Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
IPC2, 2do. Semestre 2023.

DOCUMENTACION - PROYECTO NO. 3 -

201212891 – Edgar Rolando Ramírez López

Resumen

El programa de Manejo de Mensajes es una aplicación web realizada con flask y django, para el FrontEnd se utulizo Django que se integra con una aplicación BackEnd Flask mediante solicitudes HTTP y se creo una API. Las vistas y funciones en este archivo permiten cargar archivos XML, configuraciones y consultas de datos, así como generar informes PDF. El código se comunica con Flask a través de rutas definidas, como la carga de archivos y la consulta de datos de hashtags, usuarios y sentimientos. Las solicitudes POST y GET se utilizan para intercambiar datos entre las dos aplicaciones. Se emplean bibliotecas como requests, reportlab, matplotlib, numpy, y xml.dom.minidom para llevar a cabo tareas como procesar y formatear datos XML, generar informes PDF con tablas y gráficas de barras, y realizar solicitudes HTTP.

Palabras clave

Python, funciones, clases, API, librerías, Frontend, Backend

Abstract

The Message Management program is a web application developed using Flask and Django. Django was used for the FrontEnd, seamlessly integrated with a Flask BackEnd through HTTP requests. The views and functions in this file allow for the uploading of XML files, configurations, data queries, and PDF report generation. The code communicates with Flask through defined routes, such as file uploads and data queries for hashtags, users, and sentiments. POST and GET requests are used to exchange data between the two applications. Libraries like requests, reportlab, matplotlib, numpy, and xml.dom.minidom are employed to perform tasks like processing and formatting XML data, generating PDF reports with tables and bar charts, and making HTTP requests.

Keywords

Python, functions, class, lists, libraries, API, Frontend, Backend

Introducción

La aplicación "Manejo de Mensajes" combina las potentes capacidades de Flask y Django para brindar una solución web versátil. En el frente, Django facilita la creación de una interfaz de usuario amigable. Por detrás, Flask impulsa un robusto backend que procesa archivos XML, configura datos y consulta información para generar informes PDF detallados. La comunicación fluida entre estas dos aplicaciones se logra mediante solicitudes HTTP, donde las rutas definen el flujo de datos. Con el uso de bibliotecas como requests, reportlab, matplotlib, numpy, y xml.dom.minidom, se logra procesar datos complejos y crear informes visuales atractivos. Este programa representa una solución integral para el manejo de datos de hashtags, usuarios y sentimientos, brindando una experiencia de usuario completa.

Desarrollo del tema

En el desarrollo de la práctica se utilizó el lenguaje Python 3.11.1, el IDE utilizado fue Visual Studio Code y también se subió el proyecto a GitHub. El paradigma utilizado en el proyecto fue la programación con orientación a objetos. Las librerías utilizadas fueron xml.etree.ElementTree, re, cors, etc.

a. Python

Python es un lenguaje de programación versátil, de código abierto y multiplataforma que se usa para desarrollo web y aplicaciones.

b. Django

Django es un marco web de Python que facilita la creación de aplicaciones web seguras, escalables y de alto rendimiento.

c. Flask

Flask es un marco web ligero en Python que simplifica la creación de aplicaciones web con rutas y vistas.

d. Visual Code Studio (VSCODE)

Visual Studio Code (VSCode) es un IDE de código abierto y gratuito desarrollado por Microsoft para programadores. Ofrece soporte para varios lenguajes y es ampliamente utilizado.

e. GitHub

GitHub es una plataforma de alojamiento web para desarrolladores, que utiliza Git para colaborar en proyectos, revisar código y más.

f. API

Una API (Interfaz de Programación de Aplicaciones) es un conjunto de reglas y protocolos que permiten a diferentes softwares comunicarse y colaborar entre sí.

g. Librería xml.etree.ElementTree

ElementTree es una librería Python estándar que simplifica análisis y manipulación eficiente de documentos XML, permitiendo lectura y escritura.

h. RE

La biblioteca "re" de Python es una herramienta para trabajar con expresiones regulares, permitiendo buscar y manipular patrones de texto.

i. flask_cors

Es una librería de Python para gestionar políticas de seguridad en aplicaciones Flask, permitiendo el manejo de solicitudes cruzadas.

