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 2024

PROYECTO 3 – Flas y Django

201701010 – Bryant Herrera Rubio

Resumen

Este proyecto desarrolla una herramienta para analizar y clasificar el sentimiento en mensajes de redes sociales relacionados con empresas y servicios específicos. La solución incluye una API construida en Flask que recibe y procesa mensajes formateados en XML. Utiliza programación orientada a objetos en Python, junto con bases de datos y diccionarios de sentimientos, para determinar si un mensaje es positivo, negativo o neutro. La información se organiza en archivos XML de salida, permitiendo emitir reportes detallados por fecha, empresa, y servicio, con resúmenes de clasificación. Además, el sistema se complementa con un frontend en Django, proporcionando una interfaz de usuario para cargar archivos, visualizar clasificaciones, generar gráficos y consultar datos detallados. Este frontend también permite pruebas en tiempo real y la creación de reportes PDF. La estructura modular y el uso de estructuras de datos abstractas garantizan que el sistema sea escalable y fácil de mantener.

Palabras clave

- HTML
- Framework
- API REST
- POSTMAN
- Django

Abstract

The presented project develops a simulation system for an automated assembly line, utilizing linked data structures to manage machines, products, and actions over time. The machine, with multiple production lines and components, follows detailed instructions to assemble products in various stages. Each assembly line moves and performs assembly actions in sequences defined by specific instructions, recording the total time of each process.

The code is divided into modules representing key elements such as products, positions, actions, and seconds, all managed through nodes in linked lists. A structure is used to store and update assembly positions, along with a simulation of the assembly process, calculating the total time required.

Keywords

- *HTML*
- *Framework*
- *API REST*
- *POSTMAN*
- *Django*

Introducción

Este proyecto presenta el desarrollo de un sistema para analizar y clasificar sentimientos en mensajes de redes sociales asociados a empresas y sus servicios. Con el objetivo de proporcionar a Tecnologías Chapinas, S.A. una herramienta integral de monitoreo, se ha implementado una API en Python utilizando Flask, la cual recibe mensajes formateados en XML, extrae y procesa los datos para determinar el sentimiento dominante (positivo, negativo o neutro). A través de un diccionario de palabras asociadas a sentimientos, empresas y servicios, el sistema es capaz de clasificar y almacenar los datos en una base XML de salida, la cual respalda la generación de reportes detallados por fecha y categoría.

El proyecto incluye una interfaz en Django que permite a los usuarios cargar archivos, realizar consultas, visualizar estadísticas de clasificación de forma gráfica y generar reportes en PDF. Este enfoque ofrece una estructura modular y escalable, facilitando la gestión de datos y permitiendo a la empresa un análisis profundo y automatizado de la percepción pública en redes sociales.

Desarrollo del tema

Desarrollo del Sistema de Análisis de Sentimientos
El desarrollo del sistema de análisis de sentimientos se enfocó en crear una API REST robusta que se integra con una interfaz de usuario en Django, permitiendo a los usuarios interactuar con el backend sin necesidad de comunicación directa entre backend y frontend a través de JavaScript. A continuación, se describe en detalle el desarrollo de cada componente de este sistema.

API REST con Flask y sus Endpoints

Para el desarrollo de la API REST se utilizó Flask, un framework en Python que permite construir aplicaciones web de manera ágil. El API se diseñó para recibir solicitudes HTTP que permiten procesar y clasificar mensajes en redes sociales. La API ofrece varios endpoints que cumplen funciones específicas en el análisis de sentimientos y el almacenamiento de datos en formato XML. Cada endpoint fue diseñado de acuerdo con los requerimientos específicos del proyecto, asegurando la correcta separación de responsabilidades y el fácil acceso a los datos.

