботи	04.03.2021	
9.	Написання третьої частини курсової роботи	20.03.2021	
10.	Внесення змін до роботи відповідно до коментарів керівника	08.04.2021	
11.	Створення презентації	09.04.2021	
12.	Подача курсової роботи на перевірку	11.04.2021	

Студент Мероник Т. Ю.

Керівник Корнійчук М. А.

“_____”_____

Анотація

Метою даної курсової роботи є вивчення поняття парсингу, етапів та особливостей побудови системи парсингу.

В роботі описане поняття парсингу, його особливостей. Проаналізовані етапи його проектування та побудови. Також описана реалізована система парсингу судових документів, що демонструє застосування даної технології для аналізу текстових документів.

Вступ

Ми перебуваємо в епоху глобальної цифрої трансформації, яку ще часто називають Четвертою науково-технічною революцією, коли дані, що раніше були створені для читання людиною, повинні бути пристосовані для обробки та аналізу комп’ютерами. Люди очікують від пристроїв розуміння все більших об’ємів та складнішої інформації. Проте між наданням вхідних даних та їхнім аналізом є ще один дуже важливий крок – перетворення цих даних в формат, який зрозумілий для комп’ютера. Таке перетворення можливе за допомогою парсингу. Парсинг – це процес аналізу тексту з метою перетворення його в структуру, яка складається з певних токенів, і які зрозумілі для комп’ютера. За допомогою цього процесу відбувається перетворення інформації з вигляду, в якому вона зрозуміла людині, у формат, який може бути машинно опрацьований.

Мета дослідження – вивчити поняття парсингу, етапи та особливості побудови системи парсингу.

Для досягнення мети необхідно виконати наступні етапи:

- Вивчення поняття парсингу
- Дослідження особливостей парсингу та етапів побудови системи парсингу текстових документів
- Реалізація системи парсингу судових документів. Основним завданням цього етапу є приклад обробки парсером документів, розбиття їх на структуру та дослідження практичного застосування такого розбиття.

Об'єкт дослідження – парсинг текстових документів.

Предмет дослідження – проектування та побудова системи парсингу текстових документів.

Розділ 1: ознайомлення з процесом парсингу, засобами, використовуваними текстових аналізаторів.

Розділ 2: аналіз етапів проектування та побудови парсера для текстових документів. Описано можливі особливості вхідних даних та способи створення парсерів, які покривають всі edge кейси – так звані граничні стани.

Розділ 3: демонстрація практичного застосування парсингу текстових документів на основі реалізації модуля парсингу документів для пошуковика по судових рішеннях.

1. Парсинг даних

1.1 Поняття парсингу

Дані в сучасному світі – це одна з найцінніших складових, якими можна володіти. Маючи необхідні дані, можна їх обробити і отримати прогноз економічного розвитку країни на наступні 50 років, аналіз діяльності фірми з визначенням її слабких місць та способів підвищення продуктивності... Але для обробки великих масивів даних потрібне застосування комп'ютерів та спеціального програмного забезпечення, яке здатне здійснити аналіз та генерувати очікуваний результат, так як людина не спроможна працювати з таким об'ємом даних і надавати відповідь у короткий проміжок часу.

На етапі передачі даних в комп'ютерну систему виникає проблема – як зробити так, щоб інформація, сформульована та записана людиною, була зрозуміла машині? Адже те, як ми сприймаємо інформацію, значно відрізняється від того, як її «розуміє» комп'ютер. Те, що для нас здається текстовим документом з чіткою структурою та окремими зрозумілими блоками даних, для комп'ютера просто набір символів. В таких випадках необхідна попередня структуризація документів. Це можна досягти за допомогою такого процесу як парсинг.

Парсинг – це процес перетворення тексту в структуру даних. Парсинг – це досить широке поняття і може означати будь-який рівень аналізу від простого розбиття тексту на частини до повного NLP (Natural Language Parsing), який за допомогою штучного інтелекту «розуміє» людську мову. Звичайно, від детальності та складності необхідної структури результату, залежить кількість ресурсів, які будуть використані в процесі аналізу тексту. Тому в залежності від призначення, існують різні види парсерів, які задовольняють потреби системи.

1.2 Термінологія парсингу

Парсинг складається з багатьох процесів, складових, тому для повного розуміння роботи парсера необхідно володіти термінологією. Нижче описані найпопулярніші терміни, що використовуються в парсингу.

Парсинг – це застосування правил, визначених користувачем, для валідації вхідних даних і перетворення цих даних в структуру для подальшого використання.

Токен – це окрема одиниця інформації, що викремлюється з тексту під час аналізу з застосуванням правил. Для виокремлення токенів використовуються синтаксичні чи семантичні аналізатори. Токенізація – це процес розбиття тексту на токени з застосуванням правил.

Класифікація – це процес визначення типу токену за допомогою семантичного аналізу, наприклад «дата» чи «адреса».

Лексичний аналіз – це процес співставлення лінійної послідовності лексем мови з її граматикою. Результатом є парсингове дерево.

Регулярний вираз – це паттерн, що використовується аналізатором для співставлення з вхідним текстом. Вираз формується за допомогою літералів та операторів. За допомогою регулярних виразів легко виділяти з тексту дати, адреси, номери телефонів та інші структурні одиниці, що мають визначений формат.

Синтаксичний аналіз – це наступний крок після лексичного аналізу. В результаті лексичного аналізу ми отримуємо певну послідовність токенів. Під час синтаксичного аналізу токенам надається значення.

1.3 Засоби парсингу

В основі парсингу лежить набір правил, які застосовуються до вхідних даних допоки є можливість їх застосовувати і отримані токени не є термінальними.

Для того, щоб підготувати текст до парсингу, потрібно провести лексичний аналіз. На етапі лексичного аналізу з потоку тексту, який прийшов на вхід виділяються токени. В багатьох випадках у системах, що працюють з текстовими документами, вхідні дані створені з допомогою мови розмітки. Сьогодні існує безліч бібліотек для різних, які дають можливість працювати з такими файлами і перетворювати їх у список елементів, розбиваючи їх за тегами, ідентифікаторами, та опрацьовуючи їх як дерева елементів, пересуваючись як до батьківського елементу, так і до дочірніх.

Далі необхідно визначити набір правил, які будуть застосовані до отриманого результату, щоб сформувати фінальну структуру.

Для реалізації парсингових правил необхідно визначити за якими параметрами буде здійснюватися пошук токена.

Зазвичай в результаті парсингу текстового документа утворюється дерево малої глибини.

Для створення синтаксичного аналізатора доступні також готові бібліотеки. Наприклад, для Python – це Lark. Ця бібліотека дає можливість задавати граматику і застосовувати її до вхідних даних. Приклад такої граматики зображено на рисунку 1.1.

```
parser = Lark('''?sum: product
                    | sum "+" product   -> add
                    | sum "-" product   -> sub

                ?product: item
                    | product "*" item  -> mul
                    | product "/" item  -> div

                ?item: NUMBER           -> number
                     | "-" item         -> neg
                     | "(" sum ")"

                %import common.NUMBER
                %import common.WS
                %ignore WS
            ''', start='sum')
```

Рисунок 1. 1 Генерація парсера за допомогою бібліотеки Lark [9]

Також для імплементацій правил розпізнавання токенів потужним інструментом є регулярні вирази.

Вони використовуються у випадках, коли термінальний токен має певний унікальний паттерн, який можна записати за допомогою пунктуації регулярних виразів.

Так за допомогою регулярних виразів можна виявити токен- український номер телефону ^\+?3?8?(0\d{9})$.

1.4 Прикладне застосування парсингу

Аналіз тексту використовується майже у всьому програмному забезпеченні, яке якимось чином взаємодіє з даними, що користувач подає на вхід.

Напевно, найбільш поширеним прикладом застосування текстових аналізаторів для сфери розробки є компілятор. Компілятор – це програма, що перетворює набір інструкцій/тверджень, що задає користувач використовуючи одну з мов

програмування – код програми, на машинну мову, таку, яку комп’ютер здатний опрацювати і виконати. Розробник застосунку пише код рядок за рядком, і коли наступає час подивитися на результати роботи цього застосунку, програма повинна бути оброблена компілятором.

Першим кроком є лексичний аналізатор. Він використовує певний набір правил, щоб перетворити текст на послідовність токенів. Під час цього етапу з коду видаляються зайві пробіли, коментарі та інші символи, які є некорисними з точки зору впливу на результат.