j. XML (Extensible Markup Language)

XML es un lenguaje de marcado que estructura y transporta datos en texto legible, usado para información legible por humanos y máquinas.

k. JSON

JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos utilizado para representar información estructurada, similar a un diccionario.

l. Datetime

La librería "datetime" en Python proporciona herramientas para trabajar con fechas y horas, permitiendo su manipulación y cálculos precisos.

m. ReportLab

Biblioteca Python que permite crear documentos PDF personalizados mediante programación, con gráficos, tablas y texto.

n. Librería "io"

Utilizada para trabajar con flujos de entrada/salida de datos, como archivos, cadenas y bytes.

o. Matplotlib

Librería de Python para crear gráficos y visualizaciones de datos de forma sencilla y personalizable.

p. NumPy

Biblioteca de Python para manipulación de arrays y cálculos numéricos eficientes, esencial para ciencia de datos.

q. Requests

Librería Python que simplifica las solicitudes HTTP, permitiendo interactuar con servicios web y recuperar datos.

r. librería "os"

Proporciona funciones para interactuar con el sistema operativo, como acceder a archivos, directorios y variables de entorno.

s. pathlib

La librería "pathlib" en Python facilita la manipulación de rutas de archivos y directorios de manera eficiente y sencilla.

Estructura del Código

Para comprender mejor la estructura de tu programa, es necesario considerar tanto el código de la aplicación Django como la parte correspondiente a la aplicación Flask (que está siendo invocada desde Django).

Aplicación Django

Django es el marco de trabajo principal que maneja las solicitudes web y las vistas. La estructura básica de una aplicación Django suele seguir el patrón Modelo-Vista-Controlador (MVC). En este caso, las vistas están definidas en views.py.

- **Definición de funciones y vistas**

Las vistas de Django se definen como funciones, y cada vista se encarga de una tarea específica de la aplicación. A continuación, se describen las vistas definidas en el código:

- **home(request):** Renderiza la página de inicio de la aplicación.

- **cargar_archivo(request):** Se encarga de cargar un archivo XML en la aplicación Flask, procesar la respuesta y mostrar el contenido formateado de los archivos XML generados por Flask.

- **cargar_configuracion(request):** Similar a cargar_archivo, esta vista se encarga de cargar un archivo de configuración en la aplicación Flask y mostrar el contenido formateado de los archivos XML generados por Flask.

- **resetear_datos(request):** Realiza una solicitud POST a la ruta de Flask para resetear los datos.

- **consultar_hashtags(request):** Permite consultar hashtags en mensajes en un rango de fechas específico y muestra los resultados en la plantilla.

- **generar_reporte_hashtags_pdf(request):** Genera un informe en formato PDF que resume la cantidad de mensajes para cada hashtag.

- **consultar_usuario(request):** Permite consultar usuarios mencionados en mensajes en un rango de fechas y muestra los resultados en la plantilla.

- **generar_reporte_usuarios_pdf(request):** Genera un informe en formato PDF que resume la cantidad de menciones para cada usuario.

- **consultar_sentimientos(request):** Permite consultar sentimientos en mensajes en un rango de fechas y muestra los resultados en la plantilla.

- **generar_reporte_sentimientos_pdf(request):** Genera un informe en formato PDF que resume los sentimientos en los mensajes y muestra una gráfica de barras resumida.

Aplicación Flask

Flask es otro marco de trabajo web, que actúa como el backend de la aplicación. A través de Flask, se realizan acciones como cargar archivos XML, procesar datos y proporcionar una API para consultar datos de hashtags, usuarios y sentimientos.

- **Creación de una instancia de Flask**

Se crea una instancia de la aplicación Flask utilizando Flask(__name__) y se almacena en la variable app.

- **Configuración de CORS:**
  Se utiliza la extensión flask_cors para habilitar el manejo de solicitudes CORS (Cross-Origin Resource Sharing) permitiendo que esta API sea accesible desde dominios diferentes.

Rutas de la API:

El código define múltiples rutas para la API que se gestionarán con distintas funciones. Cada ruta puede aceptar solicitudes HTTP GET o POST, dependiendo de su función.

