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 MANUAL TECNICO

202207639 – Diego Josue Guevara Abaj

Resumen

El objetivo general del proyecto es desarrollar una solución integral que implemente un API utilizando el protocolo HTTP y la programación orientada a objetos (POO) junto con bases de datos. Los objetivos específicos incluyen la implementación de un API en Python para consumirlo a través del protocolo HTTP, el uso de programación orientada a objetos, la persistencia de datos en bases de datos, la utilización de archivos XML para la comunicación con el API y el uso de expresiones regulares para extraer contenido de texto. Esta herramienta es capaz de leer mensajes de Twitter, identificando referencias a otros usuarios y hashtags en el texto. Se ha establecido un diccionario de datos para determinar si un mensaje es positivo, negativo o neutro en función de las palabras presentes. El programa a desarrollar almacenará información en formato XML después de recibir mensajes de Twitter y clasificará cada mensaje como positivo, negativo o neutro. El archivo de salida contendrá reportes y consultas basadas en la información almacenada

Palabras clave

1. "Integración"
2. "Persistencia de datos"
3. "Referencia"
4. “Análisis"
5. "Comunicación"

Abstract

General project objective is to develop a comprehensive solution that implements an API using the HTTP protocol and Object-Oriented Programming (OOP) along with databases. Specific objectives include the implementation of a Python API for consumption via the HTTP protocol, using OOP, persisting data in databases, employing XML files for API communication, and utilizing regular expressions to extract text content. This tool is capable of reading Twitter messages, identifying references to other users and hashtags in the text. A data dictionary has been established to determine whether a message is positive, negative, or neutral based on the words present. The program to be developed will store information in XML format after receiving Twitter messages and classify each message as positive, negative, or neutral. The output file will contain reports and queries based on the stored information.

Keywords

1. Integration

2. Data Persistence

3. Reference

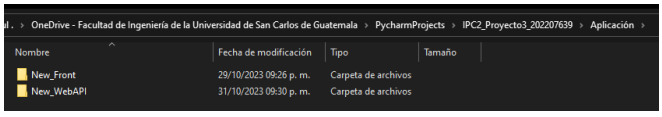
4. Analysis

5. Communication

Introducción

Bienvenido al Manual del Tecnico de la solución integral desarrollada por Tecnologías Chapinas, S.A. Este manual te guiará en el uso de un API que analiza el contenido de redes sociales, especialmente mensajes de Twitter, para determinar el sentimiento de los usuarios hacia esos mensajes. Nuestra herramienta utiliza programación orientada a objetos, bases de datos y archivos XML, junto con expresiones regulares para extraer datos. A lo largo de este manual, comprenderás los objetivos del proyecto, las reglas de procesamiento de mensajes de Twitter y cómo se clasifican como positivos, negativos o neutros. También aprenderás cómo almacenar datos en formato XML y realizar consultas. Este manual es adecuado para usuarios de todos los niveles de experiencia, desde principiantes hasta expertos en programación y análisis de datos. Explora cada sección detenidamente para aprovechar al máximo esta poderosa herramienta. ¡Esperamos que este manual te sea de gran ayuda en tu interacción con la solución de Tecnologías Chapinas, S.A.!

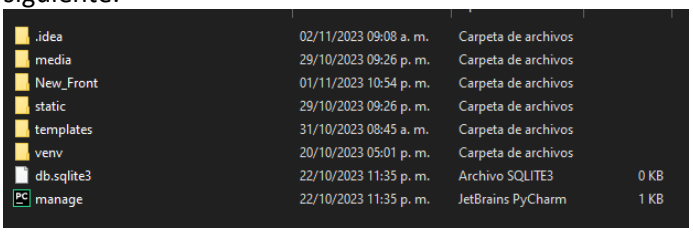
Desarrollo del tema



.

En primer lugar, el programa se encuentra dividido en 2 partes, en la sección del front se maneja todo lo relacionado con Django, y en el apartado del webAPI, se encuentra el apartado del flask.

