PROYECTO FINAL PROGRAMACIÓN 3

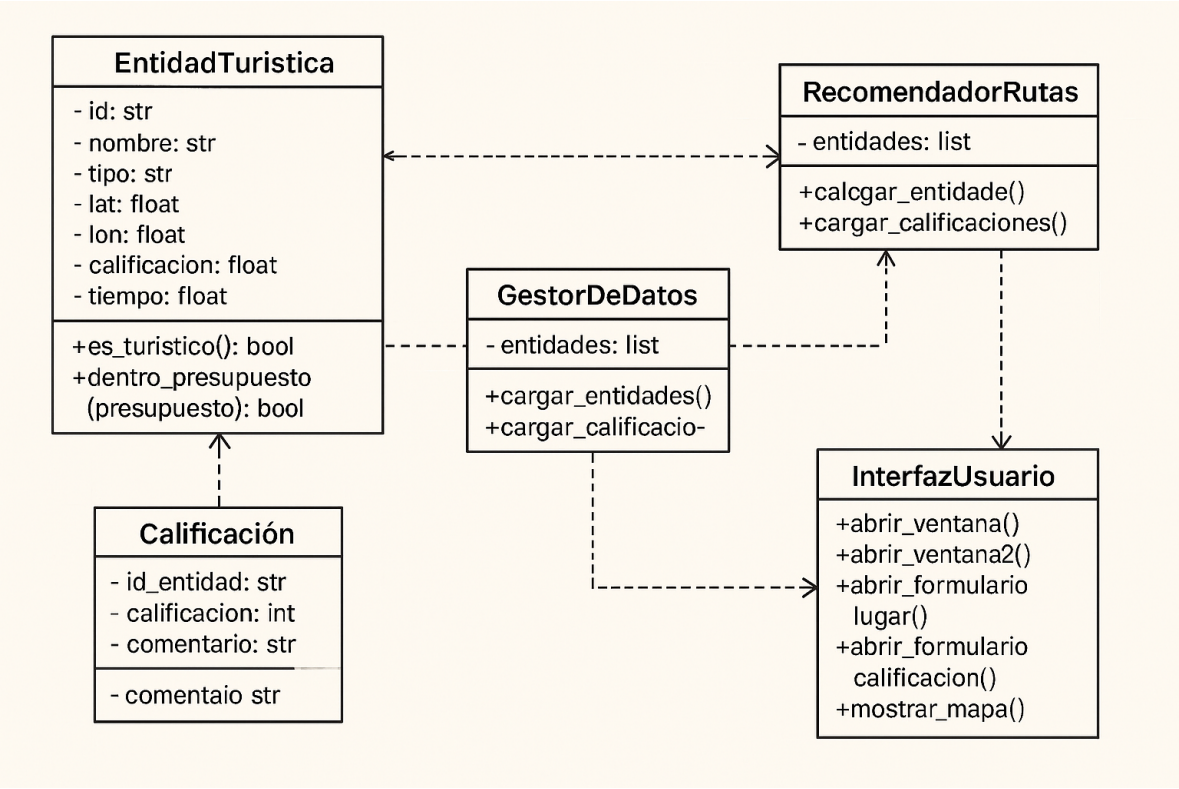
CARLOS RENÉ FLORES CUTZAL - 9490-21-1393
ERI EDUARDO GARCIA CHINCHILLA - 9490-19-6782
JOSUE DANIEL ECHEVERRIA JUAREZ - 9490-20-121

Facultad de Ingeniería en Sistemas de la Información, Universidad Mariano Gálvez de Guatemala