- **/cargar-xml:** Esta ruta maneja las solicitudes POST para cargar un archivo XML. El archivo XML se procesa para extraer mensajes, y se actualiza un archivo XML llamado "DB_TWEETS.xml" con los mensajes procesados.

- **/cargar-configuracion:** Esta ruta maneja las solicitudes POST para cargar un archivo XML de configuración. Se cargan las palabras de sentimientos positivos y negativos desde el archivo de configuración y se actualiza un archivo "DB_CONFIGS_DICT.xml" con las palabras cargadas.

- **/consultar-hashtags:** Esta ruta maneja las solicitudes GET para consultar los hashtags en un rango de fechas. Se procesa el archivo XML "uploaded.xml" para contar los hashtags por fecha y se devuelve la información en formato JSON.

- **/consultar-sentimientos:** Esta ruta maneja las solicitudes GET para consultar los sentimientos en un rango de fechas. Se procesa el archivo XML "DB_TWEETS.xml" y el archivo de configuración "DB_CONFIGS_DICT.xml" para calcular los sentimientos en función de palabras positivas y negativas y se devuelve la información en formato JSON.

- **/consultar-usuarios:** Esta ruta maneja las solicitudes GET para consultar los usuarios mencionados en un rango de fechas. Se procesa el archivo XML "DB_TWEETS.xml" para contar los usuarios mencionados por fecha y se devuelve la información en formato JSON.

- **/resetear-datos:** Esta ruta maneja las solicitudes POST para restablecer los datos. Se eliminan archivos de resumen y configuración, y se reinician las variables relacionadas con los mensajes y los sentimientos.

Clase Mensajes

Esta clase representa los mensajes y tiene métodos para obtener texto, hashtags, usuarios mencionados y para reiniciar los mensajes.

Clase Sentimientos

Esta clase maneja las palabras de sentimientos positivos y negativos y tiene un método para reiniciar estos conjuntos.

Comunicación Django - Flask

La comunicación entre Django y Flask se realiza principalmente a través de solicitudes HTTP, utilizando la biblioteca requests en Django para realizar solicitudes GET y POST a las rutas definidas en Flask.

Flujo de Trabajo General

El flujo de trabajo de tu programa puede describirse de la siguiente manera:

Cuando un usuario interactúa con la aplicación, Django maneja las solicitudes web y determina qué vista de views.py se debe ejecutar.

Las vistas en views.py pueden realizar una variedad de acciones, como cargar archivos XML, interactuar con la aplicación Flask y renderizar plantillas HTML.

Cuando es necesario interactuar con la aplicación Flask, se realizan solicitudes HTTP a rutas definidas en la aplicación Flask. La comunicación entre Django y Flask se logra utilizando la biblioteca requests.

La aplicación Flask (que se encuentra en el directorio Backend) procesa estas solicitudes, realiza acciones como cargar archivos, consultar datos y responder a las solicitudes de Django.

Flask puede acceder a archivos XML, procesarlos, realizar consultas en bases de datos o realizar otras acciones necesarias para responder a las solicitudes de Django.

Una vez que la aplicación Flask ha procesado la solicitud, envía una respuesta a Django, que luego se utiliza para renderizar plantillas HTML o generar informes en formato PDF.

Plantillas HTML

Las plantillas HTML se utilizan para mostrar datos al usuario final. En Django, estas plantillas se almacenan en el directorio templates y se pueden renderizar con datos proporcionados por las vistas.

Informes en Formato PDF

El código incluye la capacidad de generar informes en formato PDF utilizando la biblioteca reportlab. Estos informes contienen datos resumidos, como la cantidad de mensajes por hashtag, menciones de usuarios y sentimientos. Estos informes se generan y se envían al usuario final como respuestas en formato PDF.

Estructura Modular

La estructura general del programa está diseñada de manera modular, con separación de responsabilidades entre Django (encargado de las vistas y la interacción con el usuario) y Flask (encargado de las operaciones de backend y procesamiento de datos).

Uso del Programa

La aplicación web de Manejo de Mensajes se utiliza para procesar archivos XML, consultar hashtags, sentimientos y usuarios mencionados en función de un rango de fechas y reiniciar los datos.