Endpoints implementados:

1. **/cargar:** Este endpoint recibe archivos XML que contienen un diccionario de sentimientos y una lista de mensajes para clasificar. Primero, verifica que el archivo cumpla con el formato requerido y luego extrae los datos, almacenando las palabras positivas y negativas, empresas, y servicios mencionados en el diccionario. Cada mensaje en el XML se analiza para obtener información de ubicación, usuario, y red social, y se almacena en listas para su posterior procesamiento. Esto facilita la creación de un análisis de sentimiento que se basa en los términos encontrados en cada mensaje.
2. **/analizar:** Este endpoint realiza el análisis de los mensajes almacenados, clasificándolos en positivos, negativos o neutros según las palabras encontradas en el diccionario de sentimientos cargado. Para cada mensaje, se realiza un conteo de palabras positivas y negativas, y se determina el sentimiento con base en estos valores. Además, el sistema también identifica menciones de empresas y servicios, almacenando esta información en una estructura de datos para reportes detallados.
3. **/generar_respuesta:** Este endpoint genera un archivo XML de salida que contiene un

resumen del análisis de sentimientos por empresa, servicio y fecha. Para cada fecha, empresa y servicio, se muestra la cantidad de mensajes totales y clasificados por sentimiento (positivo, negativo, neutro). Este archivo de salida es fundamental para el funcionamiento del frontend, ya que contiene los datos que se mostrarán en la interfaz.

4. **/resumen_fecha y /resumen_rango_fechas:** Estos endpoints permiten realizar consultas de los datos almacenados, ya sea para una fecha específica o para un rango de fechas. Los datos devueltos incluyen un conteo de mensajes por empresa y por sentimiento. Esto permite visualizar cómo cambia la percepción de los usuarios a lo largo del tiempo y proporciona una visión detallada del sentimiento por fecha.
5. **/mensaje_prueba:** Este endpoint permite enviar un mensaje individual en formato XML para ser clasificado sin almacenarlo en la base de datos. Este endpoint ayuda a evaluar rápidamente si un mensaje tiene un sentimiento positivo, negativo o neutro sin necesidad de realizar un análisis completo. Es útil para pruebas rápidas y verificación de funcionamiento.
6. **/limpiar:** Este endpoint elimina todos los datos almacenados, dejando la base de datos en estado inicial. Esto permite realizar pruebas continuas sin acumulación de datos innecesarios, facilitando la validación y depuración del sistema.

Arquitectura Orientada a Objetos

Para garantizar una arquitectura modular y escalable, se utilizó el paradigma de programación orientada a objetos en Python. Las clases principales desarrolladas fueron:

Mensaje: Clase que representa un mensaje y contiene propiedades como la fecha, el usuario, la red social y el contenido. También implementa métodos para determinar el sentimiento del mensaje y el conteo de palabras positivas y negativas.

DiccionarioSentimientos: Clase que administra el diccionario de sentimientos, con métodos para agregar palabras positivas y negativas y buscar coincidencias en un mensaje.

Empresa y Servicio: Clases que representan las empresas y los servicios mencionados en los mensajes. Estas clases se encargan de contar las menciones y clasificar el sentimiento asociado a cada empresa o servicio.

AnalizadorSentimientos: Clase principal que coordina la interacción entre los mensajes, el diccionario de sentimientos y las empresas. Este analizador recibe los mensajes, clasifica su sentimiento y organiza los datos en la estructura de salida XML.

Estas clases permiten que el sistema sea fácilmente extensible y que cada módulo sea independiente, lo cual facilita la depuración y el mantenimiento del código.

Desarrollo de la Interfaz en Django

La interfaz en Django se desarrolló con el objetivo de proporcionar al usuario una forma sencilla y amigable de interactuar con la API. Django sigue el patrón de arquitectura Modelo-Vista-Template (MVT), lo que facilita la separación de la lógica del negocio (en el backend) de la presentación de datos en el frontend. Las principales funcionalidades en la interfaz incluyen:

Carga de Archivos: En esta sección, el usuario puede cargar archivos XML con solicitudes de clasificación. Django procesa el archivo y lo envía al endpoint /cargar del API para su análisis y almacenamiento.