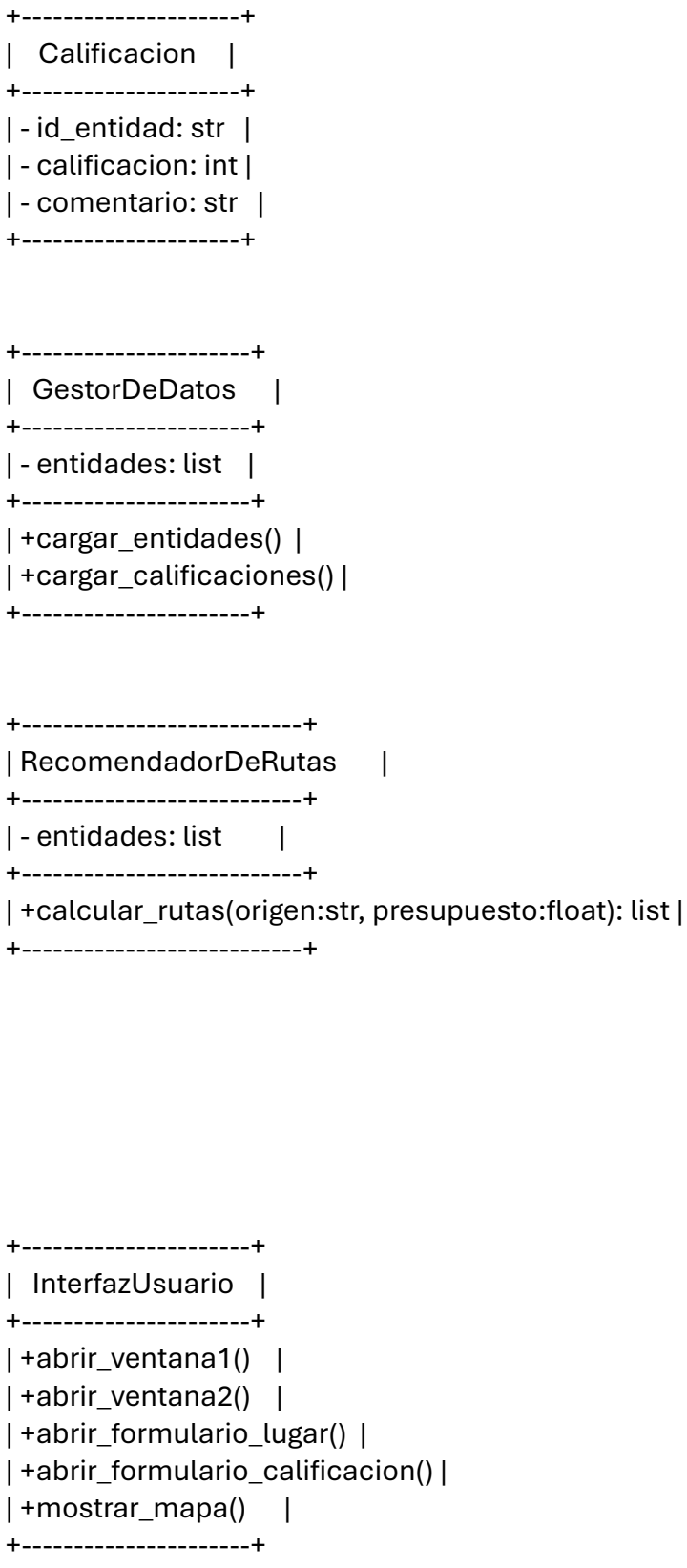
MANUAL TÉCNICO – PROYECTO FINAL PROGRAMACIÓN 3

31 de mayo de 2025

1. Diagrama de clases:




```
+---------------------+
| EntidadTuristica    |
+---------------------+
| - id: str           |
| - nombre: str       |
| - tipo: str         |
| - lat: float        |
| - lon: float        |
| - precio: float     |
| - calificacion: float |
| - tiempo: float     |
+---------------------+
| +es_turistico(): bool |
| +dentro_presupuesto(presupuesto): bool |
+---------------------+
```


Calificacion
- id_entidad: str
- calificacion: int
- comentario: str
GestorDeDatos
- entidades: list
+cargar_entidades()
+cargar_calificaciones()
RecomendadorDeRutas
- entidades: list
+calcular_rutas(origen:str, presupuesto:float): list
InterfazUsuario
+abrir_ventana1()
+abrir_ventana2()
+abrir_formulario_lugar()
+abrir_formulario_calificacion()
+mostrar_mapa()