Antes de utilizar la aplicación, es importante asegurarse de que el backend desarrollado en Flask esté en funcionamiento. Si se proporciona una aplicación web basada en Django, asegúrate de que esté configurada y en funcionamiento. Por lo que para correr ambas al mismo tiempo y el funcionamiento este en lo correcto se debe iniciar el backend con el comando:

$$python\ run\ app.py$$

Y el frontend de django se debe iniciar con el comando:

$$python\ manage.py\ runserver$$

Figura 1. Página de inicio

Fuente: elaboración propia

- **Carga de Archivos XML**

1. Abre la aplicación web en tu navegador.
2. Accede a la sección "Cargar Archivos XML" o la ruta correspondiente, dependiendo de la interfaz proporcionada.
3. Selecciona un archivo XML para cargar.
4. Haz clic en el botón "Cargar" o la acción equivalente.
5. El archivo XML se procesará y se actualizará el archivo "DB_TWEETS.xml" en el servidor.

- **Carga de Archivos de Configuración**

1. Abre la aplicación web en tu navegador.
2. Accede a la sección "Cargar Archivos de Configuración" o la ruta correspondiente, dependiendo de la interfaz proporcionada.
3. Selecciona un archivo XML de configuración para cargar.

4. Haz clic en el botón "Cargar" o la acción equivalente.
5. El archivo XML de configuración se procesará y se actualizará el archivo "DB_CONFIGS_DICT.xml" en el servidor.

- **Consulta de Hashtags**

1. Abre la aplicación web en tu navegador.
2. Accede a la sección "Consultar Hashtags" o la ruta correspondiente, dependiendo de la interfaz proporcionada.
3. Ingresa las fechas de inicio y fin para el rango de consulta.
4. Haz clic en el botón "Consultar" o la acción equivalente.
5. Se mostrará una lista de hashtags y la cantidad de veces que se mencionaron en el rango de fechas especificado.

- **Consulta de Sentimientos**

1. Abre la aplicación web en tu navegador.
2. Accede a la sección "Consultar Sentimientos" o la ruta correspondiente, dependiendo de la interfaz proporcionada.
3. Ingresa las fechas de inicio y fin para el rango de consulta.
4. Haz clic en el botón "Consultar" o la acción equivalente.
5. Se mostrará una lista de fechas junto con la cantidad de sentimientos positivos, negativos y neutros en el rango de fechas especificado.

- **Consulta de Usuarios Mencionados**

1. Abre la aplicación web en tu navegador.
2. Accede a la sección "Consultar Usuarios Mencionados" o la ruta correspondiente, dependiendo de la interfaz proporcionada.
3. Ingresa las fechas de inicio y fin para el rango de consulta.
4. Haz clic en el botón "Consultar" o la acción equivalente.
5. Se mostrará una lista de fechas junto con la lista de usuarios mencionados en el rango de fechas especificado.

- **Reinicio de Datos**

1. Abre la aplicación web en tu navegador.
2. Accede a la sección "Resetear Datos" o la ruta correspondiente, dependiendo de la interfaz proporcionada.
3. Confirma la acción de reinicio de datos.
4. Los archivos de resumen y configuración se eliminarán, y las variables relacionadas con los mensajes y los sentimientos se reiniciarán.

Conclusiones

Ambos programas, el backend en Flask y el frontend en Django, son componentes de un servidor web que ofrecen APIs para procesar datos y comunicarse con un frontend. Flask proporciona rutas para cargar y consultar datos en archivos XML. Django, por otro lado, utiliza vistas y modelos para manejar solicitudes web y realizar operaciones en la base de datos. Ambos frameworks son adecuados para el desarrollo web, con Flask siendo más ligero y Django más completo y robusto.