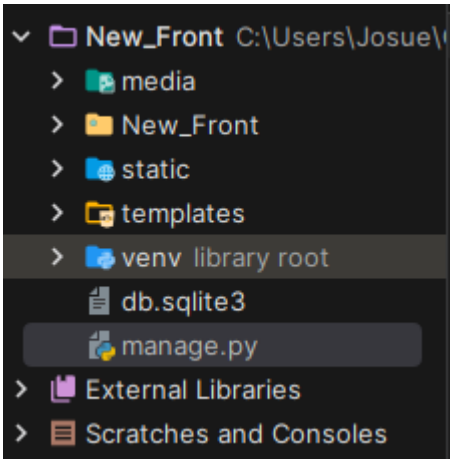
En la cuestión del apartado del Front se encuentra lo siguiente:



En donde para poder iniciar lo que es el programa, se usa la consola en esta ubicación y se coloca el siguiente comando:

“Python manage.py runserver”, con el cual dará inicio al servidor de django, con esto ya se podrá acceder en un navegador a la ruta predefinida.

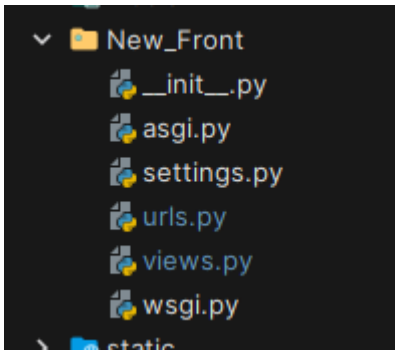
Con esto en la consola confirmamos que tenemos activo el servidor por parte de Django.



El apartado del front se encuentra compuesto por la carpeta media, new_front, static, templates, venv y manage.py

En la carpeta de media se encuentra lo que es un icono para la pagina web únicamente.

En la carpeta de new_front:



Se encuentran todas las configuraciones en donde procede settings, urs.py, views.py. las cuales fueron en donde se hicieron las modificaciones pertinentes para poder utilizar el proyecto.

Vamos a desglosar las partes clave de este archivo de configuración:

1. **BASE_DIR**: Esta variable define la ubicación del directorio principal del proyecto. Se calcula utilizando la ruta del archivo actual y se va subiendo dos niveles para llegar al directorio raíz del proyecto.
2. **INSTALLED_APPS**: Esta lista contiene los nombres de las aplicaciones que están instaladas y habilitadas en el proyecto Django. Las aplicaciones son componentes reutilizables que se pueden agregar a tu proyecto para proporcionar funcionalidades específicas.
3. **MIDDLEWARE**: Los middleware son componentes que procesan las solicitudes HTTP antes de que lleguen a las vistas. Se utilizan para aplicar transformaciones, validaciones y configuraciones a nivel de solicitud. En este apartado se agregaron configuraciones crsf, para poder evitar el problema 403.
4. **TEMPLATES**: Esta configuración define cómo se renderizan las plantillas de HTML en tu proyecto. Puedes especificar la ubicación de las plantillas y configurar opciones adicionales.
5. **MESSAGES_SERVER_IP** y otras variables: Estas son variables personalizadas que parecen contener direcciones IP o URL de servicios externos utilizados por la aplicación.
6. **STATIC_URL** y **STATICFILES_DIRS**: Estas configuraciones están relacionadas con la gestión de archivos estáticos como CSS, JavaScript e imágenes.
7. **DEFAULT_AUTO_FIELD**: Configura el tipo de campo que se utilizará como clave primaria para los modelos de base de datos. En este caso, se utiliza 'django.db.models.BigAutoField'.
8. **MEDIA_URL** y **MEDIA_ROOT**: Estas configuraciones se utilizan para gestionar archivos multimedia subidos por los usuarios, como imágenes o archivos adjuntos.

En el archivo de las urls representa la configuración de las URL para el proyecto Django. En este archivo, se define cómo se deben manejar las URL de la aplicación, es decir, cómo se asignan a vistas específicas.

1. **from django.urls import path**: Importa la función **path** de la biblioteca **django.urls**, que se utiliza para definir rutas de URL.
2. **urlpatterns**: Es una lista de rutas URL que se asignan a vistas específicas. Cada elemento de la lista consiste en una llamada a la función **path** que especifica la URL a la que se asocia una vista y un nombre que se utiliza para hacer referencia a esa URL en el código de la aplicación.