2. Descripción de la estructura de datos

La aplicación organiza los datos de forma estructurada para facilitar la carga, almacenamiento y acceso eficiente durante el proceso de recomendación. A continuación se detallan los aspectos principales:

Estructuras utilizadas:

Se emplean principalmente diccionarios de Python y listas anidadas para almacenar y acceder a las entidades. Además, se implementan estructuras más avanzadas como:

- Árbol B para almacenar y organizar los lugares turísticos y de hospedaje.
- Grafo ponderado para modelar las rutas posibles entre lugares, considerando distancia, tiempo y costo.

Carga de datos desde archivos CSV:

El programa permite al usuario importar dos archivos CSV:

- entidades.csv: contiene información de cada lugar (ID, nombre, tipo, latitud, longitud, precio, calificación, tiempo estimado de estadía).
- calificaciones.csv: contiene valoraciones de usuarios (ID del lugar, calificación numérica, comentario).

Los datos se leen usando csv.reader y se almacenan dinámicamente en memoria, estructurados por ID.

Acceso en memoria:

Una vez cargados, los datos se guardan en:

- Un diccionario de entidades, donde la clave es el ID y el valor es una instancia o registro con todos sus atributos.
- Una estructura paralela de calificaciones, que se cruza con las entidades para calcular promedios y enriquecer la recomendación.

Relación entre entidades:

- Cada lugar cargado desde el CSV es una EntidadTurística, que puede ser de tipo Hospedaje o Turístico.
- A cada Entidad Turística se le puede asociar una o varias Calificaciones individuales cargadas aparte.
- Las relaciones permiten que, al calcular rutas, se prioricen lugares con mejor calificación promedio, menor costo y adecuada duración.

3. Explicación del algoritmo de recomendación

La aplicación es su algoritmo de recomendación, diseñado para ayudar al usuario a tomar decisiones inteligentes sobre qué lugares visitar en función de su presupuesto, tiempo disponible y la calidad de las experiencias que ofrecen los destinos.

¿Qué toma en cuenta el sistema?

El algoritmo considera varios factores clave para generar sugerencias personalizadas:

- Presupuesto diario: el usuario define cuánto está dispuesto a gastar en un día de recorridos.
- Tiempo máximo disponible: se establece un límite de 8 horas por día, que incluye tanto el tiempo de traslado como el de estadía.
- Calificación promedio: los lugares con mejor puntuación por parte de otros usuarios tienen prioridad.
- Precio del lugar y tipo de lugar: solo se consideran lugares turísticos, dejando fuera los hospedajes en esta etapa.

¿Cómo funciona el proceso?

Una vez el usuario ingresa el punto de origen y su presupuesto, el sistema analiza todos los lugares turísticos disponibles.

Filtra aquellos que están dentro del presupuesto y cuyo tiempo de visita no excede el límite diario.

De los lugares válidos, selecciona los mejor calificados.

Finalmente, se muestran hasta cinco recomendaciones únicas y personalizadas, que se ajustan a las condiciones ingresadas.

¿Qué obtiene el usuario?

El resultado es un listado claro y ordenado con los mejores lugares para visitar ese día, con información como:

- Nombre del lugar
- Costo estimado
- Tiempo de estadía
- Calificación promedio

Además, todos los lugares recomendados pueden visualizarse directamente en un mapa interactivo, permitiendo al usuario tener una idea clara de su ubicación y organizar mejor su recorrido.