Другим кроком є синтаксичний аналізатор. Він отримує результат лексичного аналізатора – послідовність токенів – і застосовує його відносно правил продукції, здійснює всі підстановки доводячи результат до стану, коли всі токени є термінальними. Цей етап відповідає за виявлення помилок та побудову парсингового дерева, у якого всі «листки» - кінцеві ноди є термінальними. [8]

Але процес парсингу використовується не лише в сфері розробки програмного забезпечення. Навіть калькулятор повинен застосувати певні правила, щоб перетворити вхідних рядок в дерево парсингу, яке надає змогу здійснити обчислення виразу, враховучи пріоритети операції та інші особливості обрахунків.

Парсинг текстових документів застосовують і компанії, якщо вони використовують автоматизовану систему ведення звітності прибутків та витрат. Всі чеки/накладні/квитанції, що завантажуються в систему повинні бути оброблені і перетворені в структури, що містять інформацію про список транзакцій.

Отже, якщо у застосунку, який приймає на вхід текстовий файл або рядок, функціонал більший, ніж зберегти його на диску, а потім за назвою повернути в такому ж виді, є можливість фільтрації, сортування чи пошуку, то можна бути впевненим, що даний застосунок виконує парсинг тексту, складність і глибина якого залежить від цілей, які поставлені перед ним.

2. Проектування та розробка парсера текстових документів

2.1 Аналіз вхідних даних для парсингу

Для того, щоб знати як проектувати парсер, необхідно дослідити текст, який буде слугувати вхідною інформацією.

Під час даного дослідження треба здійснити наступні оцінки:

- формат даних
- види вхідних документів
- однорідність вхідних даних в документах одного виду

Під форматом даних мається на увазі розширення текстових документів, структура тексту, наявність розмітки, за її наявності здійснити аналіз на присутність допоміжних ідентифікаторів, які вказують на тип інформації, що знаходиться у файлі.

Види вхідних даних – це різновиди документів за змістом. В залежності від сфери застосування парсера, вхідні документи можуть бути як одного виду так і різні. Наприклад, можна створити парсер, який буде обробляти резюме кандидатів на посади в фірму, які створені за визначеним шаблоном. Тоді всі документи мають однакові структурні елементи. А може бути парсер для аналізу посвідчень особистості, таких як паперовий паспорт, ID-картка, студентський квиток, посвідчення водія і т д. В цьому випадку документи значно відрізняються за форматуванням, інформацією, яку вони містять.

Однорідність вхідних даних в документах одного виду визначає наявність так званих edge-cases (межеві випадки). Для того щоб виявити такі випадки, необхідно проаналізувати кілька документів одного виду. При аналізі порівнюється структура цих документів і здійснюється пошук структурних

відмінностей. До таких відмінностей належать різні формати запису даних, наприклад, відсутність однозначності в форматі дат – «13.02.21» чи «13 лютого 2021», чи номерів телефонів, відсутність певної структурної одиниці (електронна адреса може бути вказана в резюме або ні). Також можливі варіації написання назв пунктів чи розділів – можна писати все слово великими буквами, ставити пробіли між буквами, виділяти стилями... Всі ці особливості повинні бути виявлені і взяті до уваги на етапі розробки парсера, щоб не виникло ситуацій, коли розпізнається неправильний елемент тексту за токен чи упускається важлива інформація, а в найгіршому випадку – система ламається через відсутність алгоритму обробки такої ситуації.

Для зручності подальшого проектування парсера за всіми дослідженими характеристиками та особливостями вхідних даних необхідно формувати певний звіт, на який можна буде посилатися на наступних етапах.

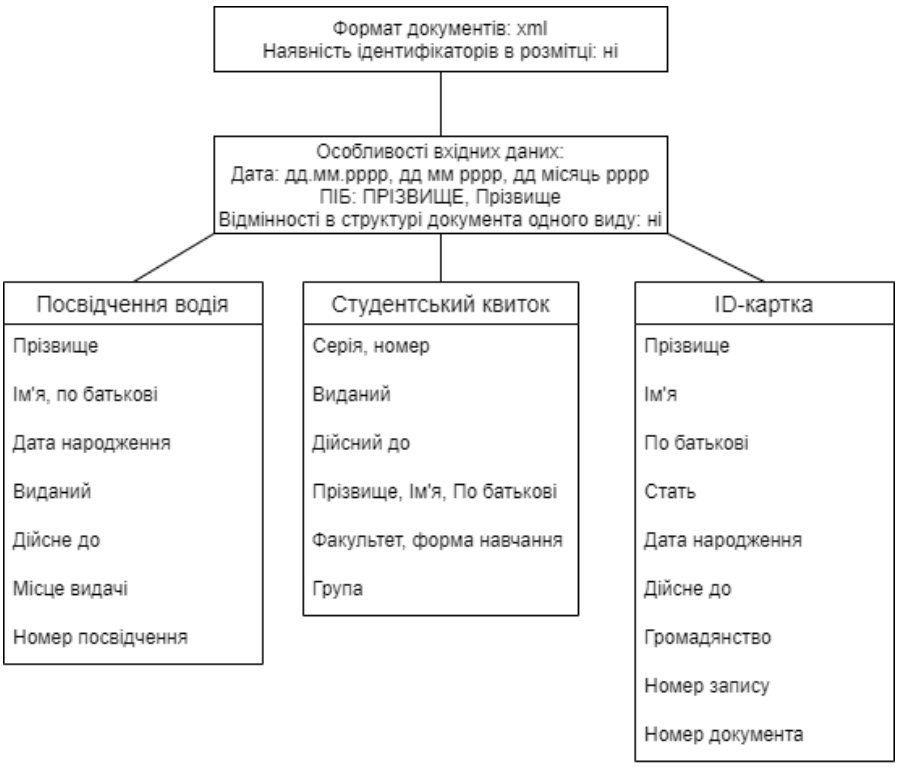


Рисунок 2. 1 Приклад результату аналізу вхідних даних для парсера посвідчень особистості

Після того як створена чітка характеристика масиву вхідних даних, можна переходити до наступних етапів проектування та розробки парсера.

2.2 Визначення результуючої структури

Після того, як проведений аналіз вхідних даних, потрібно визначити, який повинен бути результат роботи парсера.

Для цього необхідно дослідити предметну область, для якої розробляється даний парсер, вивчити випадки, в яких він буде застосовуватися. Після цього проводиться аналіз інформації, яка повинна бути виділена з вхідних даних. Наприклад, якщо відбувається аналіз посвідчення особистості для підтвердження віку, то нема потреби розбивати документ за всіма структурними елементами, достатньо виділити ПІБ та дату народження. Проте, якщо це державна установа, яка проводить оцифрування документів, то всю інформацію, що міститься в вхідних даних, необхідно помістити в результуючу структуру.

Часто, весь об'єм вхідних даних зберігається, проте змінюється формат і блоки розбиваються на токени, які містять найменшу самостійну частину інформації. Такий приклад наведений на рисунку 2.2, де блок особистих даних розбито на окремі токени за змістом та типом даних.

Рисунок 2. 2 Процес аналізу вхідних даних і створення результуючої структури

2.3 Розробка парсера

Для розробки парсера необхідно визначити набір правил парсингу.

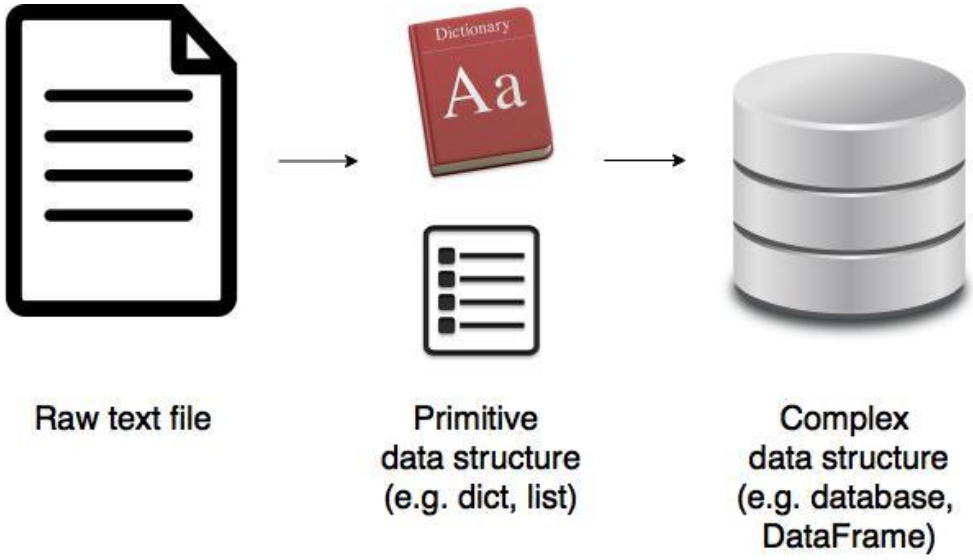


Рисунок 2. 3 Процес парсингу вхідних даних

Дані правила є своєрідними контекстними підказками для алгоритму, за якими документ опрацьовується розбивається на токени.