Referencias bibliográficas

Welcome to. (2023, 15 febrero). Python.org. https://www.python.org/

Documentation for Visual Studio Code. (2021, 3 noviembre). https://code.visualstudio.com/docs

Hello World. (s. f.). GitHub Docs. https://docs.github.com/en/get-started/quickstart/hello-world

J. (2022, 16 enero). *Funciones en Python: Definición de función y para qué se utilizan*. J2LOGO. https://j2logo.com/python/tutorial/funciones-en-python/

Bustamante, S. J. (2021, 21 febrero). *Guía de funciones de Python con ejemplos*. freeCodeCamp.org. https://www.freecodecamp.org/espanol/news/guia-de-funciones-de-python-con-ejemplos/

Programación orientada a objetos. (s. f.). https://www.ibm.com/docs/es/spss-modeler/saas?topic=language-object-oriented-programming

J. (2022b, enero 16). *Programación orientada a objetos (POO) en Python*. J2LOGO. https://j2logo.com/python/tutorial/programacion-orientada-a-objetos/

Anexos

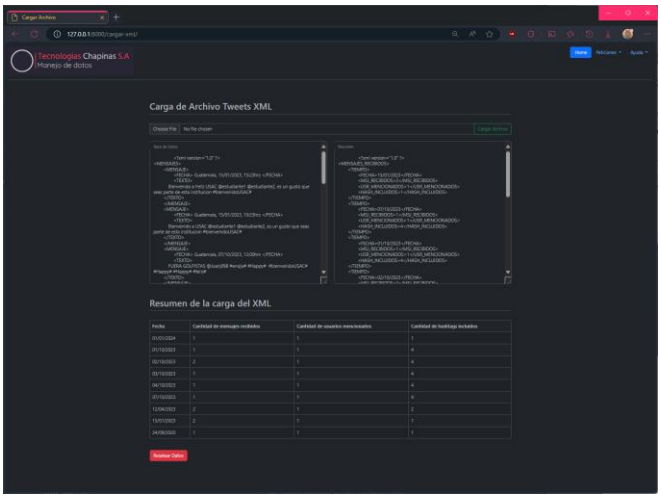

Figura 1. Carga de mensajes

Fuente: elaboración propia

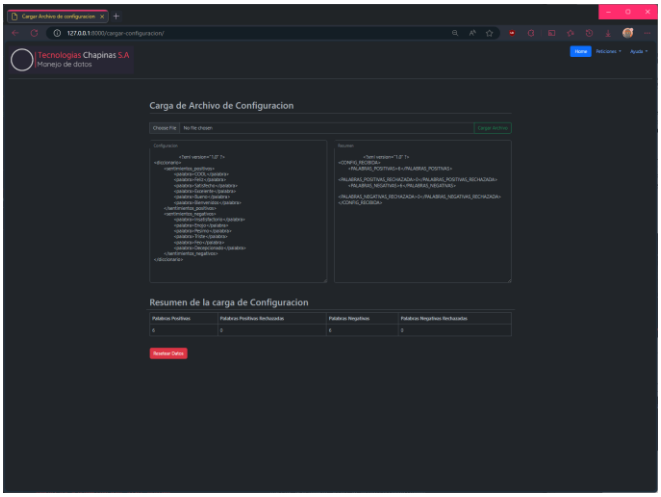

Figura 2. Carga de configuración

Fuente: elaboración propia

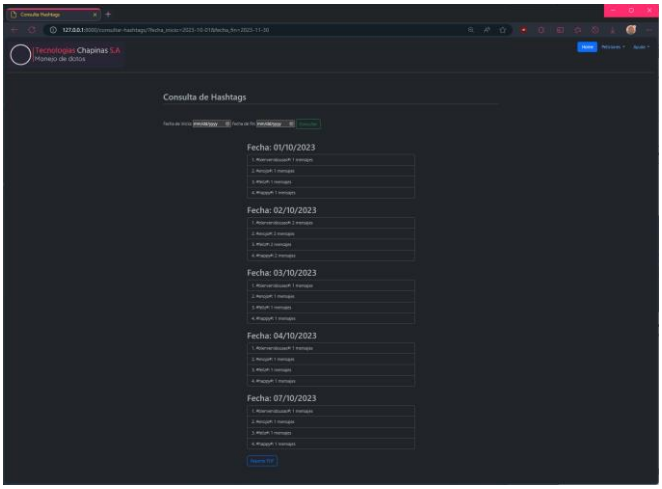

Figura 3. Consulta de hashtags

Fuente: elaboración propia

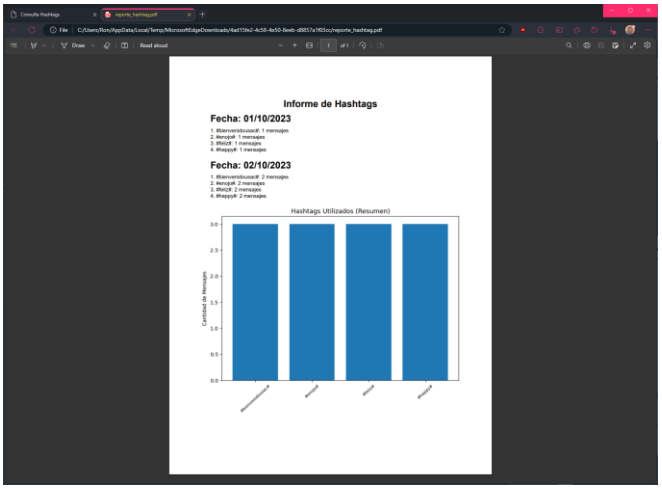

Figura 5. Informe con grafica

Fuente: elaboración propia

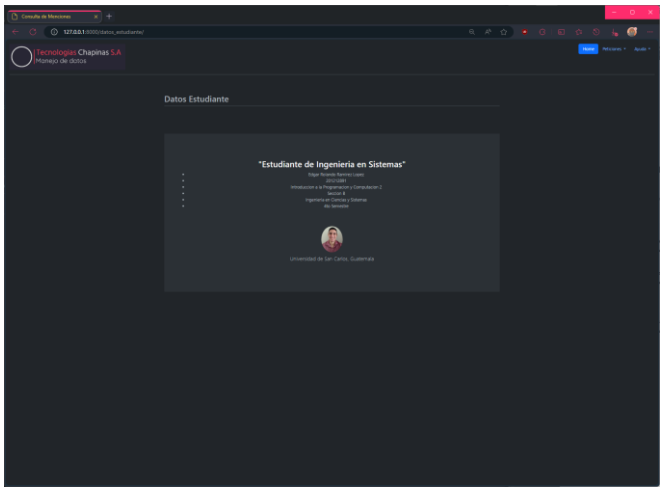

Figura 4. Datos estudiante

Fuente: elaboración propia

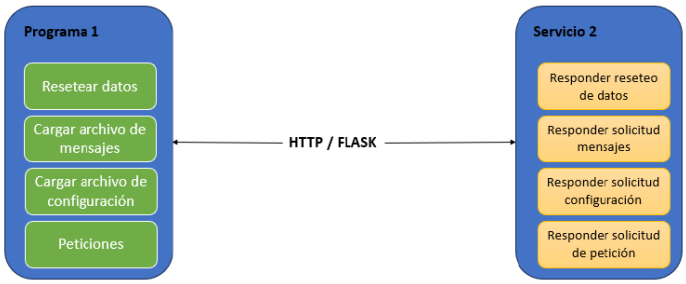

Figura 6. Arquitectura del programa

Fuente: enunciado proyecto

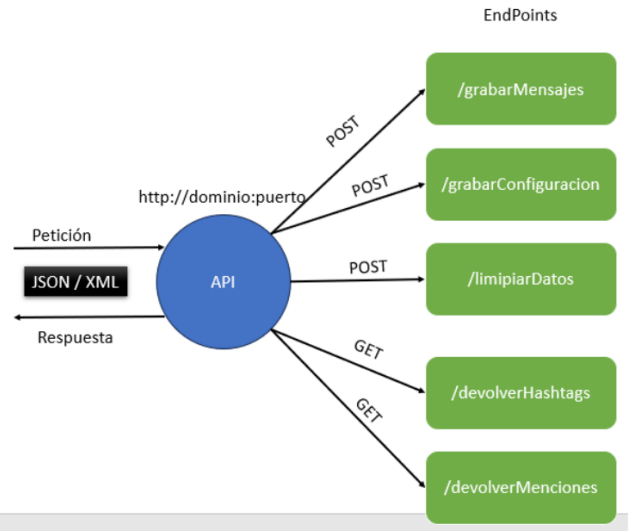

Figura 7. Estructura de la API

Fuente: enunciado proyecto