4. Estructura de Código

```
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import folium
import webbrowser
from graphviz import Digraph

# ======================== Árbol B ============================
class BTreeNode:
    def __init__(self, t, leaf=False):
        self.t = t
        self.leaf = leaf
        self.keys = []
        self.children = []

class BTree:
    def __init__(self, t):
        self.root = BTreeNode(t, True)
        self.t = t

    def insert(self, k):
```

```
        if len(self.root.keys) == (2 * self.t) - 1:
            new_root = BTreeNode(self.t, False)
            new_root.children.insert(0, self.root)
            self._split_child(new_root, 0)
            self._insert_non_full(new_root, k)
            self.root = new_root
        else:
            self._insert_non_full(self.root, k)

    def _insert_non_full(self, node, k):
        i = len(node.keys) - 1
        if node.leaf:
            node.keys.append(None)
            while i >= 0 and k < node.keys[i]:
                node.keys[i + 1] = node.keys[i]
                i -= 1
            node.keys[i + 1] = k
        else:
            while i >= 0 and k < node.keys[i]:
                i -= 1
            i += 1
            if len(node.children[i].keys) == (2 * self.t) - 1:
                self._split_child(node, i)
                if k > node.keys[i]:
                    i += 1
            self._insert_non_full(node.children[i], k)

    def _split_child(self, parent, i):
        t = self.t
        y = parent.children[i]
        z = BTreeNode(t, y.leaf)
        parent.children.insert(i + 1, z)
        parent.keys.insert(i, y.keys[t - 1])
        z.keys = y.keys[t:(2 * t - 1)]
        y.keys = y.keys[0:t - 1]
        if not y.leaf:
            z.children = y.children[t:(2 * t)]
            y.children = y.children[0:t]

    def visualizar(self):
        dot = Digraph()
        self._graficar(self.root, dot)
        dot.render('arbol_b', format='png', cleanup=True)
        return 'arbol_b.png'

    def _graficar(self, nodo, dot, parent_id=None):
        nodo_id = str(id(nodo))
        etiqueta = '|'.join(str(k) for k in nodo.keys)
```

```
        dot.node(nodo_id, etiqueta, shape='record')
        if parent_id:
            dot.edge(parent_id, nodo_id)
        for child in nodo.children:
            self._graficar(child, dot, nodo_id)

# ========================= Aplicación GUI ============================
entidades_cargadas = []
rutas_recomendadas = []

def aplicar_estilo(widget):
    widget.configure(bg="#2e2e2e")
    style = ttk.Style()
    style.configure("TButton", font=("Segoe UI", 11), padding=6)
    style.configure("TLabel", font=("Segoe UI", 11), background="#2e2e2e",
foreground="white")
    style.configure("TEntry", font=("Segoe UI", 11))

def cargar_entidades():
    global entidades_cargadas
    entidades_cargadas = []

    ruta = filedialog.askopenfilename(filetypes=[("CSV Files", "*.csv")])
    if ruta:
        try:
            with open(ruta, encoding='utf-8') as archivo:
                next(archivo)
                for linea in archivo:
                    datos = linea.strip().split(',')
                    entidad = {
                        "id": datos[0],
                        "nombre": datos[1],
                        "tipo": datos[2],
                        "lat": float(datos[3]),
                        "lon": float(datos[4]),
                        "precio": float(datos[5]),
                        "calificacion": float(datos[6]),
                        "tiempo": float(datos[7]) if datos[7] else 0
                    }
                    entidades_cargadas.append(entidad)
            messagebox.showinfo("Carga exitosa", "Entidades cargadas
correctamente.")
        except Exception as e:
            messagebox.showerror("Error", f"No se pudo cargar el