Після виконання попередніх двох кроків проектування – аналізу вхідних даних та визначення структури, яка повинна утворитися в результаті роботи алгоритму, необхідно виділити паттерн кожного токену.

Серед типових способів розпізнавання токенів є:

- Визначений індекс(позиція) розташування елемента у документі
- Довжина токену
- Формат запису(для дат, ідентифікаторів, кодів)
- Розпізнавальні слова/символи
- Розташування відносно іншого токена

Розпізнавання за індексом. Часто файли починаються або закінчуються на певний набір токенів. При наявності такого універсального формату можна визначити, який номер рядка, або позицію в рядку займає початок даного токена. Якщо відомо його довжину, то для визначення його завершення, ці 2 значення додаються, або ж використовується інший спосіб. Документи як ID-картка повністю складаються з токенів, які розміщені за координатами.

Довжина токену. Даний спосіб використовується в комбінації з іншими. Він корисний при потребі визначити одну з кінцевих координат токена за умови відомої першої координати.

Формат запису. Це певний паттерн, який визначає даний токен. Паттерн - це узагальнена схема запису даного токена. Він складається з послідовності символів певного типу і довжини, яка є унікальною для конкретного шуканого елемента. Для того, щоб знайти токен за паттерном, використовують регулярні вирази, які описані в розділі 1.4.

Розпізнавальні слова/символи. Даний спосіб частково схожий на попередній, проте він більш примітивний і вимагає не паттерн всього токена, а лише унікальну

комбінацію, яка передує чи слідує за токеном. Номер справи може починатися символом «№».

Розташування відносно іншого токена. Цей спосіб використовує абсолютну позицію іншого токена та відносну позицію даного токена відносно нього.

Після того, як для кожного токена було підібране відповідне правило, здійснюється підбір засобів, які можуть бути використані для імплементації даних правил і починається процес розробки алгоритму парсингу. Серед засобів, що можуть бути використані, можуть буи регулярні вирази та готові бібліотеки, що розроблені спеціально для таких завдань.

Є токени, для яких одне правило не є вичерпуючим – не покриває всі випадки, так як будова таких токенів не є достатньо чіткою, унікальною чи універсальною. Не рідко приходиться поєднувати кілька варіантів правил для одного токена для підвищення точності процесу аналізу тексту.

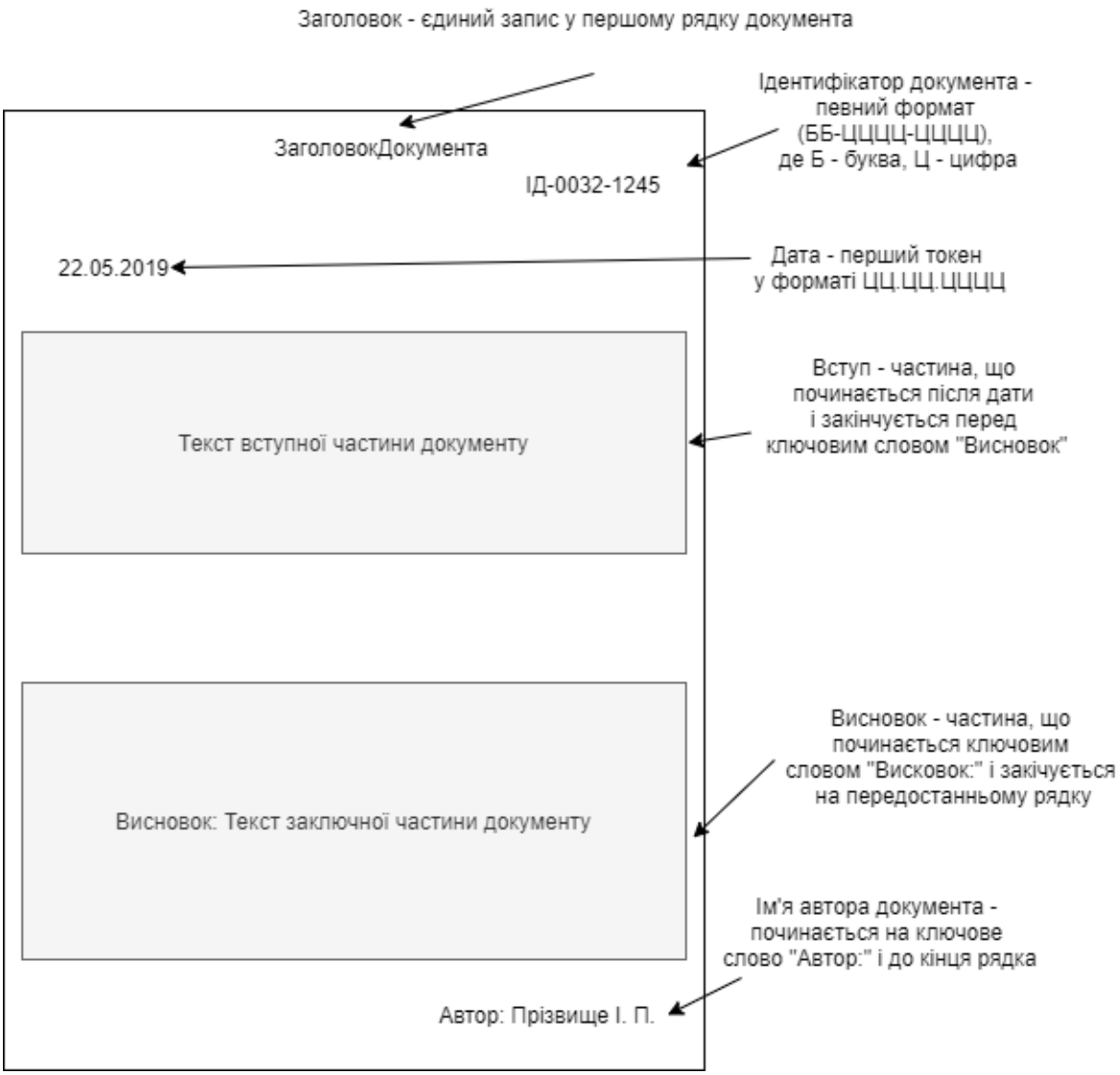


Рисунок 2. 4 Аналіз текстового файлу на правила парсингу

Під час розробки парсера проводиться тестування на невеликому датасеті. І хоча цього достатньо, щоб переконатися в робочому стані парсера і його здатності створювати необхідні структури, це не гарантує 100% успіх на повному датасеті. Тому обов’язковим кроком є додавання певного способу фіксації результатів парсингу для тестування. Наприклад створення журналу логування при парсингу цілого датасету, куди будуть вноситися всі аномалії, які зафіксовані під час парсингу, разом з ідентифікатором(назвою) документа, в якому вони зустрілися. До таких аномалій належать відсутність певного обов’язкового токена,

невідповідність числових значень очікуваному діапазону тощо. Також для швидшого знаходження потенційних помилок потрібно поступово збільшувати об’єм датасету. Адже парсинг всього датасету може зайняти години, чи навіть дні, в залежності від його розміру, складності структури та фізичних характеристик комп’ютера. Чим раніше будуть зафіксовані та виправлені помилки, тим менш ресурсозатратним буде процес парсингу датасету.

3. Розробка парсера для пошуковика судових рішень

3.1 Опис проекту

Для демонстрації прикладного застосування парсингу текстових документів було реалізовано модуль парсингу судових документів для пошуковика по судових рішеннях, розробленого командою ThemidaDevs. Сам пошуковик розміщений на сервері за посиланням: https://themida-devs-kma.herokuapp.com/ .

При виборі теми для розробки було проаналізовано правову сферу та проблеми, що виникають при автоматизації процесів. Шляхом аналізу доступних продуктів було зроблено висновок, що пошук по судових документах реалізований з багатьма дефектами, працює повільно, а відображення документів є незручним. Тому було прийнято рішення створити власний пошуковик.

Отже, ThemidaDevs – це пошуковик по судових документах. Користувач може вводити запит, який складається як з частини слова, так і з багатьох слів, задавати фільтри за типом справи, документа і т.д. Для покращення користувацького досвіду імплементовано групування документів за номером справи, сніппети для кожного документу в результаті пошуку і пошук з пріоритетом. Для виставлення пріоритетів був проведений аналіз структури документу, визначені його найважливіші елементи, і визначена їхня важливість у пошуку. Для отримання можливості застосувати ці коефіцієнти пріоритету до пошукового двигуна, потрібно було розбити документ на частини, тому це одна з функцій, для якої було необхідно реалізувати парсер.