A continuación, se describen las rutas definidas en **urlpatterns** y las vistas asociadas:

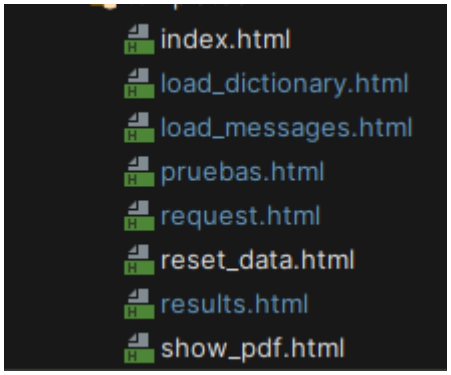
- **path('', index, name='index')**: Esta ruta está asociada a la vista **index** y se utiliza para la página principal de la aplicación. El nombre **index** se usa como una referencia en el código para generar URL de manera dinámica.
- **path('reset/', reset_data, name='reset_data')**: Esta ruta se asocia a la vista **reset_data** y se utiliza para restablecer datos en la aplicación.
- **path('load_messages/', load_messages, name='load_messages')**: Esta ruta está asociada a la vista **load_messages** y se utiliza para cargar mensajes en la aplicación.

- **path('load_dictionary/', load_dictionary, name='load_dictionary')**: Asocia la vista **load_dictionary** y se utiliza para cargar un diccionario en la aplicación.
- **path('requests/', requests, name='requests')**: Asocia la vista **requests** y se utiliza para gestionar solicitudes en la aplicación.
- **path('upload_file', upload_file, name='upload_file')**: Esta ruta se asocia a la vista **upload_file** y se utiliza para cargar archivos en la aplicación.
- **path('results/', results, name='results')**: Asocia la vista **results** y se utiliza para mostrar resultados en la aplicación.
- **path('show_pdf/', show_pdf, name='show_pdf')**: Asocia la vista **show_pdf** y se utiliza para mostrar un documento PDF.
- **path('download_xml/', download_xml, name='download_xml')**: Asocia la vista **download_xml** y se utiliza para descargar archivos XML de resumen.

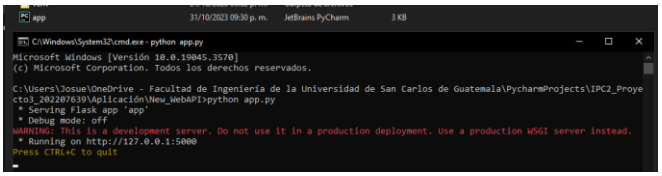
En el apartado de views en donde se encuentran las funciones que manejan las solicitudes web y generan respuestas.

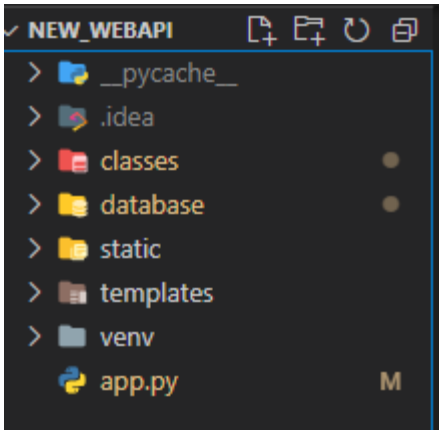
1. **index(request)**: Esta vista se utiliza para renderizar la página principal de la aplicación. En este caso, se devuelve una plantilla llamada '**index.html**'.
2. **reset_data(request)**: Esta vista maneja una solicitud POST. Se comunica con un servidor externo para realizar alguna acción, para restablecer datos. El resultado de esta comunicación se muestra en una plantilla llamada '**reset_data.html**'.
3. **load_messages(request)**: Esta vista se utiliza para renderizar una página donde se pueden cargar mensajes.
4. **load_dictionary(request)**: Similar a la vista anterior, esta se utiliza para renderizar una página donde se pueden cargar diccionarios.
5. **upload_file(request)**: Esta vista maneja una solicitud POST para cargar archivos. Dependiendo del valor de **source**, la vista decide a qué servidor enviar el archivo. Si la carga es exitosa, se muestra el contenido del archivo XML en una página llamada '**download_xml**'.
6. **requests(request)**: Esta vista maneja solicitudes GET con parámetros, como **action**, **date_1** y **date_2**. Según la acción, se realiza una solicitud a un servidor externo y se muestra el resultado en la página '**results.html**'.
7. **results(request)**: Esta vista se utiliza para renderizar la página de resultados. Puede mostrar datos obtenidos de solicitudes externas o cualquier otro tipo de resultados.
8. **show_pdf(request)**: Esta vista muestra un archivo PDF al usuario. Lee el archivo PDF, lo convierte en una cadena codificada en base64 y lo pasa a la plantilla '**show_pdf.html**'.
9. **download_xml(request)**: Esta vista se utiliza para descargar un archivo XML. El contenido del archivo XML se almacena en la variable **xml_string** y se devuelve al usuario como una respuesta con un encabezado de descarga.

