Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 2do. Semestre 2023.

PROYECTO 3

202200075 – Dominic Juan Pablo Ruano Pérez

Resumen

Utilizando Tipos de Datos Abstractos (TDA), Django para crear una interfaz gráfica de usuario amigable en formato de página web, se desarrollado un sistema capaz de procesar archivos con extensión ".XML" y almacenar los datos en bases de datos los cuales se leen al iniciar el programa para guardar los mismos. Se implementaron dos tipos de servidores para esta práctica de implemento con flask un API el cual se encarga de el manejo de todo lo relacionado con la lectura de archivos, generación de las lenturas, guardado y leído de la base de datos.

Con el uso de Flask se logro crear un servidor capaz de leer n archivos de mensajes y n archivos de configuración y asegurando el guardado de todos esos en archivos .XML los cuales se leerán de nuevo la iniciar el programa, siendo así que se conservan los datos del programa a pesar de cerrarlo, apagarlo o incluso reiniciando el equipo

Mediante Django se crearon distintas vistas las cuales ayudan al usuario a tener un mejor manejo del sistema, y a su vez segmentando todas las funcionalidades necesarias para evitar confusiones.

Palabras clave

1. Desarrollo
2. Procesamiento
3. Django
4. API

Abstract

Using Abstract Data Types (ADT), Django to create a friendly graphical user interface in web page format, we developed a system capable of processing files with extension ".XML" and store the data in databases which are read when starting the program to save them. Two types of servers were implemented for this practice with Flask, an API which is responsible for the management of everything related to the reading of files, generation of the readings, saving and reading of the database.

With the use of Flask we were able to create a server capable of reading n message files and n configuration files and ensuring the saving of all these in .XML files which will be read again when the program is started, thus preserving the program data despite closing it, shutting it down or even restarting the computer.

Using Django different views were created which help the user to have a better management of the system, and at the same time segmenting all the necessary functionalities to avoid confusion..

Keywords

1. *Development*
2. *Processing*
3. Django
4. API

Introducción

En el transcurso de este proyecto, se ha desarrollado un sistema de procesamiento de datos que se vale de Tipos de Datos Abstractos (TDA), la plataforma Django para la creación de una interfaz web de gran usabilidad y Graphviz para la generación de gráficos visuales impactantes. Este sistema se distingue por su capacidad para analizar y procesar archivos ".XML," permitiendo la posterior conservación de los datos en una base de datos XML que se carga al iniciar el proyecto, asegurando la retencion de información previamente ingresada.

El propósito fundamental de este sistema radica en la extracción de contenido en archivos ".XML," específicamente tweets. La representación de estos datos se lleva a cabo de manera más legible y accesible, clasificando los tweets según distintos criterios, como fechas, hashtags, usuarios y sentimientos. Este enfoque posibilita la comprensión y el análisis de la información.

Además, se ha prestado especial atención a la usabilidad, y para asegurar que el sistema sea accesible a una amplia audiencia, se ha implementado una interfaz gráfica intuitiva y amigable. Esta elección tiene como objetivo simplificar el uso del sistema, incluso para aquellos usuarios que carecen de experiencia en programación, de modo que puedan aprovechar al máximo sus capacidades sin complicaciones innecesarias.

Desarrollo del tema

Para el desarrollo de este sistema se implementaron algunos puntos importantes para la consturccion del mismo, De esta forma mantener un flujo de trabajo mas ordenado.

Los puntos principales en este trabajo fueron los siguiente:

- Back-end.
- Front-end.

A continuación se mostrara la escructura de código del sistemas donde se iniciara mostrando los TDA existente y sus respectivos métodos, además de que también pueden ser observados en el diagrama de clases que se encuentra en anexos.

Seguido se mostrara y describirá en funcionamiento de la GUI realizada con Django, realizada para mayor comodidad del usuario.

Back-end

Para la creación del back de esta apliacion se utilizo flaks y Python para toda la lógica.

A continuación se describirán todos los archivos y a su vez las funciones de cada uno.