При перегляді документу завантажується не суцільний текст, а також фронтенд отримує структуру в вигляді json-об’єкту, яка містить всі структурні елементи судового рішення. Таким чином вдається створити сталий вигляд для всіх документів незалежно від їхнього оригінального форматування.

Кожен тип судового рішення має власну структуру, різну кількість структурних одиниць та інші ключові слова. Розроблений модуль, що відповідає, за парсинг документів, включає в себе визначення типу документу. Це зробити не складно, так як цих типів є визначена кількість і їхня назва вказана в документі в певному форматі. Шляхом аналізу документів одного типу була визначена їх структура, особливості, та розроблений парсер для різних типів документів, що доступні в системі. Після парсингу частини документа формують в список, який додається до списку раніше оброблених документів і передається в роботу ElasticSearch для подальшої роботи.

3.2 Вибрані інструменти розробки

ThemidaDevs – це веб-застосунок.

Для написання серверної частини застосунку здійснювався вибір між мовами програмування Java та Python. Так як важливу роль в даному застосунку відіграє ElasticSearch, який імплементований на Java, то і має сенс написання серверу застосунку на Java. Проте після глибшого аналізу всіх необхідних процесів, які повинні бути реалізовані в застосунку для обробки тексту, дослідження доступних бібліотек для обробки тексту, було встановлено, що використання Python дає доступ до залучення більшої кількості корисних бібліотек. Для обох мов доступні офіційні клієнтські API та документація[5]. В роботі обидва варіанти реалізації дають схожі результати. Єдиною перевагою використання Java є швидше оновлення клієнта ElasticSearch, проте в межах даного дослідження ця перевага не є визначальною. В результаті цього аналізу було обрано Python.

Для зручності реалізації бекенду для веб-застосунку було використано Flask – веб фреймворк. Flask – це модуль Python, який має мале, проте легко розширюване ядро. Хоча він вважається мікро фреймворком, так як не забезпечує застосунок вбудованим ORM менеджером, валідацією введених даних, чи

верифікації користувачів, такий обмежений функціонал є швидше перевагою. Flask має багато базових корисних функцій, таких як URL-роутинг, застосування шаблонів. Інші необхідні для конкретного застосунку можливості можна додати за допомогою розширень і сторонніх бібліотек, що забезпечує економію ресурсів, так як в такому випадку застосунок не містить модулів, які не використовуються. Flask забезпечує недвозначність коду, що підвищує його читабельність. В порівнянні з іншими веб фреймворками для Python, наприклад Django, Flask не вимагає довгого навчання, так як код, який пишеться за допомогою даного фреймворку дуже наближений до того, що написаний на чистому Python.

Для реалізації пошуку було використано ElasticSearch – розподілений пошуковий та аналітичний двигун для різних типів даних. Створений на Apache Lucene, він надає ряд REST API для клієнтів, реалізованих на різних мовах програмування. Функціонал, реалізований в ElasticSearch робить його корисним при реалізації пошуку в мобільних, десктопних та веб застосунках, логуванні аналітики, моніторингу роботи застосунку та багатьох інших можливостей.

Beautiful Soup – це бібліотека Python, що використовується для обробки документів HTML, XML та інших мов розміток. Вона дає можливість отримати інформацію з цих документів, позбавляючи їх тегів розмітки, зберігаючи лише контент. Застосування цієї бібліотеки дає змогу обробляти текст, сприймаючи його як дерево. Реалізована можливість доступатися до елемента за класом, тегом чи ідентифікатором, пересуватися від елемента до його батька чи дітей, замінювати контент, додавати чи видаляти елементи. Таким чином можна здійснювати аналіз тексту та редагувати його ще на етапі читання. Каталог складається судових документів формату HTML, тому для їхнього опрацювання було обрано дану бібліотеку.

Не завжди достатньо розбити текст за тегами, які вже наявні в ньому. Деколи потрібно виділити токени з вже виділених елементів. Цей процес

допомагає зробити легшим таке поняття як регулярний вираз. Як вже було зазначено в розділі 1.2 Термінологія парсингу, регулярний вираз, це певний паттерн, який характеризує токен і може бути виділений з масиву тексту. Для використання регулярних виразів було додано модуль re до проекту. Цей модуль є в стандартній бібліотеці Python. Його визначною характеристикою є широкий функціонал та можливість використовувати сторонні модулі парсингу регулярних виразів без виникнення конфліктів і неоднозначностей.

Для розробки клієнтської частини застосунку був використаний Jinja версії 2. Jinja – це мова шаблонізації загального призначення, бібліотека для проектів на Python. Її використання робить розробку швидкою, безпечною та гнучкою. Використання Flask на бекенді дозволяє зручно передавати необхідні параметри клієнту та генерує контент в залежності від отриманих даних.

Для зв’язку з серверною частиною застосунку використано ajax – Asynchronous JavaScript and XML. Ця технологія дозволяє асинхронно надсилати запити до сервера, отримувати відповідь та відображати зміни без потреби перезавантажувати сторінку. Запити на пошук, фільтрацію та відображення документів створюються саме за допомогою ajax.

Для дизайну та розмітки був використаний фреймворк Bootstrap.

3.3 Дослідження датасету

3.3.1 Визначення типів документів та їх структури

Датасет, який повинен бути оброблений парсингом, складається з судових рішень. Судове рішення в широкому значенні – це акт судового розгляду будь-якого типу провадження. Проте рішенням називається і один з типів судового рішення. Для уникнення непорозуміння, термін «судове рішення» в широкому значенні буде замінене на «судовий документ», який в контексті одиниці датасету є коректним.

Розмір датасету – 240 000 документів. Кожен з документів належить до одного з наступних типів: рішення, ухвала, вирок, постанова та наказ.

Всі типи судових документів мають логічно визначену структуру, яка є універсальною.

Кожен документ починається вступною частиною. В ній міститься набір даних, які ідентифікують даний документ. В нього входить дата ухвалення, населений пункт, точна повна назва суду та прізвище та ініціали учасників судового процесу(суддів та секретаря судового процесу).

Друга частина – описова. В ній вичерпно описана позиція сторін судового процесу. І об’єктивний опис всіх виявлених доказів. Ця частина повинна не містити жодного оцінювального ставлення суду до будь-чого, описаного в ній.

Третя частина – мотивувальна. В цій частині суд описує обставини, які були виявлені на основі наявних доказів, описує коректність наведених доказів та аргументує їх використання/ігнорування на етапі прийняття рішення.

Остання частина – резолютивна. В ній суд дає відповідь на кожну вимогу позивача. Отже, ця частина – це присуд суду, в якому вказуються дії, які повинні бути виконані після набуття провадження законної сили, і термін на їх виконання.

3.3.2 Недоліки та особливості датасету

Датасет складається з документів, що сформовані в судових організаціях на території всієї України. Також ці документи створені в різні роки. Вимоги до оформлення рішень змінювалися, тому змінювалися і шаблони. Ці фактори сприяли утворенню багатьох невідповідностей між структурами різних документів одного виду.

При аналізі судових документів було виявлено, що номер справи не є унікальним, хоч він і повинен бути ідентифікатором. Згідно з наказом Державної Судової Адміністрації України від 17.12.2013р., кожній судовій справі

присвоюється унікальний номер, який залишається незмінним впродовж всіх заходів судочинства по цій справі. Формат даного ідентифікатора виглядає наступним чином: код суду/порядковий номер справи в цьому році/номер року. Проте в 28.09.2018р цією ж адміністрацією було затверджено новий класифікатор судів, і як наслідок деякі суди отримали однаковий код. У зв’язку з цим, покладатися на номер справи як на ідентифікатор не можна.

Також великі труднощі виникли при дослідженні формату написання токенів. Наприклад, ключові слова, за якими визначається тип справи, мають різне написання в залежності від суду, в якому цей документ було сформовано. Рисунок 3.1 і рисунок 3.2 демонструють те, наскільки відрізняються документи одного виду.

Номер справи. Хоч формат самого номера справи однаковий, проте в першому випадку він розміщений в правому верхньому куті документа і йому передує «номер провадження справи», а в другому у вступній частині з передуючим «справа №». В деяких документах номер справи написаний взагалі без будь-яких розпізнавальних позначок. А так як в деяких документах дата написана в форматі дд/мм/рррр, то без них важко відрізнити номер справи від дати початку провадження.

