ANALIZADOR Y CLASIFICADOR DE MENSAJES TECNOLOGIAS CHAPINAS S.A.

202006353 – Juan Josue Zuleta Beb

Resumen

Se realizo una minuciosa preparación en modelado de programación orientada a objetos para solucionar el problema planteado por la empresa Tecnologías Chapinas S.A. con lo cual se logró el manejo de archivos con extensión XML en el lenguaje de programación Python con el uso de la librería MiniDom. Implementa programación orientada a objetos para el manejo de la memoria estática y esta utiliza el manejo de la memoria dinámica. Para el desarrollo del proyecto se utilizó estructuras tradicionales extracción de datos y almacenamiento en clases para generar objetos con atributos propios según se presenten en el archivo de entrada XML se generó peticiones por medio de la herramienta Flask, para poder acceder a los métodos mediante el cliente (Frontend), para que el usuario tenga una mejor idea de lo que son los datos extraídos del archivo XML. El dominio total del proyecto se basó en el backend el cual es el encargado de manipular la información enviada y solicitada por el frontend mediante herramientas propias de análisis léxico para la clasificación de las palabras y su significado. En conclusión, se puede decir que este proyecto es una introducción a las APIs y a la implementación de cliente-servidor.

Palabras clave

- XML
- MiniDom
- Flask
- API
- Frontend
- Backend

Abstract

A thorough preparation in object-oriented programming modeling was carried out to solve the problem posed by the company Tecnologías Chapinas S.A. with which the handling of files with XML extension in the Python programming language was modified with the use of the MiniDom library. It implements object-oriented programming for static memory management and it uses dynamic memory management. For the development of the project, traditional data extraction and storage structures were carried out in classes to generate objects with their own attributes as presented in the XML input file, requests were reaffirmed through the Flask tool, in order to access the methods through the client. (Frontend), so that the user has a better idea of what the data extracted from the XML file is. The total domain of the project was based on the backend which is in charge of manipulating the information sent and requested by the frontend through its own lexical analysis tools for the classification of words and their meaning. In

conclusion, it can be said that this project is an introduction to APIs and client-server implementation.

Keywords

- XML
- MiniDom
- Node
- Linked List
- ADTs (Abstract Data Type)
- Graphviz

Introducción

A continuación, se presentará la solución del problema del proyecto, para la cual se utilizó el lenguaje de programación Python y el paradigma de programación orientado objetos, y para el manejo de memoria se utilizó librerías nativas de Python.
Estas herramientas de programación permitieron manejar la memoria del archivo y poder manipularla en diferentes clases, sabiendo que ya que se usó el paradigma orientado a objetos esto facilitó el orden y la comunicación de datos entre clases y así evitar la perdida de datos enviados y recibidos entre las mismas. Para las distintas clases se utilizaron las importaciones nativas de Python las cuales nos permiten manejar las clases dentro del mismo archivo evitando así importaciones circulares y también falta de importaciones en el manejo de las distintas clases.

Desarrollo del tema

El problema de diseño de mensajes, tipos de mensajes y su clasificación, consiste generalmente en un problema de determinar el tipo de patrón obtenido y de esto generar el análisis de palabras que son extraídas de del mensaje, estas son el lugar en el que se envio el mensaje, la fecha en la cual se envio, también incluye la hora y el usuario que realiza el envio del mensaje, este también contiene la red social en la cual ha sido enviado el mensaje y el cuerpo del mensaje el cual es el objetivo principal a evaluar ya que el resto de datos serán atributos de un objeto llamado Message, el cual contendrá los datos antes mencionados y la clasificacion del mensaje.Algunas de las situaciones comunes que hemos observado cuando se resuelven instancias muy grandes de un problema de este tipo son: Fuerte requerimiento de tiempo y fuerte demanda de recursos de memoria. Iniciando por extraer los datos del archivo XML y almacenándolos en los objetos de las clases, esta tarea la realizamos por medio de la librería que implementa Python llamada MiniDom, la cual nos permite buscar fuentes de datos por medio de nombres de etiquetas, de esta manera separamos los nombres de las empresas, nombres de los servicios y nombres de los alias de los servicios, así también se extrajeron los diccionarios de las palabras positivas y negativas las cuales servirán de referencia al momento de analizar los mensajes. Los datos fueron clasificados por sus etiquetas y mediante Minidom se extrajo el dato de cada etiqueta buscando la etiqueta por su nombre, con la función getElementsByTagName, esto permitió facilitar la obtención de elementos del mismo tipo dentro del XML así mismo se extrajeron los datos dentro de la etiqueta con la función firtsChild.data, y se extrajo los datos dentro del atributo de la etiqueta con la función atributes.value , Una vez extraídos los datos y almacenados en las siguientes clases:

