**Universidad San Carlos de Guatemala**

**Facultad de Ingeniería**

**Escuela de Ciencias y Sistemas**

**Introducción a la Programación y Computación 2**

**Ing. Byron Rodolfo Zepeda Arevalo**

**Aux. Erwin Fernando Vásquez**

# ENSAYO-PROYECTO3

**Nombre:** Frander Oveldo Carreto Gómez

**Carné:** 201901371

**Fecha:** 31/10/23

# INTRODUCCION

Flask es un marco de trabajo web minimalista y altamente popular para el desarrollo de aplicaciones web en el lenguaje de programación Python. Diseñado por Armin Ronacher, Flask se ha ganado una gran reputación en la comunidad de desarrollo debido a su simplicidad y flexibilidad. A pesar de ser conocido como un "microframework", Flask proporciona las herramientas esenciales para construir aplicaciones web sólidas sin imponer una estructura rígida. Esta característica permite a los desarrolladores la libertad de elegir sus herramientas y bibliotecas, lo que hace que Flask sea una excelente opción para proyectos de todos los tamaños.

La simplicidad de Flask radica en su diseño modular y su enfoque en la elegancia del código. Con Flask, puedes definir rutas y vistas de manera intuitiva mediante decoradores, lo que facilita la creación de páginas web dinámicas y funcionales. Además, Flask utiliza el motor de plantillas Jinja2 para renderizar contenido web de manera flexible y eficiente. Este marco de trabajo también se integra bien con una variedad de extensiones y módulos que simplifican tareas comunes, como la gestión de bases de datos, la autenticación de usuarios y la creación de APIs RESTful.

Flask es una herramienta valiosa en el mundo del desarrollo web, adecuada tanto para principiantes como para desarrolladores experimentados. Su enfoque en la elegancia del código y su facilidad de uso lo convierten en una elección popular para aquellos que desean crear aplicaciones web de manera rápida y efectiva. En los siguientes párrafos, exploraremos con más detalle los componentes clave de Flask y proporcionaremos ejemplos prácticos para comprender su funcionamiento en profundidad.

**Estructura de una Aplicación Flask**

Una aplicación Flask sigue una estructura modular que incluye rutas, vistas, plantillas y modelos. Veamos cada uno de estos componentes y cómo se relacionan.

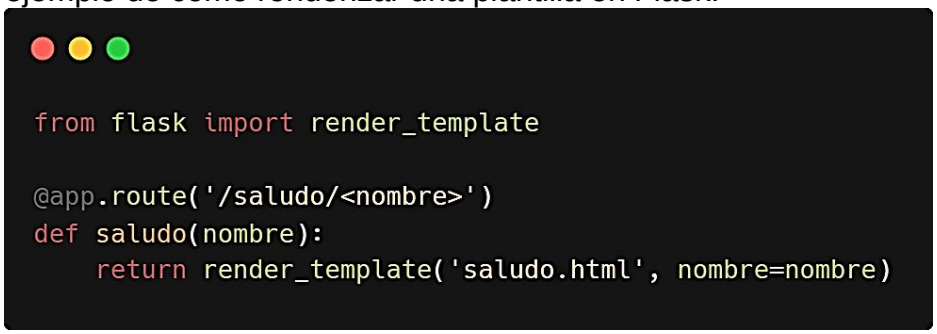
1. **Rutas (Routes)**: Las rutas definen cómo se manejarán las solicitudes entrantes. En Flask, las rutas se definen utilizando decoradores, como **@app.route('/')**. ¡A continuación, se muestra un ejemplo de una ruta simple que muestra “¡Hola, mundo!" en la página de inicio:

```python
from flask import Flask

app = Flask(__name)

@app.route('/')
def home():
    return '¡Hola, mundo!'
```

2. Vistas (Views): Las vistas son las funciones que se ejecutan cuando se accede a una ruta específica. En el ejemplo anterior, la función home() es la vista para la ruta '/' y devuelve el mensaje "¡Hola, mundo!".
3. Plantillas (Templates): Las plantillas permiten separar la lógica de presentación del código Python. Flask utiliza el motor de plantillas Jinja2 para renderizar HTML de manera dinámica. A continuación, se muestra un ejemplo de cómo renderizar una plantilla en Flask:



```python
from flask import render_template

@app.route('/saludo/<nombre>')
def saludo(nombre):
    return render_template('saludo.html', nombre=nombre)
```

4. Modelos (Models): Flask no impone un ORM (Mapeo Objeto-Relacional) específico, pero es común utilizar SQLAlchemy para interactuar con bases de datos en aplicaciones Flask. Los modelos representan la estructura de la base de datos y se utilizan para realizar operaciones de lectura y escritura en la misma.

5. Ejemplo de Aplicación Flask
6. A continuación, presentamos un ejemplo más completo de una aplicación Flask que incluye rutas, vistas, plantillas y una conexión a una base de datos SQLite.

```python
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///mi_basededatos.db'
db = SQLAlchemy(app)

class Usuario(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    nombre = db.Column(db.String(80), unique=True, nullable=False)

@app.route('/')
def inicio():
    usuarios = Usuario.query.all()
    return render_template('inicio.html', usuarios=usuarios)

if __name__ == '__main__':
    db.create_all()
    app.run()
```

En este ejemplo, definimos una ruta */* que muestra una lista de usuarios almacenados en una base de datos SQLite. La vista **inicio()** recupera los usuarios de la base de datos y los pasa a una plantilla para su renderización.