Дата прийняття рішення. В першому випадку дата написана повністю числами в форматі дд.мм.рррр, а в другому дд місяць рррр, де місяць написаний словом. Через те, що виявлено різні формати написання дати, зроблено висновок, що немає вимоги щодо єдиного формату дати, тому інші формати можуть з'явитися в майбутньому і потрібно прописати edge cases для всіх можливих форматів, щоб в подальшому не виникало ситуацій, коли дати не розпізнаються.

Тип справи. Тип справи може бути написаний всіма великими літерами без пробілів або з пробілами. В даних прикладах це «РІШЕННЯ» та «Р І Ш Е Н Н Я».

Вступна частина. В одному з файлів відсутні відступи після ключових слів, коли в іншому вони чітко виділені жирним шрифтом, з двокрапкою після ключового слова і відступом перед значеннями.

Також зустрічаються документи, де наявні додаткові частини, такі як відмітки чи наявність пункту, який властивий іншому типу документу. Такі випадки теж необхідно взяти до уваги і в алгоритмі додати фрагмент коду, який буде опрацьовувати їх, щоб не було втрати інформації.

номер провадження справи 9/145/18

ГОСПОДАРСЬКИЙ СУД ЗАПОРІЗЬКОЇ ОБЛАСТІ

РІШЕННЯ

ІМЕНЕМ УКРАЇНИ

02.01.2019 **Справа № 908/2391/18**

м.Запоріжжя

За позовом: Товариства з додатковою відповідальністю "Експрес Страхування", код ЄДРПОУ 36086124 (01004, м. Київ, вул. Велика Васильківська, 15/2)

до відповідача: Приватного акціонерного товариства "Українська акціонерна страхова компанія "АСКА", код ЄДРПОУ 13490997 (69005, м. Запоріжжя, вул. Перемоги, 97-А)

про стягнення відшкодування у розмірі 100 000,00 грн.

Суддя Боєва О.С.

Без виклику сторін

СУТЬ СПОРУ:

До господарського суду Запорізької області 13.11.2018 надійшла позовна заява Товариства з додатковою відповідальністю

Рисунок 3. 1 Приклад рішення 1

МІКОЛАЇВСЬКИЙ ОКРУЖНИЙ АДМІНІСТРАТИВНИЙ СУД

Р І Ш Е Н Н Я

І М Е Н Е М У К Р А Ї Н И

м. Миколаїв.

02 січня 2019 р.справа № 1440/1699/18

Миколаївський окружний адміністративний суд, у складі судді Мороза А.О., в письмовому провадженні розглянув адміністративну справу

за позовомтовариства з обмеженою відповідальністю "Миколаївський глиноземний завод", пр. Жовтневий, 471, м. Миколаїв, 54051

до відповідачаМиколаївської митниці ДФС, вул. Московська, 57-а, м. Миколаїв, 54017

третя особа, яка не заявляє самостійних вимог щодо предмета спору на стороні відповідачаГоловне управління Державної казначейської служби України у Миколаївській області, пр. Центральний, 141-В, м. Миколаїв, 54055

провизнання бездіяльності протиправною та зобов'язання вчинити дії

В С Т А Н О В И В:

Товариство з обмеженою відповідальністю "Миколаївський глиноземний завод" (надалі - позивач або ТОВ "МГЗ") звернулось із

Рисунок 3. 2 Приклад рішення 2

3.3.3 Визначення структури даних, що повинна утворитися в результаті парсингу

На етапі аналізу вхідних даних були виділені певні токени, які присутні в усіх типах документів. Серед них є номер справи, прізвище та ім’я судді, номер документа, дата ухвалення, тип справи. Тип справи визначає решту структури документа і на наступному етапі впливає на алгоритм опрацювання справи.

Серед доступних типів судових документів виділені рішення, накази, ухвали, вироки та постанови.

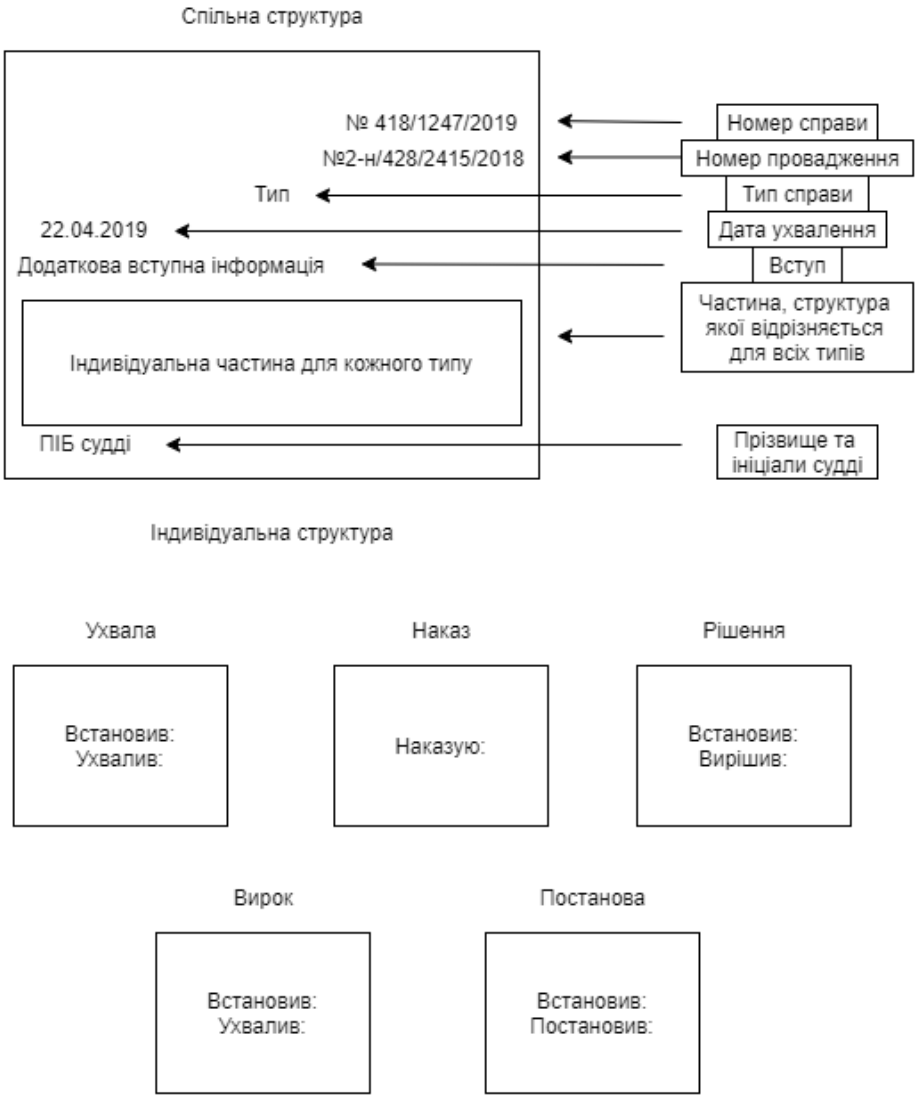


Рисунок 3. 3 Схема результуючих структур

Схема 3.3 ілюструє структури, які повинні утворитися в результаті парсингу. На ній зображено елементи, які є спільними токенами для всіх типів вхідних даних, відображено частину документу, де зустрічаються розбіжності між структурними елементами, і нижче наведено структуру індивідуальної частини кожного розглянутого типу документа.

Отже, в загальному випадку кожен документ починається номером справи, в наступному рядку вказано номер провадження, рядком нижче вказано тип судового документа. Під ним починається вступна частина, з якої окремо

виділяється дата ухвалення. Нижче починається частина, яка є індивідуальною для кожного типу. Кожен документ закінчується вказаним суддею, який головував при ухваленні даного рішення.

В індивідуальній частині ухвали і вироку спільні токени – частини, що слідують після ключових слів «Встановив:» та «Ухвалив:» до початку наступного токена. В наказах індивідуальна частина містить один токен, що слідує після ключового слова «Наказую:». Для рішень ця частина складається з токенів після «Встановив:» та «Вирішив:», а для постанов – після «Встановив:» та «Постановив:».

Структура даних, що повинна утворитися в результаті парсингу, є близькою до структури вхідних даних, з ключовою відмінність, що в результаті окремо виділені додаткові токени.

3.4 Створення парсера для судових рішень

Після того, як було здійснено аналіз вхідного датасету, виявлено типи документів, всі особливості та граничні випадки, потрібно визначити правила для кожного токена.