1. Back.py

Encargado de las respuesta se le puede considedara el cerebro del api.

i. getHome

endpoint que utiliza el método get, y se implmenta simplemente para saber si el api esta arriba.

```python
@app.route("/")
def getHome():
    return jsonify({"message": "ipc2 desde un get ahora mismo"})
```

Figura 1. Imagen del código del programa
Fuente: elaboración propia.

ii. subir_mensaje

es la que se encarga de recibir el archivo para la lectura.

```python
@app.route('/grabarMensajes', methods=['POST'])
def subir_mensajes():
    if 'archivo' not in request.files:
        return jsonify({"message": "No se envio ningun archivo"})

    archivo = request.files['archivo']

    if archivo.filename == '':
        return 'No se seleccionó ningún archivo'

    archivo.filename = "back\\app\\enviadoApi\\" + archivo.filename
    archivo.save(archivo.filename)
    leerMensajes(archivo.filename)

    return jsonify({"message": "El archivo se subio exitosamente"})
```

Figura 2. Imagen del código del programa
Fuente: elaboración propia.

iii. subir_palabras

este se encarga de recibir el archivo de configuraciones.

```python
@app.route('/grabarPalabras', methods=['POST'])
def subir_Palabras():
    if 'archivo' not in request.files:
        return jsonify({"message": "No se envio ningun archivo"})

    archivo = request.files['archivo']

    if archivo.filename == '':
        return 'No se seleccionó ningún archivo'

    archivo.filename = "back\\app\\enviadoApi\\" + archivo.filename
    archivo.save(archivo.filename)
    leerConfig(archivo.filename)

    return jsonify({"message": "El archivo se subio exitosamente"})
```

Figura 3. Imagen del código del programa
Fuente: elaboración propia.

iv. borrarDatos

se encarga de inicializar el sistema.

```python
@app.route('/limpiarDatos', methods=['POST'])
def borrarDatos():
    try:
        limpiarDatos()
        return jsonify({"message": "Se inicializo con exito"})
    except:
        return jsonify({"message": "Se produjo un error, es posible que la DB ya este vacia"})
```

Figura 4. Imagen del código del programa
Fuente: elaboración propia.

v. estaVacio

verifica si el sistema esta inicializado o no.

```python
@app.route('/estaVacio', methods=['POST'])
def siEstaVacio():
    try:
        if estaVacion():
            return jsonify({"message": "Vacio"})
        else:
            return jsonify({"message": "No esta Vacio"})
    except:
        return jsonify({"message": "Error"})
```

Figura 5. Imagen del código del programa
Fuente: elaboración propia.

vi. docu

abre la documentación del proyecto.

```python
@app.route('/documentacion', methods=['GET'])
def docu():
    try:
        documentacion()
        return jsonify({"message": "Se abrira el archivo"})
    except:
        return jsonify({"message": "No se pudo abrir el archivo"})
```

Figura 6. Imagen del código del programa
Fuente: elaboración propia.

vii. getHastags

recibe un intervalo de fechas con las cuajes debe trabajar para crear un listado de todos los hashtags que se utilizaron en ese intervalo de tiempo.

```python
@app.route('/devolverHashtags/<fechas>', methods=['GET'])
def getHastags(fechas):
    try:
        return jsonify({"message": devovlerHas(fechas)})
    except:
        return jsonify({"message": "No se pudo abrir el archivo"})
```

Figura 7. Imagen del código del programa
Fuente: elaboración propia.

viii. getMenciones

el igual que la anterior recibe un intervalo de tiempo en fechas con el cual debe regresar todos los usuarios que se mencionaron en ese intervalo.

```python
@app.route('/devolverMenciones/<fechas>', methods=['GET'])
def getMenciones(fechas):
    try:
        return jsonify({"message": devovlerMen(fechas)})
    except:
        return jsonify({"message": "No se pudo abrir el archivo"})
```

Figura 8. Imagen del código del programa
Fuente: elaboración propia.

ix. getSentimiento

continuando con las anteriores esta no es la excepción, también recibe un intervalo de fechas con el cual deberá regresar un listado con el sentimiendo de los mensajes obtenidos en esas fechas.

```python
@app.route('/devolverSentimientos/<fechas>', methods=['GET'])
def getSentimientos(fechas):
    try:
        return jsonify({"message": devovlerSent(fechas)})
    except:
        return jsonify({"message": "No se pudo abrir el archivo"})
```

Figura 9. Imagen del código del programa
Fuente: elaboración propia.