Luego se encuentra el apartado de templates en donde se manejan todos los html necesarios para el programa.



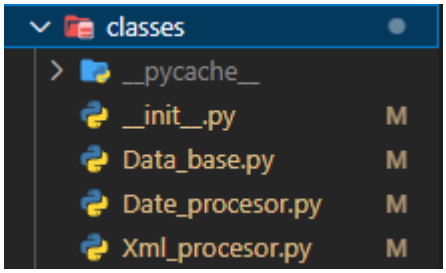
En la cuestión del flask en el apartado de lo que es la webAPI, se corre el archivo app.py





En este apartado se manjan las carpetas clases, database, y el archivo app.py, únicamente.

En el directorio de clases se tiene:



En la clase data_base que se utiliza para manejar datos y realizar operaciones en archivos XML. Esta clase tiene varios métodos y se utiliza para procesar datos almacenados en archivos XML y realizar tareas como la obtención de datos por fechas, procesamiento de sentimientos, limpieza de bases de datos y más.

1. **__init__(self)**: Este es el método constructor de la clase. Inicializa algunas variables, como **date_1**, **date_2**, **data_type**, **sentiment_dict** y **root**, que se utilizan en otros métodos.
2. **set_dates(self, date_1, date_2)**: Este método se utiliza para establecer las fechas **date_1** y **date_2** que se utilizarán en las operaciones posteriores.
3. **process_data(self, data_type)**: Este método procesa datos a partir de un archivo XML en función del tipo de datos proporcionado. Si el tipo de datos es "SENTIMIENTOS", se procesan datos relacionados con sentimientos. En otros casos, se obtienen datos por fechas. El resultado se devuelve como un diccionario.
4. **get_data_by_date(self)**: Este método obtiene datos por fecha a partir de un archivo XML. Filtra los datos en función de las fechas **date_1** y **date_2**, y devuelve un diccionario de datos.
5. **sentiment_dictionary(self)**: Este método procesa un archivo XML llamado "dictionary.xml" para crear un diccionario de sentimientos. El diccionario se almacena en **self.sentiment_dict**.
6. **get_feelings_by_date(self)**: Este método obtiene datos de sentimientos por fecha a partir de un archivo XML. Utiliza el diccionario de sentimientos creado anteriormente para clasificar los datos en "Positivo", "Negativo" o "Neutro".
7. **clear_dictionary_db(self)**: Este método elimina todas las palabras del diccionario almacenado en "dictionary.xml".
8. **clear_messages_db(self)**: Este método elimina todos los elementos de tipo 'MENSAJE' en un archivo XML llamado "messages.xml".
9. **clear_all(self)**: Este método llama a **clear_dictionary_db()** y **clear_messages_db()** para realizar una limpieza completa de los archivos XML.

En la clase Date_procesor que se utiliza para procesar datos en función de las solicitudes recibidas en una aplicación. Esta clase depende de la clase **Data_base** que procesa datos almacenados en archivos XML, como se mencionó anteriormente.

1. **__init__(self)**: Este es el constructor de la clase. Inicializa algunas variables y crea una instancia de la clase **Data_base** para trabajar con los datos.
2. **format_dates(self, request)**: Este método toma una solicitud (request) y extrae las fechas **date_1** y **date_2** de la solicitud. Luego, verifica si las fechas se proporcionaron correctamente y si el formato es válido. Si las fechas son válidas, las convierte en objetos **datetime** y las establece en la instancia de **Data_base**. Devuelve un diccionario que contiene las fechas o un mensaje de error si hubo problemas.
3. **get_hashtags(self, request)**: Este método se utiliza para obtener datos relacionados con hashtags en función de las fechas

proporcionadas. Primero llama a **format_dates** para obtener las fechas. Luego, utiliza la instancia de **Data_base** para procesar los datos de tipo 'HASHTAGS' y calcular el recuento total de hashtags en las fechas especificadas. Devuelve un diccionario con los resultados.