На першому етапі було виявлено основні правила для кожного токена. Проте при детальному вивченні особливостей датасету шляхом порівняння структури документів одного типу, було виявлено, що є багато варіацій позиціонування елементів один відносно іншого та у файлі. Через це парсер містить багато edge cases, щоб було покриття всіх виявлених відмінностей.

Велику складність становило завдання отримати з тексту окремо номер рішення, номер справи та дату. Адже в документі присутні багато числових значень. Всі вони досить схожі, що вимагає чіткого формулювання правила розпізнавання, але в той же час, існує кілька варіантів написання цих токенів, що унеможливлює створення універсального парсера для них.

Основну роботу парсингу виконує parse_doc(filename), що приймає на вхід шлях до файлу, який потрібно проаналізувати, зчитує його, розбиває на токени за розміткою документа і запускає алгоритм обробки. Алгоритм побудований так, що більша частина є універсальною для всіх типів документів. Тільки на етапі поділу на логічні частини судового документу, алгоритм розділяється на кілька віток. Звичайно, була можливість побудувати алгоритм таким чином, щоб не звертати увагу на вид файлу впродовж всього аналізу до моменту збереження структури в результат. Проте ефективність роботи це не покращило б, так як забираючи перевірку тип провадження, довелося б здійснювати підстановку правил, що мають сенс лише в конкретному випадку. Більше того, таких підхід значно б погіршив читабельність коду.

Також в ході аналізу документів було виявлено, що послідовність перевірки правил є важливою. Наприклад, в загальному випадку номер справи складається з трьох чисел записаних через «/», а цій послідовності передує «Справа №». А номер провадження – це послідовність з чотирьох чисел, записаних через «/». В деяких випадках їй передує «номер провадження», але послідовність з чотирьох числових комбінацій є унікальною для документа, тому цього достатньо для написання правила. Проте не всі судді дотримуються стандарту і зустрічаються документи з номером, що складається з трьох чисел. Ключові слова перед номерами теж деколи відсутні. В таких випадках звичайного регулярного виразу для пошуку номерів не достатньо. Тому правила відсортовані в порядку від найбільш точного до найбільш загального.

```
for i in range(0, len(strips)):
    if not caseFound:
        if re.search("№", unicodedata.normalize('NFD', strips[i])):
            if re.search("/", strips[i]):
                if i + 1 < len(strips) and re.findall("\d+\/\d+\/\d+", strips[i]):
                    caseNo = re.findall("\d+\/\d+\/\d+", strips[i])[0]
                    caseFound = True
            elif i+1 < len(strips) and re.findall("\d+\/\d+\/\d+", strips[i + 1]):
                caseNo = re.findall("\d+\/\d+\/\d+", strips[i + 1])[0]
                caseFound = True
```

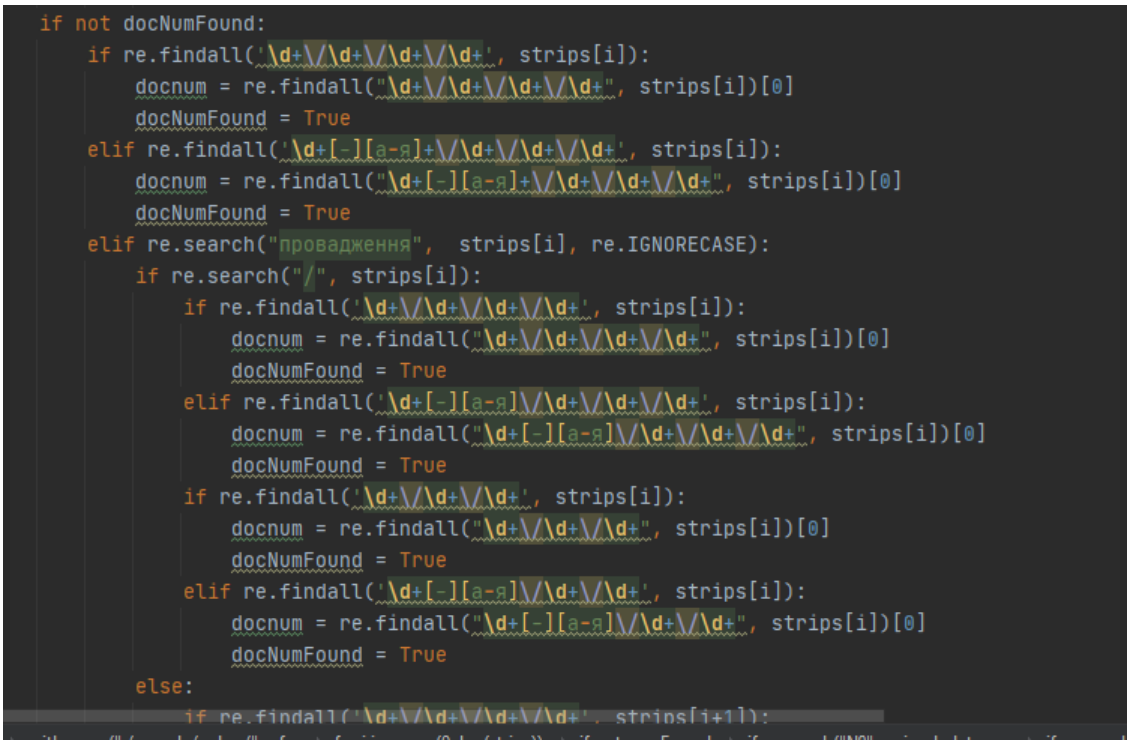
Рисунок 3. 4 Початок алгоритму аналізу судового документа

Правила застосовуються до результатів лексичного аналізу, який розбив суцільний текст на частини відповідно до розмітки документа.

Як продемонстровано на рисунку 3.4, правила застосовуються по черзі до кожного токена, отриманого на попередньому етапі, починаючи з пошуку номера справи. Правила, в результатах застосування яких утворюється однакова продукція, застосовуються тільки до того моменту, поки воно не було застосоване до правильного токена. Таким чином уникається заміна чітко виділеного токена на той, що був отриманий в результаті підстановки більш загального правила.

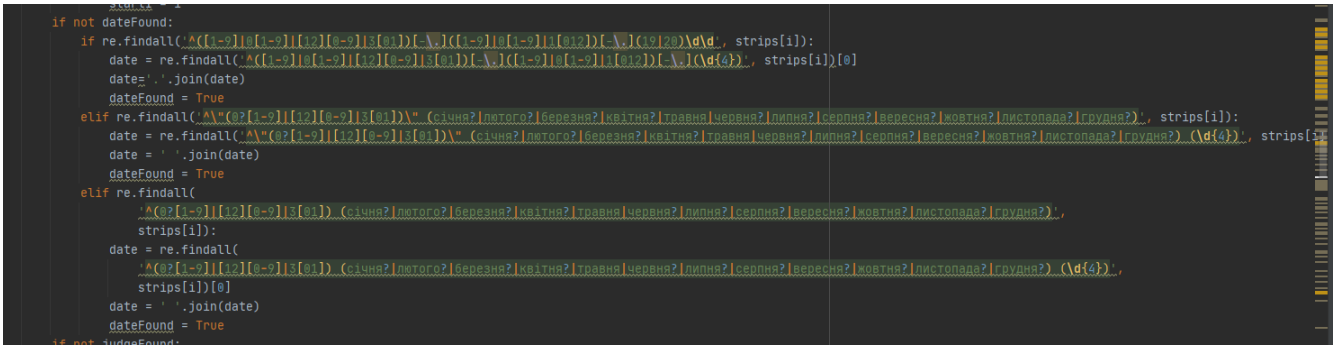
Наступна перевірка на наявність номера документа в токені (рисунок 3.5). В ході аналізу вхідних даних було виявлено, що формат запису теж відрізняється, а також частою є ситуація, коли ключове слово і сам номер в результаті лексичного аналізу опиняються в різних токенах. Набір правил для визначення номера документа передбачає кожну з описаних ситуацій.