1. **Importaciones de Módulos**:
    - **import xml.etree.ElementTree as ET**: Importa la biblioteca ElementTree de Python, que se utiliza para analizar y manipular documentos XML.
    - **from flask import Flask, request, jsonify, make_response**: Importa las clases y funciones necesarias de Flask para crear una aplicación web.
    - **from flask_cors import CORS**: Importa el módulo CORS de Flask para habilitar el intercambio de recursos entre dominios en tu aplicación.

- **from utileria.mntConfiguracionXML import SentimientosDictionary** y **from utileria.mntMensajeXML import MensajesXMLManager**: Importa clases específicas de tus propios módulos, lo que indica que estás utilizando utilidades personalizadas para el manejo de datos XML y configuraciones.

2. **Creación de la Aplicación Flask**:
    - **app = Flask(__name__)**: Crea una instancia de la clase Flask, que representa tu aplicación web.
    - **CORS(app)**: Habilita el intercambio de recursos entre dominios (CORS) en tu aplicación para permitir el acceso desde diferentes ubicaciones.
3. **Almacenamiento de Mensajes en Formato XML**:
    - **messages_data = []**: Define una lista para almacenar los mensajes procesados en formato XML, proporcionando un lugar para conservar los datos de los mensajes.
4. **Ruta '/grabarMensaje'**:
    - **@app.route('/grabarMensaje', methods=['POST'])**: Define una ruta llamada **/grabarMensaje** que acepta solicitudes POST. Esta ruta se utilizará para grabar mensajes.
    - **request.files['file']**: Accede al archivo enviado en la solicitud POST.
    - **ET.fromstring(file_contents)**: Analiza el contenido del archivo XML en un objeto de árbol XML.
    - **manager.agregar_mensaje(nueva_fecha, nuevo_texto)**: Agrega mensajes al objeto **manager** para su posterior procesamiento.
5. **Ruta '/grabarConfiguracion'**:
    - **@app.route('/grabarConfiguracion', methods=['POST'])**: Define una ruta llamada **/grabarConfiguracion** que acepta solicitudes POST. Esta ruta se utilizará para grabar configuraciones del servidor.
    - **file_contents = file.read().decode('utf-8')**: Lee y decodifica el contenido del archivo XML enviado en la solicitud.
    - **SentimientosDictionary()**: Crea una instancia de la clase **SentimientosDictionary** para manejar configuraciones de sentimientos.
    - **diccionario.cargar_desde_archivo("sentimientos.xml")**: Carga configuraciones de sentimientos desde un archivo XML existente o lo crea si no existe.

- Procesa el contenido XML y agrega palabras al diccionario de sentimientos.

6. **Ruta '/limpiarDatos'**:
    - **@app.route('/limpiarDatos', methods=['POST'])**: Define una ruta llamada **/limpiarDatos** que acepta solicitudes POST. Esta ruta se utiliza para limpiar los datos almacenados en **messages_data**.
7. **Ruta '/devolverMenciones'**:
    - **@app.route('/devolverMenciones', methods=['GET'])**: Define una ruta llamada **/devolverMenciones** que acepta solicitudes GET. Esta ruta se utiliza para obtener menciones de palabras con sentimientos positivos y negativos.
    - Carga configuraciones de sentimientos desde un archivo XML.
    - Recopila palabras únicas y cuenta repeticiones en función de los sentimientos.
8. **Ruta '/devolverHashtags'**:
    - **@app.route('/devolverHashtags', methods=['GET'])**: Define una ruta llamada **/devolverHashtags** que acepta solicitudes GET. Esta ruta se utiliza para obtener hashtags en función de las fechas proporcionadas.
    - Analiza mensajes XML, filtra mensajes por fechas y recopila hashtags junto con su frecuencia.
9. **Funciones Auxiliares**:
    - **generate_xml_response(hashtags_by_date)**: Genera una respuesta XML a partir de datos de hashtags por fecha.
    - **get_hashtags(text)**: Extrae hashtags de un texto y los devuelve en forma de lista.
    - **get_fecha(text)**: Busca una fecha en un texto y la devuelve como un objeto **datetime.date**.

```
import xml.etree.ElementTree as ET

from flask import Flask, request, jsonify, make_response

from flask_cors import CORS

from utileria.mntConfiguracionXML import SentimientosDictionary
from utileria.mntMensajeXML import MensajesXMLManager

from collections import defaultdict
import re
from datetime import datetime
```

```python
#POST
#Grabar mensajes
@app.route('/grabarMensaje', methods=['POST'])

#Grabar configuracion
@app.route('/grabarConfiguracion', methods=['POST'])

#Limpiar datos
@app.route('/limpiarDatos', methods=['POST'])
```

```python
#GET
#Devolver menciones
@app.route('/devolverMenciones', methods=['GET'])

#Devolver Hastags
@app.route('/devolverHashtags', methods=['GET'])
```