2. app.py

esta tiene funciones que se ejecutan al momento de hacer una llamada a los distintos endpoint, y se genera un objeto principal con el que basamos todo el proyecto.

```
import os
from app.TDA.datos import Datos

#crear el objeto
obj = Datos("","")
```

Figura 10. Imagen del código del programa
Fuente: elaboración propia.

a. leerMensajes

se encarga de hacer la lectura respectiva del archivo de mensajes.

```
def leerMensajes(path):
    global obj
    obj.path[0] = path
    obj.leerMensajes()
```

Figura 11. Imagen del código del programa
Fuente: elaboración propia.

b. leerConfiguraciones

la función encargada de hacer la lectura del archivo de configuraciones.

```
def leerConfig(path):
    global obj
    obj.path[1] = path
    obj.leerConfiguraciones()
```

Figura 12. Imagen del código del programa
Fuente: elaboración propia.

c. limpiarDatos

esta función inicializa el objeto y a su vez borra las base de datos para que no se puedan rescatar de nuevo la información haciendo así una inicialización absoluta del sistema.

```
def limpiarDatos():
    global obj
    obj.inicilizar()
    os.remove("back\\app\\informacion\\MensajesDB.xml")
    os.remove("back\\app\\informacion\\PalabrasDB.xml")
```

Figura 13. Imagen del código del programa
Fuente: elaboración propia.

d. demás funciones

estas se encargan de realizar la veirificacion de que el sistema este inicializado, abrir la documentación y las ultimas realizan y devuelven las listas que se generan para hashtags, menciones y sentimientos de los mensajes.

```
def estaVacion():
    global obj
    return obj.vacio()

def documentacion():
    filepath = "Documentacion_Proyecto2_IPC2_202200075.pdf"
    os.system(f"start {filepath}")

def devovlerHas(fechas):
    global obj
    return obj.has(fechas)

def devovlerMen(fechas):
    global obj
    return obj.men(fechas)

def devovlerSent(fechas):
    global obj
    return obj.sent(fechas)
```

Figura 14. Imagen del código del programa
Fuente: elaboración propia.

3. carpeta DTA

esta contiene las clases de todos los objetos utilizados en este proyecto contando con datos la clase del objeto principal en este proyecto con el cual hacermos las lecturas y la inicialización.
Cuenta con mensajes el cual tiene un atributo de texto y uno de fecha para almacenar todo lo necesario de un mensaje.
Y con la clase palabras la cual tiene 4 atributos positivas, negatias, positivas rechazadas, negativas rechazadas.

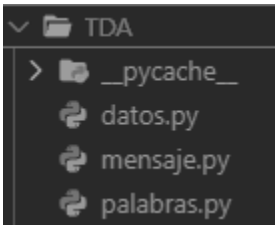

Figura 15. Imagen del código del programa
Fuente: elaboración propia.

4. Carpeta información

Esta carpeta guarda los datos durmientes del proyecto estos archivos almacenan las información mientras el proyecto no esta en ejecucion, así sirviendo como base de datos.

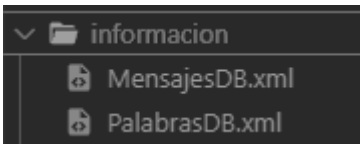

Figura 16. Imagen del código del programa
Fuente: elaboración propia.

Front-end

Se hablara de Django con el cual se realizo todo el fron end del proyecto, se mencionaran aspector importantes del html aunque únicamente si es algo relevante ya que se considera que todo eso no es relevante ni necesario de explicar, pero si todos lo relacionado con la vista de la interfaz sera explicado.

1. /

Siendo esta la pagina principal es la mas sencilla de todas, cuenta con un mensajes indicando que es la pagina de inicio y el nombre del alumno que la cre.

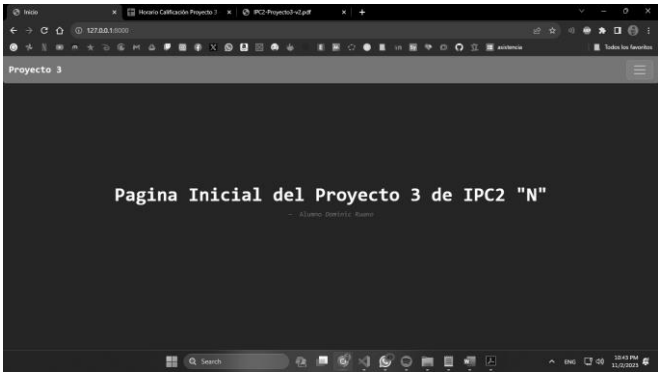

Figura 17. Imagen del código del programa
Fuente: elaboración propia.