Визначення типу документу є нескладним, так як для них назва написана великими літерами разом або через пробіл.



```
if not docNumFound:
    if re.findall('\d+\/\d+\/\d+\/\d+', strips[i]):
        docnum = re.findall("\d+\/\d+\/\d+\/\d+", strips[i])[0]
        docNumFound = True
    elif re.findall('\d+[-][а-я]+\/\d+\/\d+\/\d+', strips[i]):
        docnum = re.findall("\d+[-][а-я]+\/\d+\/\d+\/\d+", strips[i])[0]
        docNumFound = True
    elif re.search("провадження",  strips[i], re.IGNORECASE):
        if re.search("/", strips[i]):
            if re.findall('\d+\/\d+\/\d+\/\d+', strips[i]):
                docnum = re.findall("\d+\/\d+\/\d+\/\d+", strips[i])[0]
                docNumFound = True
            elif re.findall('\d+[-][а-я]\/\d+\/\d+\/\d+', strips[i]):
                docnum = re.findall("\d+[-][а-я]\/\d+\/\d+\/\d+", strips[i])[0]
                docNumFound = True
            if re.findall('\d+\/\d+\/\d+', strips[i]):
                docnum = re.findall("\d+\/\d+\/\d+", strips[i])[0]
                docNumFound = True
            elif re.findall('\d+[-][а-я]\/\d+\/\d+', strips[i]):
                docnum = re.findall("\d+[-][а-я]\/\d+\/\d+", strips[i])[0]
                docNumFound = True
        else:
            if re.findall('\d+\/\d+\/\d+\/\d+', strips[i+1]):
```

Рисунок 3. 5 правила для вибору номера провадження



```
if not dateFound:
    if re.findall('^([1-9]|0[1-9]|[12][0-9]|3[01])[-\.]([1-9]|0[1-9]|1[012])[-\.](19|20)\d\d', strips[i]):
        date = re.findall('^([1-9]|0[1-9]|[12][0-9]|3[01])[-\.]([1-9]|0[1-9]|1[012])[-\.](\d{4})', strips[i])[0]
        date='.'.join(date)
        dateFound = True
    elif re.findall('^\"(0?[1-9]|[12][0-9]|3[01])\" (січня?|лютого?|березня?|квітня?|травня|червня?|липня?|серпня?|вересня?|жовтня?|листопада?|грудня?)', strips[i]):
        date = re.findall('^\"(0?[1-9]|[12][0-9]|3[01])\" (січня?|лютого?|березня?|квітня?|травня|червня?|липня?|серпня?|вересня?|жовтня?|листопада?|грудня?) (\d{4})', strips[i])
        date = ' '.join(date)
        dateFound = True
    elif re.findall(
            '^(0?[1-9]|[12][0-9]|3[01]) (січня?|лютого?|березня?|квітня?|травня|червня?|липня?|серпня?|вересня?|жовтня?|листопада?|грудня?)',
            strips[i]):
        date = re.findall(
            '^(0?[1-9]|[12][0-9]|3[01]) (січня?|лютого?|березня?|квітня?|травня|червня?|липня?|серпня?|вересня?|жовтня?|листопада?|грудня?) (\d{4})',
            strips[i])[0]
        date = ' '.join(date)
        dateFound = True
```

Рисунок 3. 6 Правила для розпізнавання дати ухвалення рішення

Дати ухвалення рішень можуть бути присутні в трьох форматах: «дд.мм.рррр», « «дд» місяця рррр», «дд місяця рррр». Набір правил для цього випадку зображений на рисунку 3.6.

Окремо визначений набір правил для отримання токену – імені судді.

Потім здійснюється розгалуження алгоритму для поділу документу на частини відповідно до типу. Після цього залежно від типу документу формується

результуюча структура і повертається tuple зі списками структур документів кожного типу.

Нижче на рисунках 3.7-3.9 наведено результати парсингу документів.

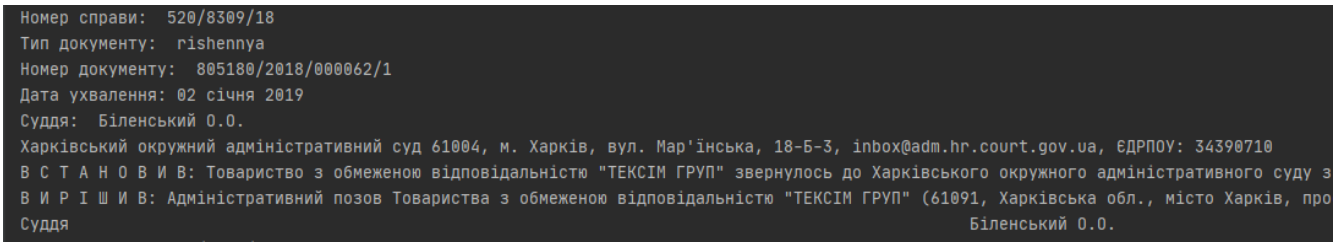


Рисунок 3. 7 Результат парсингу рішення

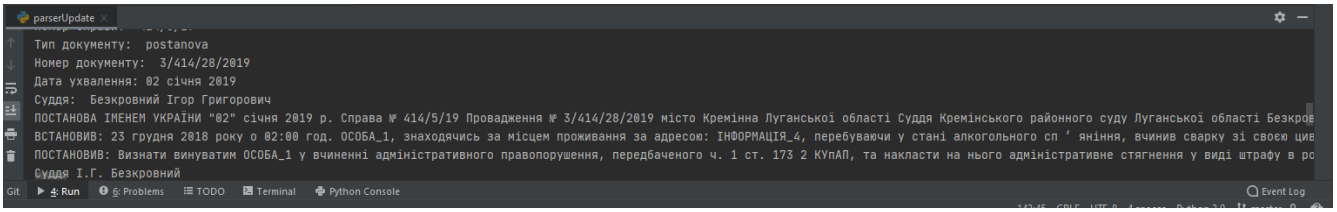


Рисунок 3. 8 Результат парсингу постанови

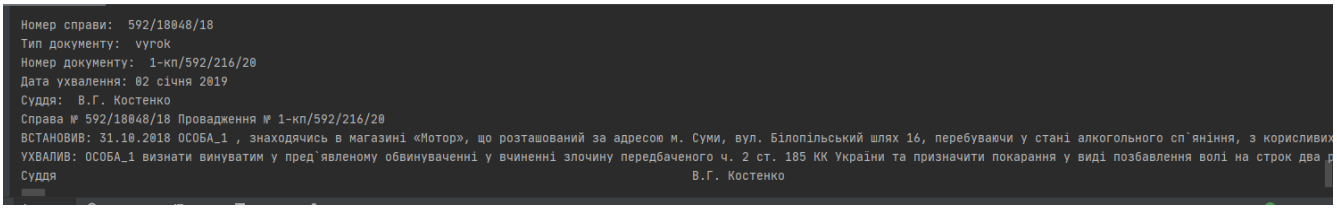


Рисунок 3. 9 Результат парсингу вироку

Під час тестування були виявлені випадки, коли не всі токени були знайдені, або розпізналася інша комбінація замість токена. Серед проблемних ситуацій виявлення токена судді. Так як нема конкретно визначеної позиції його ПІБ, неможливо отримати 100% успішний результат. Коли є прізвище судді з ініціалами, такий токен розпізнати можна з високою ймовірністю. Проте коли ПІБ вказано повністю, то точність значно спадає. Адже в багатьох документах запис судді йде у форматі типу «Суддя Дніпровського Центрального Адміністративного Суду», і як ПІБ розпізнається будь-яка з послідовностей з 3 слів, що починаються великою буквою.

Помилки, які були типовими в роботі парсера, були усунені. Проте деякі, які зустрічаються в поодиноких документах, залишилися, так як було вирішено недоцільно створювати додатковий ряд правил задля одного документу.

В загальному результати парсингу можна вважати успішними. Отримані структури були передані в ElasticSearch і дають можливість фільтрувати пошук і створювати запити, отримуючи очікувані результати.

3.5 Використання результатів парсингу в проекті

Процес парсингу в межах даного проекту є необхідною умовою попередньої обробки тексту. Переваги зберігання документів як наборів полів має цілий ряд переваг, які були продемонстровані в даному проекті. Його застосування має вплив як на ефективність роботи системи з точки зору обмеженої об’єму опрацьовуваних даних при кожному запиті, так і на можливість створення універсального вигляду документів, ігноруючи розбіжності, які зустрічаються між документами одного типу, завдяки використанню окремих частин контенту в єдиному шаблоні.

Парсинг запускається в момент збереження датасету на сервері. В основі зберігання даних є Apache Lucene, так як саме її використовує ElasticSearch. Документи, завантажені в Lucene зберігаються як обернений список. Отже, індекс – це послідовність документів. Документ – це послідовність полів. Поле – це іменована послідовність термів. Термом виступає рядок. Однаковий рядок в різних полях вважається різним термом. Терми представляються як пари рядків, перший рядок називає поле, інший текст цього поля.

Для кожного типу документів існує окремий індекс. Це пришвидшує процес пошуку за умови використання фільтру по типу документу. Те, в який індекс повинен бути доданий документ, визначається саме в процесі парсингу, так як цей тип документу є одним з шуканих токенів. Кожен файл в залежності від свого типу розбивається на частини. Для всіх документів обов’язковою частиною є тип

документу, номер справи, номер документу, дата затвердження документу та ім’я судді. Далі, в залежності від типу, виділяються основні частини документу. Наприклад, для рішення, ці частини – вступ, «встановив» та «вирішив», що в термінах правової сфери називаються вступна, описова, мотивувальна та резолютивна частина. Після розбиття парсером всіх документів, здійснюється завантаження утворених пар значень кожного документа у відповідний індекс.