archivo:\n{str(e)}")

def cargar_calificaciones():
    ruta = filedialog.askopenfilename(filetypes=[("CSV Files", "*.csv")])
```

```
    if ruta:
        try:
            with open(ruta, encoding='utf-8') as archivo:
                for linea in archivo:
                    datos = linea.strip().split(',')
                    print("Calificación cargada:", datos)
            messagebox.showinfo("Carga exitosa", "Calificaciones cargadas
correctamente.")
        except Exception as e:
            messagebox.showerror("Error", f"No se pudo cargar el
archivo:\n{str(e)}")

def calcular_rutas(origen, presupuesto, resultado_text):
    global rutas_recomendadas
    rutas_recomendadas = []
    try:
        presupuesto = float(presupuesto)
    except ValueError:
        messagebox.showerror("Error", "El presupuesto debe ser un número.")
        return

    rutas_validas = []
    for e in entidades_cargadas:
        if e["nombre"].lower() == origen.lower():
            continue
        if e["tipo"].lower() != "turístico":
            continue
        if e["precio"] <= presupuesto and (e["tiempo"] + 0.5) <= 8:
            score = e["calificacion"] / (1 + e["precio"])
            rutas_validas.append((e, score))

    rutas_validas.sort(key=lambda x: x[1], reverse=True)

    resultado_text.delete(1.0, tk.END)
    if not rutas_validas:
        resultado_text.insert(tk.END, "No se encontraron rutas con el presupuesto
disponible.")
    else:
        rutas_recomendadas = [e[0] for e in rutas_validas[:5]]
        for i, e in enumerate(rutas_recomendadas, start=1):
            resultado_text.insert(tk.END, f"{i}. {e['nombre']} - Q{e['precio']},
{e['tiempo']}h, {e['calificacion']}⭐\n")

def abrir_formulario_lugar():
    form = tk.Toplevel()
    form.title("Agregar Lugar")
    form.geometry("400x500")
    aplicar_estilo(form)
```

```
    frame = ttk.Frame(form, padding=20)
    frame.pack(expand=True)
    campos = ["ID", "Nombre", "Tipo (Hospedaje/Turístico)", "Latitud",
"Longitud", "Precio", "Calificación", "Tiempo (si turístico)"]
    entradas = {}
    for campo in campos:
        ttk.Label(frame, text=campo).pack()
        entrada = ttk.Entry(frame)
        entrada.pack(pady=2)
        entradas[campo] = entrada

    def guardar_lugar():
        datos = {campo: entradas[campo].get() for campo in campos}
        print("Nuevo lugar:", datos)
        messagebox.showinfo("Éxito", "Lugar agregado correctamente.")
        form.destroy()

    ttk.Button(frame, text="Guardar", command=guardar_lugar).pack(pady=20)

def abrir_formulario_calificacion():
    form = tk.Toplevel()
    form.title("Calificar Lugar")
    form.geometry("400x300")
    aplicar_estilo(form)
    frame = ttk.Frame(form, padding=20)
    frame.pack(expand=True)

    ttk.Label(frame, text="ID del lugar:").pack()
    entry_id = ttk.Entry(frame)
    entry_id.pack()
    ttk.Label(frame, text="Calificación (1-5):").pack()
    entry_cal = ttk.Entry(frame)
    entry_cal.pack()
    ttk.Label(frame, text="Comentario (opcional):").pack()
    entry_com = ttk.Entry(frame)
    entry_com.pack()

    def guardar_calificacion():
        id_lugar = entry_id.get()
        calificacion = float(entry_cal.get())
        comentario = entry_com.get()
        print(f"Calificación para {id_lugar}: {calificacion}, Comentario:
{comentario}")
        messagebox.showinfo("Éxito", "Calificación registrada.")
        form.destroy()

    ttk.Button(frame, text="Guardar", command=guardar_calificacion).pack(pady=20)
```

```
def mostrar_mapa(entidades):
    if not entidades:
        messagebox.showinfo("Mapa", "No hay rutas recomendadas para mostrar.")
        return

    mapa = folium.Map(location=[14.6349, -90.5069], zoom_start=13)
    for lugar in entidades:
        folium.Marker(