2. Interfaz de selección

A esta se accede desde el botno superior derecho de color azul que se encuentra a la vista durante todo el recorrido de la pagina.

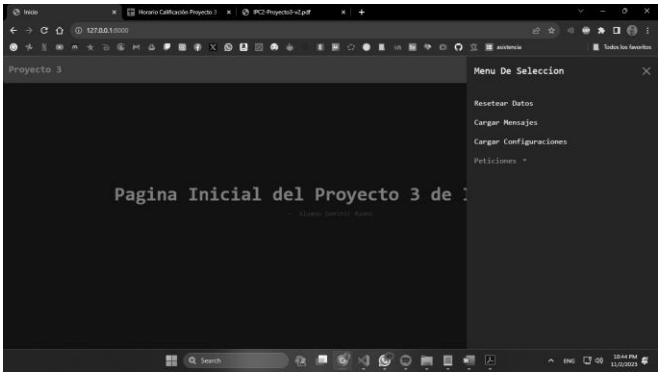

Figura 18. Imagen del código del programa
Fuente: elaboración propia.

3. Resetear datos

Esta seria la vista general de la pestaña de resetear datos.

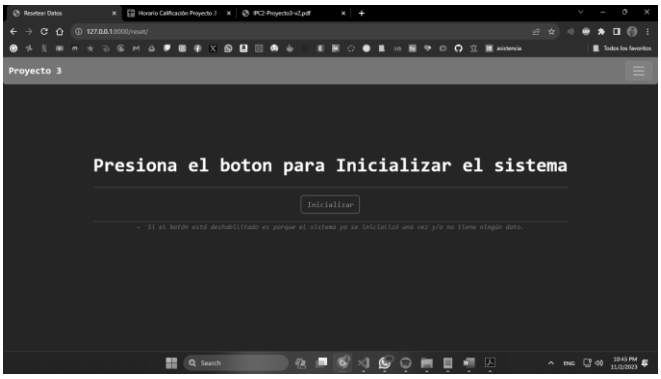

Figura 19. Imagen del código del programa
Fuente: elaboración propia.

Además el botón solo puede ser presionado si el sistema detecta que se están guardando datos en el mismo.

4. Cargar mensajes y configuraciones

Pagina donde se debe subir un archivo y dar en el botón para enviar a su respectivo endpoint para la carga de esos datos

5. Peticiones
    a. Consultar hashtags, menciones, sentimientos

Esta se encarga de la visualización de los listados según el intervalo de tiempo que se muestre.

6. Ayuda

Esta tiene la opción de mostrar los datos y de abrir la respectiva documentación.

Conclusiones

El sistema desarrollado ha demostrado ser efectivo en la mejora de la accesibilidad de datos contenidos en archivos XML, particularmente en lo que respecta a tweets. La clasificación por fechas, hashtags, usuarios y sentimientos proporciona a los usuarios una visión más organizada y legible de la información, lo que puede facilitar la toma de decisiones basadas en estos datos.

La implementación de una interfaz gráfica amigable se ha mostrado exitosa en la eliminación de barreras para usuarios no técnicos o sin experiencia en programación. Esto amplía la audiencia y hace que el sistema sea accesible para una gama más amplia de usuarios, permitiéndoles aprovechar sus capacidades sin tener que aprender a utilizar herramientas de programación.

La capacidad de almacenar y retener datos previamente ingresados en una base de datos XML facilita la gestión de la información a largo plazo. Esto garantiza que los datos no se pierdan y que los usuarios puedan acceder a ellos de manera rápida y efectiva en futuras interacciones con el sistema. Esto puede ser valioso tanto para la toma de decisiones como para el seguimiento de tendencias a lo largo del tiempo.

este proyecto ha demostrado ser exitoso en mejorar la accesibilidad de datos, facilitar la interacción para usuarios no técnicos y aumentar la eficiencia en la retención de información, lo que lo convierte en una herramienta valiosa para la gestión y análisis de datos contenidos en archivos XML, en particular, datos de redes sociales como tweets.