Перша можливість, яку надає таке збереження документу – це пошук з пріоритетами. Така функція при роботі з великими об’ємами даних є критичною. ElasticSearch вже сам надає результати в ранжованому порядку, завдяки сталій кількості результатів, які повертаються, відповіді є максимально релевантними до запиту. Це допомагає обмежити користувача в потребі перегляду сотень документів в пошуку потрібного. Також, ким точніше сформований запит, тим більша ймовірність отримати необхідний документ вже серед перших результатів запиту. Проте частота і точність запиту – не єдине, що впливає на релевантність документу.

В судових документах важливу роль грає саме частина, в якій знайдене це слово. Часто є ситуації, коли користувачі потребують ознайомитися з документами, які були сформовані за рішенням певного судді. Суддей існує дуже багато, щоб прізвище судді виносити в фільтр. Проте в документах також вказані особисті дані учасників судового процесу. А враховуючи, що таких документів в системі тисячі або й мільйони – залежно від повноти датасету, то кількість документів за одним прізвищем буде перевищувати кількість відображуваних результатів. Для довідки, у Єдиному Державному Реєстрі Судових Рішень на даний момент більше 94 мільйонів документів. Тому полю з прізвищем судді пріоритет присвоєний вищий, ніж вступній чи описовій частині. Також резолютивна частина – фрагмент документу, де описані деталі рішення, прийнятого суддівською колегією, для більшості користувачів несе вирішальну роль, тому цій частині теж був наданий більший пріоритет. Пріоритети полів

визначаються для індексу. І при здійсненні пошуку ці пріоритети відіграють роль при обрахунку коефіцієнту релевантності. Така можливість визначення пріоритетів була б неможливою без розділу текстового документу на токени.

Другою перевагою є можливість легкої реалізації фільтрів. З документів виділяється найважливіші ключові слова/фрази, вони записуються окремими полями, тоді в запиті можна вказати, назву фільтра, що відповідає назву поля в індексі, і значення, що обрав користувач. Звичайно, можна було б щоразу здійснювати пошук даного запиту по всьому тексту, але це є дуже ресурсозатратно та вимагає постійного повторювання виконуваної дії. Значно ефективніше виділити це поле раз, при обробці всього датасету, а тоді здійснювати пошук по записах в індексі, де дане поле має вказане значення. Особливо відчутною така перевага була б при фільтрації за значеннями, які можуть містити різні формати. Наприклад при пошуку по даті чи номеру телефону. При записі в індекс можна дату чи номер конвертувати в один універсальний формат і поле в фільтрі зробити форматованим, яке б відповідало тому, в якому воно зберігається в індексі. Тоді не потрібно щоразу здійснювати пошук всіх можливих форматах в кожному документі, достатньо знайти документи з повним співпадінням відповідних полів.

Третьою перевагою є можливість відображувати документи в сталому вигляді. На рисунку 3.10 зображена розмітка рішення № 2-о/219/9/2018, на якому демонструється відсутність чіткої норми щодо відступів між частинами документу. Ці відступи різняться між документами. Це негативно впливає на досвід користувача. Парсинг документів та передача їх у вигляді json-об’єктів з окремими полями-складовими документа на клієнтську частину застосунку дозволяє відображати документи зберігаючи однакову відстань між блоками та стиль відображення. Така можливість була б неможлива у випадку надсилання документу як єдиного рядка.

```
<p style="..."><b><span style="...">       Суддя
</p>
<p> </p>
<p> </p>
<p> </p>
<p> </p>
<p> </p>
<p> </p>
<p> </p>
<p style="..."><b><span
        style="...">                                                    </span></b><span
        style="..."> Провадження № 2-о/219/9/2018</span></p>
<p> </p>
<p style="..."><b><span style="...">Р І Ш Е Н Н Я</span></b>
</p>
<p><b><span style="...">І М Е Н Е М У К Р А Ї Н И</span></b></p>
<p><span style="..."> </span></p>
<p style="..."><span style="...">        </span></p>
<p> </p>
<p> </p>
<p> </p>
<p style="..."><b><span
        style="...">       Суддя Л.І.Хомченко</span></b></p>
<p> </p>
<p> </p></body>
```

Рисунок 3. 10 Приклад розмітки документу

І останньою явною перевагою, яку система парсингу принесла проекту, це можливість групування документів за номером справи. Тепер, коли номер справи не знаходиться серед десятків кілобайтів інших даних, результат пошуку можна видати як мапу масивів, де ключем буде значення поля документа, що зберігає номер справи, а значенням масив з рештою полів відповідних документів.

Висновок

У даній роботі були досліджені етапи проектування та реалізації парсингу, інструменти, які використовуються в процесі парсингу.

Досліджено особливості датасетів, які можуть впливати на алгоритм парсера та вимагати опрацювання edge-cases, якщо відсутній універсальний спосіб охоплення всіх розбіжностей. Серед таких особливостей є розширення документів, наявність/відсутність розмітки та ідентифікаторів в ній, відсутність єдиного формату типів (дата, номер телефону тощо), необов'язкові елементи структури.

Був проведений аналіз практичного застосування парсингу. У епоху цифрової трансформації, коли всі сфери як державного, так і комерційного призначення переходять до автоматизації, прозорості та глобалізації ресурсів та звітності, необхідно перемістити мільйони документів у сховища(дата центри), та дати можливість певним групам осіб працювати з даними, що містяться у них.

При конвертації документів, що були створені «людьми для людей», необхідно провести певну обробку, яка дозволить алгоритмам коректно та ефективно надалі працювати з ними. Для цього з одного текстового документа створюється структура, виділяються токени. Для того, щоб результати використовували фізичні ресурси економно та надавали можливість масштабування, необхідне ретельне проектування структур, в яких дані будуть зберігатися, парсера, в якому враховані всі особливості та недоліки датасету, щоб покрити всі граничні випадки.

З метою демострації побудови парсера було проведено створення системи парсингу судових документів. В даному проекті парсинг використовується для попередньої обробки документів перед завантаженням їх у сховище.

Для створення парсера було проведено аналіз датасету. Виділено типи документів, вивчено їхню розмітку та структуру. Проведено порівняння документів одного виду для знаходження всіх структурних та форматних відмінностей. Потім було визначено структури, в які повинен кожен з типів перетворитися. Після цього було реалізовано систему парсингу судових документів.

В цілому, під час дослідження етапів процесу парсингу текстових документів було зроблено висновок, що даний процес сприяє уніфікації датасету, перетворенню його на структури, які потребують менше ресурсів для збереження, аналізу та повернення результатів. Також визначено важливість ретельного проектування парсера, яка впливає на своєрідний «hit rate» - відсоток успішно опрацьованих документів, що потім має вплив на достовірність результатів роботи системи та ефективність цього процесу.

Список використаних джерел

1. Oracle: Parsing Concept Guide [Електронний ресурс] / Oracle. – 2006. – Режим доступу до ресурсу:.https://www.oracle.com/webfolder/technetwork/data-quality/edqhelp/Content/advanced_features/parsing.htm

2. Schwarz K. Syntax Analysis [Електронний ресурс] / Keith Schwarz – Режим доступу до ресурсу: https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/lectures/02/Slides02.pdf

3. Ontotext. Text Analysis Fundamentals [Електронний ресурс] / Ontotext – Режим доступу до ресурсу: https://www.ontotext.com/knowledgehub/fundamentals/text-analysis/

4. The Apache Software Foundation. Apache Lucene - Index File Formats [Електронний ресурс] / The Apache Software Foundation. – 6. – Режим доступу до ресурсу: https://lucene.apache.org/core/3_5_0/fileformats.html

5. Elasticsearch Clients [Електронний ресурс] – Режим доступу до ресурсу: https://www.elastic.co/guide/en/elasticsearch/client/index.html

6. Єдиний державний реєстр судових рішень [Електронний ресурс] – Режим доступу до ресурсу: https://reyestr.court.gov.ua/

7. Regular expression operations [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/library/re.html

8. Compiler Design - Syntax Analysis [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/compiler_design/compiler_design_syntax_analysis.htm

9. Tomassetti G. Parsing In Python: Tools And Libraries [Електронний ресурс] / Gabriele Tomassetti – Режим доступу до ресурсу: https://tomassetti.me/parsing-in-python/