Consulta de Datos: Se muestra un recuadro de texto en el que los usuarios pueden visualizar el resumen de clasificación más reciente, obtenido del archivo de salida generado por el endpoint /generar_respuesta.
Resúmenes por Fecha y Rango de Fechas: Mediante esta opción, el usuario puede seleccionar una fecha o un rango de fechas para obtener un análisis detallado por empresa y sentimiento. Django muestra los datos en una representación gráfica, utilizando la biblioteca Chart.js.
Prueba de Mensaje: En esta funcionalidad, los usuarios pueden ingresar un mensaje en un formato específico para analizarlo individualmente sin almacenar el resultado. Esto permite una interacción en tiempo real y ayuda a verificar el funcionamiento del sistema.
Opciones de Ayuda: La sección de ayuda incluye información del estudiante y acceso a la documentación del sistema, mostrando un archivo PDF.

Uso de Plantillas y Renderizado en Django
En Django, se usaron plantillas HTML para estructurar la interfaz de usuario. Las plantillas permiten reutilizar código, lo que simplifica el mantenimiento. La plantilla base incluye el menú de navegación y el encabezado, y en cada sección se carga contenido dinámico según la acción del usuario. Se usó {% include %} para integrar componentes repetitivos, como la barra lateral, y {% static %} para acceder a archivos estáticos, como CSS y JavaScript.
Cada vista en Django se comunica con el backend (sin usar JavaScript) a través de formularios HTML que envían solicitudes a la API. Cuando el usuario selecciona una opción en el frontend, se envía un formulario con la información de la solicitud, y Django procesa los datos y los envía al endpoint correspondiente.

Evitando la Comunicación Directa Backend-Frontend con JavaScript
Para evitar la comunicación directa entre el backend y frontend usando JavaScript, cada solicitud se realiza desde el frontend hacia Django, y luego Django comunica los datos a la API de Flask. Este flujo de datos es posible utilizando los métodos HTTP POST y GET de Django, donde cada formulario o acción del usuario envía una solicitud al servidor de Django, evitando el uso de JavaScript.
Cuando el usuario realiza una acción en el frontend, como seleccionar un archivo o elegir una fecha para un reporte, Django captura el evento y procesa la solicitud en la vista correspondiente, utilizando el framework de formularios de Django para manejar los datos de entrada. Por ejemplo, en el resumen por rango de fechas, el usuario selecciona las fechas en el frontend, y Django se encarga de enviar estos parámetros al backend a través de una solicitud HTTP, recibe los datos procesados en respuesta y los renderiza en la plantilla. Esto garantiza la funcionalidad del sistema sin necesidad de JavaScript para comunicación entre backend y frontend.
Generación de Gráficas en Django con Chart.js
Las gráficas en el frontend se generaron con Chart.js, una biblioteca de JavaScript para gráficos que es independiente y se carga directamente en las plantillas de Django. Chart.js toma los datos renderizados en la plantilla (sin JavaScript para comunicación con el backend) y los convierte en gráficos dinámicos y visualmente atractivos. De esta manera, el usuario puede ver las clasificaciones de sentimientos en gráficos que ilustran el análisis de los mensajes de redes sociales. Esta visualización se logra al pasar los datos al frontend como contexto de la plantilla, que luego es interpretado por Chart.js.

Conclusión

Este proyecto implementa un sistema completo para el análisis de sentimientos de mensajes en redes sociales, diseñado para funcionar de manera independiente y sin dependencias directas de JavaScript entre el frontend y backend. La arquitectura orientada a objetos y la implementación en Flask y Django garantizan la modularidad y escalabilidad del sistema. Gracias a este enfoque, Tecnologías Chapinas, S.A. puede monitorear la percepción pública sobre sus servicios de forma detallada y precisa, aprovechando un sistema que organiza y almacena los datos en un formato estandarizado y consultable en tiempo real.

Referencias bibliográficas

1. Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.