            location=[float(lugar['lat']), float(lugar['lon'])],
            popup=f"{lugar['nombre']} ({lugar['tipo']})\nQ{lugar['precio']} -
{lugar['calificacion']}⭐",
        ).add_to(mapa)

    mapa.save("mapa.html")
    webbrowser.open("mapa.html")

def generar_arbol_b():
    if not entidades_cargadas:
        messagebox.showwarning("Advertencia", "Debe cargar primero las
entidades.")
        return

    arbol_b = BTree(3)
    for entidad in entidades_cargadas:
        arbol_b.insert(entidad["nombre"])

    imagen_path = arbol_b.visualizar()
    webbrowser.open(imagen_path)

def abrir_ventana1():
    root.withdraw()
    ventana1 = tk.Toplevel()
    ventana1.title("Carga de datos")
    ventana1.geometry("500x300")
    aplicar_estilo(ventana1)
    frame = ttk.Frame(ventana1, padding=20)
    frame.pack(expand=True)
    ttk.Label(frame, text="Carga de Entidades").pack(pady=5)
    ttk.Button(frame, text="Cargar archivo CSV de entidades",
command=cargar_entidades).pack(pady=5)
    ttk.Label(frame, text="Carga de Calificaciones").pack(pady=5)
    ttk.Button(frame, text="Cargar archivo CSV de calificaciones",
command=cargar_calificaciones).pack(pady=5)
    ttk.Button(frame, text="Volver a principal", command=lambda:
regresar(ventana1)).pack(pady=20)

def abrir_ventana2():
    root.withdraw()
```

```
    ventana2 = tk.Toplevel()
    ventana2.title("Recomendaciones de Rutas")
    ventana2.geometry("800x700")
    aplicar_estilo(ventana2)
    frame = ttk.Frame(ventana2, padding=20)
    frame.pack(expand=True)

    ttk.Label(frame, text="Recomendaciones de Rutas").pack(pady=10)
    ttk.Label(frame, text="Punto de origen:").pack()
    entry_origen = ttk.Entry(frame)
    entry_origen.pack(pady=5)
    ttk.Label(frame, text="Presupuesto diario:").pack()
    entry_presupuesto = ttk.Entry(frame)
    entry_presupuesto.pack(pady=5)
    resultado_text = tk.Text(frame, height=10, width=60)
    resultado_text.pack(pady=10)
    ttk.Button(frame, text="Calcular rutas recomendadas",
               command=lambda: calcular_rutas(entry_origen.get(),
entry_presupuesto.get(), resultado_text)).pack(pady=10)
    ttk.Button(frame, text="Agregar nuevo lugar",
command=abrir_formulario_lugar).pack(pady=5)
    ttk.Button(frame, text="Calificar lugar",
command=abrir_formulario_calificacion).pack(pady=5)
    ttk.Button(frame, text="Mostrar Mapa", command=lambda:
mostrar_mapa(rutas_recomendadas)).pack(pady=5)
    ttk.Button(frame, text="Volver a principal", command=lambda:
regresar(ventana2)).pack(pady=20)

def abrir_ventana3():
    root.withdraw()
    ventana3 = tk.Toplevel()
    ventana3.title("Estructura de Datos")
    ventana3.geometry("400x300")
    aplicar_estilo(ventana3)
    frame = ttk.Frame(ventana3, padding=20)
    frame.pack(expand=True)

    ttk.Label(frame, text="Estructura de Datos", font=("Segoe UI", 16,
"bold")).pack(pady=10)
    ttk.Button(frame, text="Generar Árbol B",
command=generar_arbol_b).pack(pady=10)
    ttk.Button(frame, text="Volver a principal", command=lambda:
regresar(ventana3)).pack(pady=20)

def regresar(ventana):
    ventana.destroy()
    root.deiconify()
```

```
root = tk.Tk()
root.title("Ventana Principal")
root.geometry("600x400")
aplicar_estilo(root)
frame = ttk.Frame(root, padding=20)
frame.pack(expand=True)

ttk.Label(frame, text="Bienvenido", font=("Segoe UI", 16, "bold"),
background="#f0f0f0", foreground="black").pack(pady=10)
ttk.Label(frame, text="Seleccione la opción que desea utilizar",
background="#f0f0f0", foreground="black").pack(pady=5)
ttk.Button(frame, text="Carga de datos desde archivo .csv",
command=abrir_ventana1).pack(pady=5, fill="x")
ttk.Button(frame, text="Recomendaciones de rutas",
command=abrir_ventana2).pack(pady=5, fill="x")
ttk.Button(frame, text="Estructura de datos",
command=abrir_ventana3).pack(pady=5, fill="x")

root.mainloop()
```