Referencias bibliográficas

Bootstrap. (s.f.). Introduction. Recuperado de https://getbootstrap.com/docs/5.3/getting-started/introduction/

Anexos

Diagrama de clases:

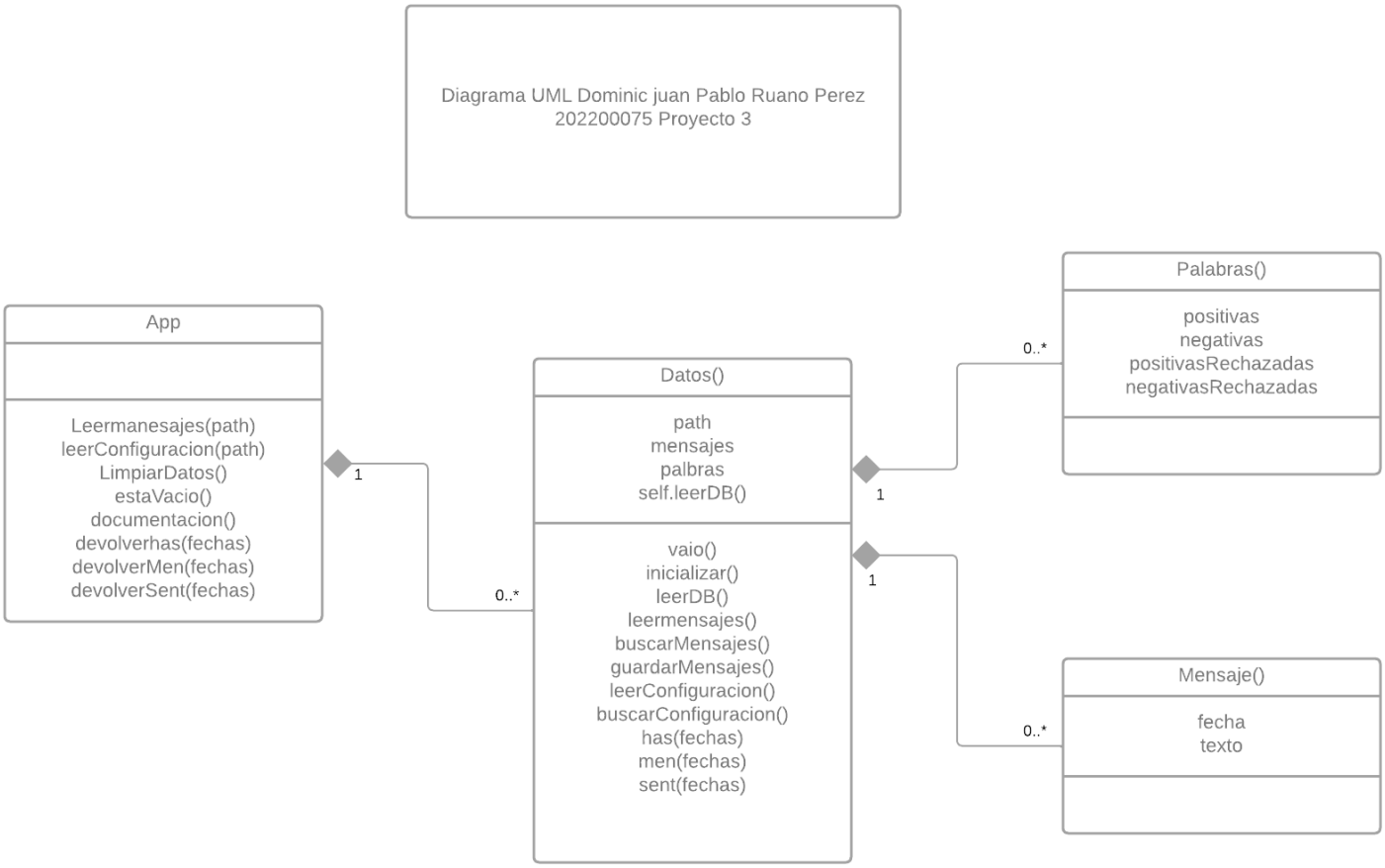

Figura 20. Diagrama de clases.
Fuente: elaboración propia.