- Corp (Objeto empresa)
- Corp_List (Lista de objetos empresa)
- Service (Objeto servicio)
- Service_List (Lista de objetos de servicio)
- Aka (Objeto Alias)
- Aka_List (Lista de Objetos de alias)
- State (Lista de sentimientos)
- Posivito (Objeto de palabra positiva)
- Negativo (Objeto de palabra negativa)

```
xml = request.get_data().decode('utf-8')
mydoc = minidom.parseString(xml)
diccionarios = mydoc.getElementsByTagName('diccionario')
mensajes = mydoc.getElementsByTagName('lista_mensajes')

for elemento in diccionarios:
    positivos = elemento.getElementsByTagName('sentimientos_positivos')
    negativos = elemento.getElementsByTagName('sentimientos_negativos')
    empresas = elemento.getElementsByTagName('empresas_analizar')

    for items in positivos:
        palabra = items.getElementsByTagName('palabra')
        for pala in palabra:
            txt = pala.firstChild.data
            txt = re.sub(r"([^\n\u0300-\u036f]|n(?!\u0303(?![\u0300-\u036f])
            manage.add_positive(txt)

    for itemses in negativos:
        palabra2 = itemses.getElementsByTagName('palabra')
        for pala2 in palabra2:
            txt1 = pala2.firstChild.data
            txt1 = re.sub(r"([^\n\u0300-\u036f]|n(?!\u0303(?![\u0300-\u036f]
            manage.add_negative(txt1)
    for subElemento in empresas:
        empresa = subElemento.getElementsByTagName('empresa')
        for attribs  in empresa:
            nombres = attribs.getElementsByTagName('nombre')
            servicios = attribs.getElementsByTagName('servicio')
            for names in nombres:
                nombre = names.firstChild.data
                corpse.add_to_end(nombre)
            for ser in servicios:
                servicio = ser.attributes['nombre'].value
                aliases = ser.getElementsByTagName('alias')
                corps: Corp = corpse.get_by_name(nombre)
                corps.services.add_to_end(servicio)
                for aka in aliases:
                    aliases: Service = corps.services.get_by_name(servicio)
                    alias = aka.firstChild.data
                    aliases.aka.add_to_end(alias)

for elemento in mensajes:
    msm = elemento.getElementsByTagName('mensaje')
    for item in msm:
        txt = item.firstChild.data
        txt = re.sub(r"([^\n\u0300-\u036f]|n(?!\u0303(?![\u0300-\u036f])))[\u0300-\u036f]
        texto.append(txt)
```

Figura 1. Extracción de datos.

Fuente: Elaboración propia.

La manipulación de estas se llevo a cabo por medio de un método llamado menú principal, por medio del cual accedimos a todas las listas y manejamos ampliamente y a voluntad, nuestros datos almacenados.

Para iniciar con la ejecución se debe ingresar los datos para lo cual se debe cargar un archivo de extensión XML que cumpla con la estructura inicialmente planteada siguiendo las mismas distribuciones de nombre y raíces, por consiguiente podremos observar por separado cada uno de los mensajes con sus atributos separados y sus textos listos para ser impresos, por lo cual realizamos un algoritmo tradicional para analizar texto, un automata finito determinista el cual nos ayudara a obtener la secuencia de palabras dentro de cada mensaje, y esto nos ayudara a clasificar los atributos de los mensajes según el significado de cada palabra, esto debido a que tendremos clases de palabras para cada tipo de entrada, por ejemplo, las palabras lugar y fecha, son palabras “reservadas” del lenguaje manejado por lo cual no se tomaran como parte del mensaje, sino que estas nos servirán como referencia para encontrar la palabra que les sigue y así encontrar el texto que indica el lugar de donde fue realizado el post del mensaje y también la fecha.

Seguido de esto llamaremos al método, Scanner que realizara el análisis de la lista de tokens retornada por el automata, esta lista tendrá la clasificación de las palabras y facilitara al scanner encontrar las palabras que necesita para crear los objetos de tipo Message, los cuales tienen como atributos cada uno de los atributos del mensaje de entrada, estos serán :

```
class Message:
    def __init__(self) -> None:
        self.lugar: str = ''
        self.fecha: str = ''
        self.hora: str = ''
        self.usuario: str = ''
        self.red_social: str = ''
        self.texto = ''
        self.estado = ''
```

Figura 2. Objeto mensaje.