4. **get_mencions(self, request)**: Similar a **get_hashtags**, este método obtiene datos relacionados con menciones en función de las fechas proporcionadas. Utiliza la instancia de **Data_base** para procesar los datos de tipo 'USUARIOS' y calcular el recuento total de menciones en las fechas especificadas.
5. **get_feelings(self, request)**: Este método se utiliza para obtener datos relacionados con sentimientos en función de las fechas proporcionadas. Llama a **format_dates** para obtener las fechas, y luego utiliza la instancia de **Data_base** para procesar los datos de tipo 'SENTIMIENTOS' y calcular el recuento total de sentimientos (Negativo, Neutro, Positivo) en las fechas especificadas.
6. **clear_db(self)**: Este método llama a **clear_all** de la instancia de **Data_base** para realizar una limpieza completa de las bases de datos, lo que incluye la eliminación de palabras del diccionario y elementos 'MENSAJE' en archivos XML.

Y en la clase xml_procesor que se utiliza para procesar archivos XML y realizar diversas operaciones, como la extracción de datos y la actualización de bases de datos.

1. **__init__(self)**: Este es el constructor de la clase. Inicializa las variables **request** y **response_data**, que se utilizan para procesar archivos XML y almacenar los datos resultantes.
2. **process_xml_files(self, request, type_file)**: Este método toma una solicitud HTTP que contiene archivos XML y procesa estos archivos. Puede procesar dos tipos de archivos: "messages" y "feelings". Dependiendo del tipo, realiza operaciones específicas.
    - Para archivos de tipo "messages", crea un resumen de mensajes recibidos, guardando la información en un nuevo archivo XML llamado "resumenMensajes.xml". Luego, actualiza una base de datos original ("messages.xml") con los mensajes nuevos y ordena los mensajes por la fecha. Devuelve el resumen de mensajes como una cadena XML formateada.
    - Para archivos de tipo "feelings", agrega palabras asociadas con sentimientos positivos y negativos a un archivo XML de diccionario ("dictionary.xml"). También crea un resumen de la configuración y lo guarda en un archivo XML llamado "resumenConfig.xml". Devuelve el resumen de configuración como una cadena XML formateada.
3. **extract_messages(self)**: Este método se utiliza para extraer mensajes de una lista de datos JSON. Busca mensajes dentro de la estructura JSON utilizando la clave "MENSAJES". Devuelve una lista de mensajes, donde cada mensaje es un diccionario.
4. **extract_feelings(self)**: Este método extrae palabras asociadas con sentimientos positivos y negativos de una lista de datos JSON. Busca estas palabras en la estructura JSON utilizando las claves "diccionario", "sentimientos_positivos" y "sentimientos_negativos". Devuelve una lista de palabras, donde cada palabra se representa como un diccionario con campos 'type' y 'data'.
5. **format_xml(self, element)**: Esta función formatea un elemento XML y sus elementos hijos, eliminando espacios en blanco innecesarios al principio y al final del texto de cada elemento.
6. **save_messages_in_db(self, messages)**: Este método guarda los mensajes recibidos en la base de datos. Primero crea un resumen de mensajes en un nuevo archivo XML y lo guarda. Luego, actualiza la base de datos original con los mensajes nuevos, ordena los mensajes por fecha y guarda el archivo XML resultante. Devuelve el resumen de mensajes como una cadena XML formateada.
7. **save_dictionary_in_db(self, feelings_list)**: Este método agrega palabras asociadas con sentimientos positivos y negativos a un archivo XML de diccionario. También crea un resumen de configuración y lo guarda en un archivo

XML. Devuelve el resumen de configuración como una cadena XML formateada.

Conclusiones

Esta sección debe orientarse a evidenciar claramente las principales ideas generadas, propuestas que deriven del análisis realizado y si existen, expresar las conclusiones o aportes que autor quiera destacar.

Enfatizando, lo importante es destacar las principales posturas fundamentadas del autor, que desea transmitir a los lectores.

Adicionalmente, pueden incluirse preguntas abiertas a la reflexión y debate, temas concatenados con el tema expuesto o recomendaciones para profundizar en la temática expuesta.

Referencias bibliográficas

Máximo 5 referencias en orden alfabético.

C. J. Date, (1991). *An introduction to Database Systems.* Addison-Wesley Publishing Company, Inc.

Extensión: de cuatro a siete páginas como máximo

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.