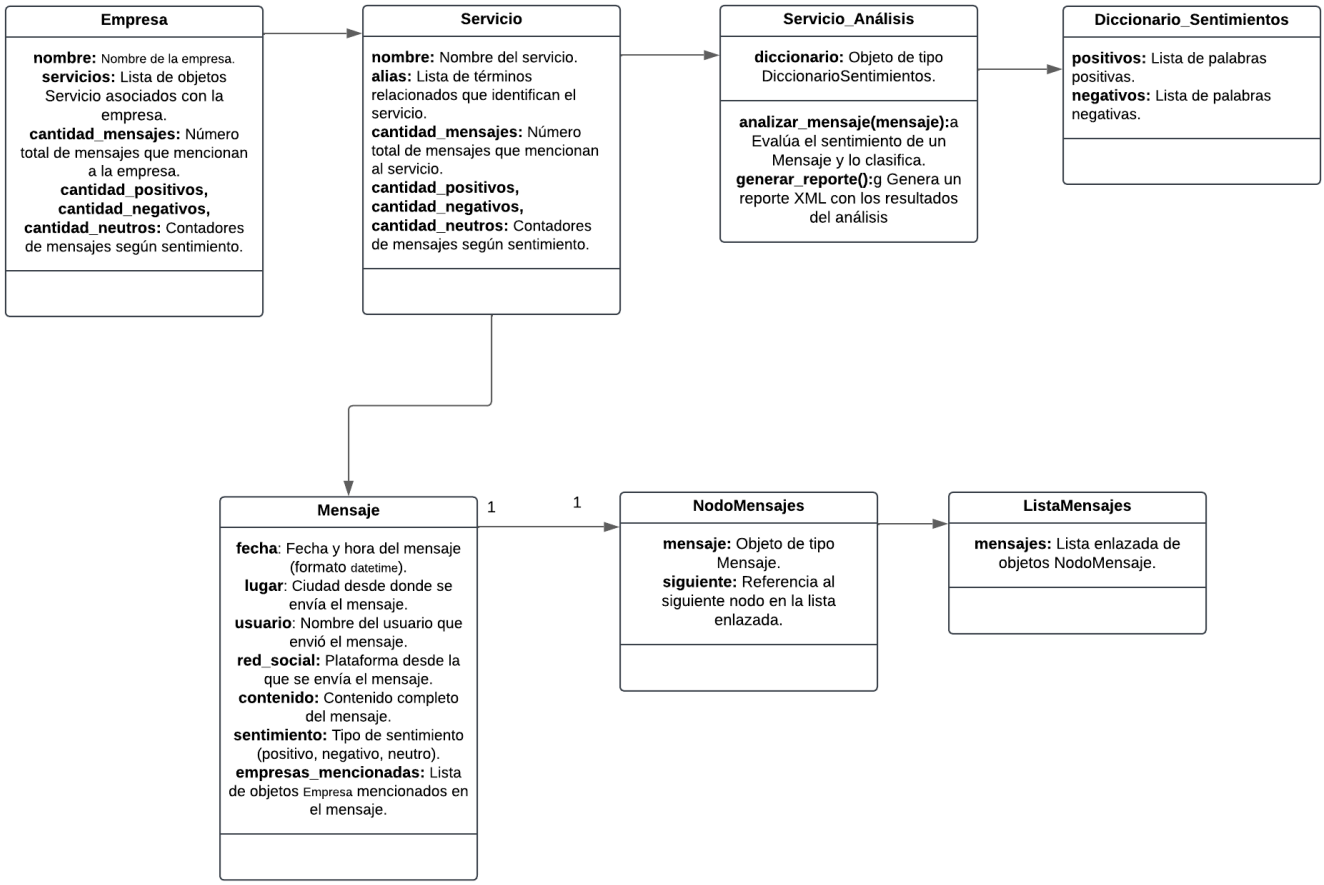
2. Goodrich, M. T., & Tamassia, R. (2014). Data Structures and Algorithms in Python. Wiley.

3. Cheng, Y., & Liu, Z. (2017). "Design and Implementation of a Simulation System Based on Python". International Journal of Software Engineering & Applications (IJSEA), 8(1), 1-12..

4. Manual de Django

Anexos

Diagrama de clases:



Diagramas de actividades: