Proyecto 3

202105658 – BRAYAN ALEXANDER GUZMAN MARGOS

Resumen

El proyecto consta de dos componentes principales: un frontend en Django y un backend en Flask. El frontend actúa como un simulador para probar la API del backend, permitiendo cargar archivos XML, consultar datos almacenados y generar resúmenes de clasificación por fecha y rango de fechas. También incluye opciones para realizar un reporte en PDF y probar mensajes específicos, los cuales no se almacenan en la base de datos. Los usuarios pueden visualizar información de ayuda sobre el programa y enviar solicitudes de clasificación a la API. El backend, por su parte, se encarga de procesar los datos recibidos, almacenarlos en archivos XML y devolver la información necesaria al frontend. Utiliza el protocolo HTTP para la comunicación y permite el acceso a sus servicios a través de clientes externos como Postman. Ambos componentes deben interactuar eficientemente para garantizar el correcto funcionamiento del sistema de análisis y clasificación de mensajes, facilitando así el análisis de sentimientos.

Palabras clave

1. **API:** Interfaz de Programación de Aplicaciones que permite la comunicación entre diferentes software, facilitando el intercambio de datos y servicios.

2. **Clasificación:** Proceso de categorizar información o datos en grupos específicos según características o criterios establecidos.

3. **Django/Flask:** Frameworks de desarrollo web en Python; Django se enfoca en la rapidez y facilidad de uso, mientras que Flask es más ligero y flexible, ideal para aplicaciones pequeñas y microservicios.

Abstract

The project consists of two main components: a Django frontend and a Flask backend. The frontend acts as a simulator to test the backend API, allowing XML files to be loaded, stored data to be queried, and sorted by date and date range to be generated. It also includes options to create a PDF report and test specific messages, which are not stored in the database. Users can view help information about the program and send sorting requests to the API. The backend, on the other hand, is responsible for processing the received data, storing it in XML files, and returning the necessary information to the frontend. It uses the HTTP protocol for communication and allows access to its services through external clients such as Postman. Both components must interact efficiently to ensure the correct functioning of the message analysis and classification system, thus facilitating sentiment analysis.

Keywords

1. API: Application Programming Interface that allows communication between different software, facilitating the exchange of data and services.

2**. Classification:** Process of categorizing information or data into specific groups according to established characteristics or criteria.

3. Django/Flask: Web development frameworks in Python; Django focuses on speed and ease of use, while Flask is lighter and more flexible, ideal for small applications and microservices.

Introducción

La transformación digital ha llevado a las empresas a buscar soluciones innovadoras que les permitan adaptarse a un entorno en constante cambio. En este contexto, el Proyecto 3 de Tecnologías Chapinas, S.A. tiene como objetivo desarrollar una herramienta integral que permita analizar el contenido de las redes sociales y evaluar el sentimiento de los usuarios respecto a empresas y servicios. Utilizando el Protocolo HTTP y el paradigma de programación orientada a objetos (POO), este proyecto se enfoca en la implementación de una API que facilita la interacción con los datos. Mediante el uso de archivos XML como insumos, el sistema es capaz de procesar mensajes estructurados y clasificarlos en categorías de sentimiento, ofreciendo así a la empresa una visión clara de la percepción pública. Además, se busca almacenar información de manera persistente y utilizar expresiones regulares para extraer contenido relevante, garantizando la eficacia y eficiencia del análisis. Este ensayo examina los componentes técnicos del proyecto, la metodología de desarrollo empleada y los beneficios potenciales para Tecnologías Chapinas, S.A.

Desarrollo del tema

La construcción de aplicaciones web modernas a menudo requiere una cuidadosa separación de responsabilidades entre el frontend y el backend. Este enfoque no solo facilita el mantenimiento y la escalabilidad del código, sino que también permite aprovechar las fortalezas de diferentes frameworks para cumplir con requisitos específicos del proyecto. En este contexto, el presente código implementa un sistema que combina Flask para el backend y Django para el frontend, orientado al procesamiento de mensajes en formato XML y al análisis de sentimientos.

1. Backend con Flask

1.1. Estructura de la API

Flask se configura como un microframework para gestionar la API que recibe y procesa los mensajes. Se define una serie de rutas que responden a las solicitudes HTTP, permitiendo la comunicación con el frontend. La arquitectura del backend se fundamenta en dos elementos principales:

1. **Lectura del Diccionario de Sentimientos**:
   - La ruta `/config/leerDiccionario` es responsable de recibir un archivo XML que contiene listas de palabras positivas y negativas, así como la estructura de empresas y sus servicios.
   - Utilizando la biblioteca `xml.etree.ElementTree`, se parsea el XML y se extraen las palabras clave. Las palabras se almacenan en listas, y se crea una instancia de las clases `Empresa` y `Servicio` para estructurar la información relacionada con las empresas que se analizarán.
   - Este enfoque orientado a objetos permite encapsular la lógica de negocio relacionada con las empresas y sus servicios, facilitando la extensión y el mantenimiento del código en el futuro.
2. **Análisis de Mensajes**:
   - La ruta `/analisis/analizarMU` recibe mensajes en formato XML que

incluyen información contextual como la fecha, el usuario y la red social de origen.
- Se implementa un análisis de texto utilizando expresiones regulares para extraer datos relevantes del mensaje. Las palabras se comparan con las listas de palabras positivas y negativas previamente almacenadas, y se generan métricas de análisis de sentimientos, como el conteo de palabras positivas y negativas, así como el sentimiento general del mensaje (positivo, negativo o neutro).
- La respuesta se genera en formato XML, proporcionando una estructura clara y organizada que se puede consumir fácilmente en el frontend.

Estructura de la Interfaz de Usuario

Django se utiliza para construir la interfaz de usuario, proporcionando un marco robusto para el desarrollo de aplicaciones web. Se implementan diversas vistas y formularios que permiten a los usuarios interactuar con la API de Flask de manera intuitiva. Los componentes principales incluyen:

1. **Carga y Visualización de Archivos XML**:
   - La vista `visualizarXML` permite a los usuarios cargar archivos XML que contienen mensajes. Una vez cargados, se parsea el contenido y se genera una representación gráfica utilizando la biblioteca `graphviz`, que puede facilitar la comprensión del flujo de datos dentro del XML.
   - Este enfoque visual ayuda a los usuarios a verificar el contenido del archivo antes de enviarlo al backend para su procesamiento.
2. **Interacción con la API**:
   - La vista `subirXML` se encarga de enviar el contenido del XML a la API para su análisis. Tras recibir la respuesta, se muestra el resumen de los resultados en la interfaz.
   - Esto se complementa con opciones para resetear la base de datos a su estado inicial, lo que permite a los usuarios empezar de nuevo sin necesidad de reiniciar la aplicación.
3. **Secciones de Ayuda y Documentación**:
   - Se incluyen vistas para proporcionar información adicional sobre el uso de la aplicación y detalles de contacto del estudiante responsable del proyecto. Esto es fundamental para guiar a los usuarios a través de las funcionalidades de la aplicación y ayudarles a comprender cómo utilizarla eficazmente.

Conclusiones

La implementación presentada ilustra cómo la integración de Flask y Django puede resultar en un sistema eficaz y escalable para el análisis de sentimientos en mensajes XML. Este enfoque modular no solo mejora la claridad del código, sino que también facilita el desarrollo y el mantenimiento a largo plazo. A medida que la tecnología y los requisitos evolucionan, la capacidad de adaptar y extender cada componente se convierte en un activo valioso para cualquier proyecto de desarrollo de software. Este sistema no solo cumple con su propósito inicial, sino que también sienta las bases para futuras mejoras y ampliaciones, asegurando su relevancia en un panorama tecnológico en constante cambio.

Anexos

Diagrama de clases

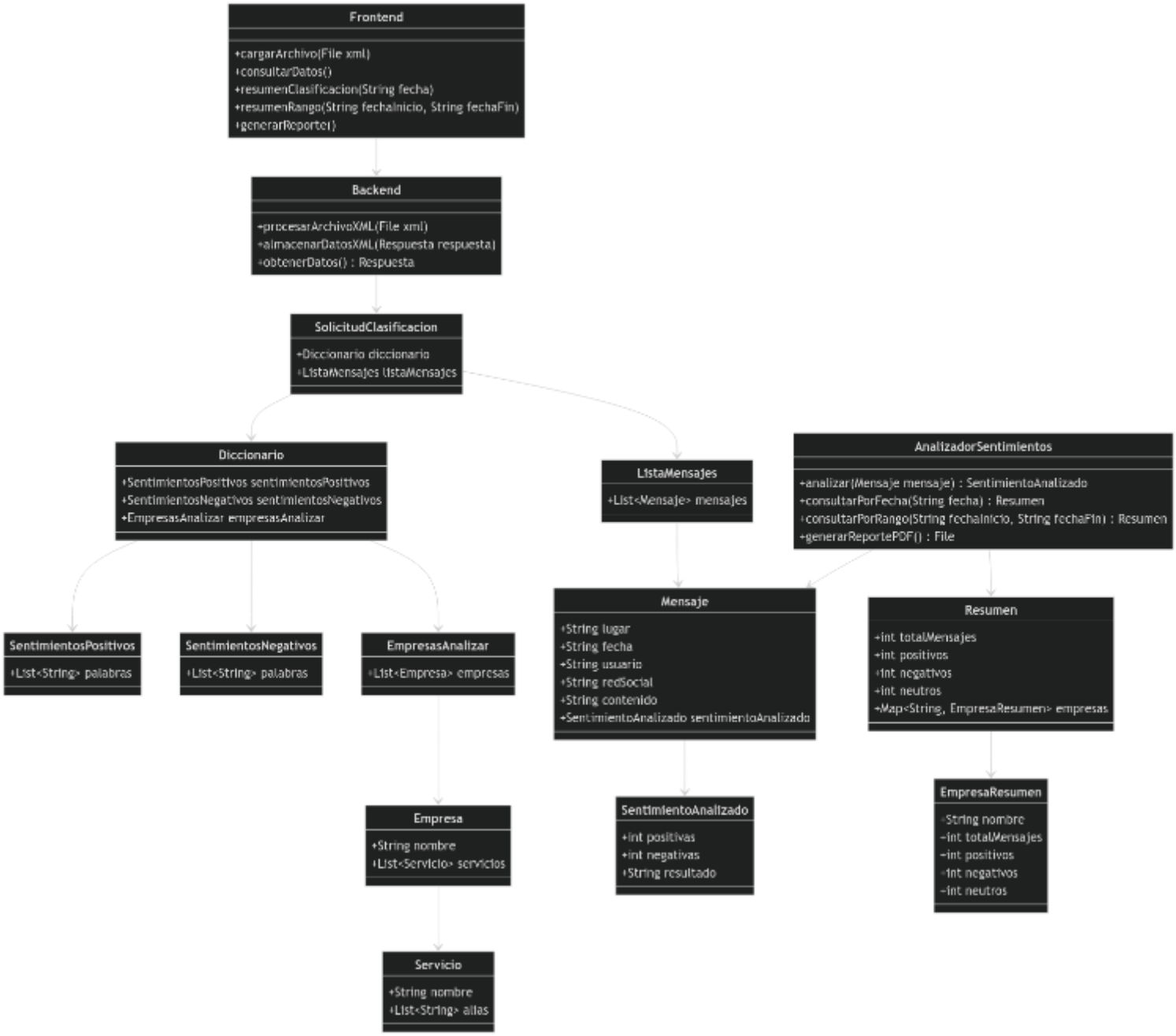
Frontend
+cargarArchivo(File xml)
+consultarDatos()
+resumenClasificacion(String fecha)
+resumenRango(String fechaInicio, String fechaFin)
+generarReporte()
Backend
+procesarArchivoXML(File xml)
+almacenarDatosXML(Respuesta respuesta)
+obtenerDatos() : Respuesta
SolicitudClasificacion
+Diccionario diccionario
+ListaMensajes listaMensajes
Diccionario
+SentimientosPositivos sentimientosPositivos
+SentimientosNegativos sentimientosNegativos
+EmpresasAnalizar empresasAnalizar
ListaMensajes
+List<Mensaje> mensajes
AnalizadorSentimientos
+analizar(Mensaje mensaje) : SentimientoAnalizado
+consultarPorFecha(String fecha) : Resumen
+consultarPorRango(String fechaInicio, String fechaFin) : Resumen
+generarReportePDF() : File
SentimientosPositivos
+List<String> palabras
SentimientosNegativos
+List<String> palabras
EmpresasAnalizar
+List<Empresa> empresas
Mensaje
+String lugar
+String fecha
+String usuario
+String redSocial
+String contenido
+SentimientoAnalizado sentimientoAnalizado
Resumen
+int totalMensajes
+int positivos
+int negativos
+int neutros
+Map<String, EmpresaResumen> empresas
Empresa
+String nombre
+List<Servicio> servicios
SentimientoAnalizado
+int positivas
+int negativas
+String resultado
EmpresaResumen
-String nombre
-int totalMensajes
-int positivos
-int negativos
-int neutros
Servicio
+String nombre
+List<String> alias

Diagrama de Actividades