Fuente: Elaboración propia.

s

Figura 3. conversión de XML a matriz de colores.

Fuente: Elaboración propia.

Una vez que tenemos cada mensaje como un objeto de nuestra lista con cada uno de sus atributos definidos, podemos pasar a manipularlos en cualquier clase y en cualquier método, de esta premisa se sugiere que lo mejor sea manipularlos en

la misma lista en donde fueron creados, por lo cual en la lista Corps_List se crea el método de buscar por nombre, con lo cual podremos buscar un objeto dentro de la lista y así acceder al nombre del objeto y a sus atributos los cuales serán estos:

```
class Corps_List():
    def __init__(self) -> None:
        self.corps = []

    def add_to_end(self,name):
        new = Corp(name)
        self.corps.append(new)

    def get_by_name(self,name):
        for i in range(len(self.corps)):
            if self.corps[i].name == name:
                return self.corps[i]
```

Figura 3. Lista de Objetos Corp (Empresas).

Fuente: Elaboración propia.

De esta forma podemos manejar en general nuestros Objetos de tipo Corp (empresa). La implementación de la programación orientada objetos por medio de la cual recorrimos cada objeto buscando sus atributos en el caso principal comparamos nombres de esta forma se facilitó la búsqueda de palabras y la obtención de coincidencias con los nombres de las empresas y sus servicios y sus alias con los textos en los mensajes.

Algo importante a mencionar es que, al manipular datos por medio de la programación orientada a objetos es fácil perder de vista donde estamos situados al momento de realizar un cambio a un atributo o añadir un nuevo atributo a un objeto, por lo cual es necesario nombrar de forma adecuada a las clases y a los atributos para que al momento de llamar objetos dentro de objetos y listas dentro de listas esto no sea algo perjudicial para el programa y que la manipulación de los datos sea limpia sin pérdida de datos.

```
def scanner(tokens: List[Token],positives,negatives,empresas: Corps_List):
    global contador_mensajes
    list_parameters: List[Message] = []
    tmp_messages: list = []
    messages: list = []
    tmp_messages.append(messages)
    for i in range(len(tokens)):
        if tokens[i].token == '':
            new_msms: list = []
            tmp_messages.append(new_msms)
            continue
        tmp_messages[-1].append(tokens[i])
    try:
        for messages in tmp_messages:
            msm = Message()
            for i in range(len(tokens)):
                #extraer datos
                if tokens[i].token == 'reservada' and tokens[i].lexeme.lower() == 'lugar' \
                    and tokens[i+1].lexeme.lower() == 'y'and tokens[i+2].token == 'reservada' \
                    and tokens[i+2].lexeme.lower() == 'fecha' and tokens[i+3].lexeme == ':' \
                    and tokens[i+4].token == 'id'and tokens[i+5].lexeme == ',' \
                    and tokens[i+6].token == 'fecha' and tokens[i+7].token == 'hora' \
                    and tokens[i+8].token == 'reservada' and tokens[i+8].lexeme.lower() == 'usuario' \
                    and tokens[i+9].lexeme == ':'  and tokens[i+10].token == 'user' \
                    and tokens[i+11].token == 'reservada' and tokens[i+11].lexeme.lower() == 'red' \
                    and tokens[i+12].token == 'reservada' and tokens[i+12].lexeme.lower() == 'social' \
                    and tokens[i+13].lexeme == ':' and tokens[i+14].token == 'id':
```

Figura 4. Método Scanner (Extraer Datos)

Fuente: Elaboración propia.

Conclusiones

Se puede concluir que la programación orientada a objetos es fundamental para el manejo y la implementación de datos en la memoria estática y dinámica de nuestra aplicación.

También se puede afirmar que las APIs son el principal movimiento de desarrollo actual y que es necesario conocer estas tecnologías para el uso y la implementación de los conocimientos adquiridos en el curso.

Se concluye que al utilizar librerías nativas en los lenguajes de programación como en este caso Python es de gran ayuda ya que simplifican la forma en que programamos y manejamos la memoria dinámica, sin embargo, cuando no se utilizan librerías nativas es más fácil tener el control total de cómo se está manejando la memoria y los datos almacenados.

Referencias bibliográficas

- https://ebooksonline.es/leer-un-archivo-xml-de-ejemplo-minidom-elementtree/
- https://flask.palletsprojects.com/en/2.1.x/
- https://docs.insomnia.rest/
- https://flask.palletsprojects.com/en/2.1.x/api/

Anexos

Diagrama de clases: