MANUAL TÉCNICO DEL SISTEMA “TRAVELMAP”

PRESENTACIÓN

Este manual técnico fue elaborado con el propósito de documentar en detalle el sistema *TravelMap*, una solución web desarrollada por estudiantes de Ingeniería en Sistemas para recomendar rutas turísticas y de hospedaje en Guatemala. El sistema permite gestionar lugares, hospedajes y comentarios, generar rutas óptimas basadas en tiempo, distancia y presupuesto, así como calificar sitios visitados.

Está dirigido a desarrolladores, mantenedores y personal técnico que requiera comprender el funcionamiento interno del sistema y desee realizar futuras modificaciones o implementaciones.

RESUMEN

TravelMap es una aplicación web compuesta por un backend en **Python con Flask**, un frontend en **HTML/ CSS y JS,** estructuras de datos como **Árbol B** y **Grafos ponderados** implementados desde cero, y una interfaz para interacción visual con mapas y formularios. Este manual abarca desde las herramientas utilizadas, instalación, arquitectura, diseño de base de datos, modelos, hasta los diagramas técnicos que soportan el sistema.

OBJETIVO

Describir detalladamente los aspectos técnicos del sistema *TravelMap*, su instalación, configuración, arquitectura de desarrollo, uso de librerías, estructuras de datos utilizadas, y lineamientos para mantener y escalar el sistema.

FINALIDAD DEL MANUAL

Proveer una guía estructurada y completa que permita a cualquier desarrollador:

- Entender el diseño e implementación del sistema.
- Modificarlo o extenderlo según futuras necesidades.
- Integrarse fácilmente al proyecto en caso de mantenimiento o mejora.

INTRODUCCIÓN

TravelMap responde a la necesidad de optimizar la experiencia turística mediante una solución que sugiere rutas ideales según preferencias del usuario. Además, recopila calificaciones para retroalimentar las recomendaciones. El sistema se desarrolló siguiendo buenas prácticas de ingeniería de software y patrones de arquitectura MVC.

Herramientas Utilizadas

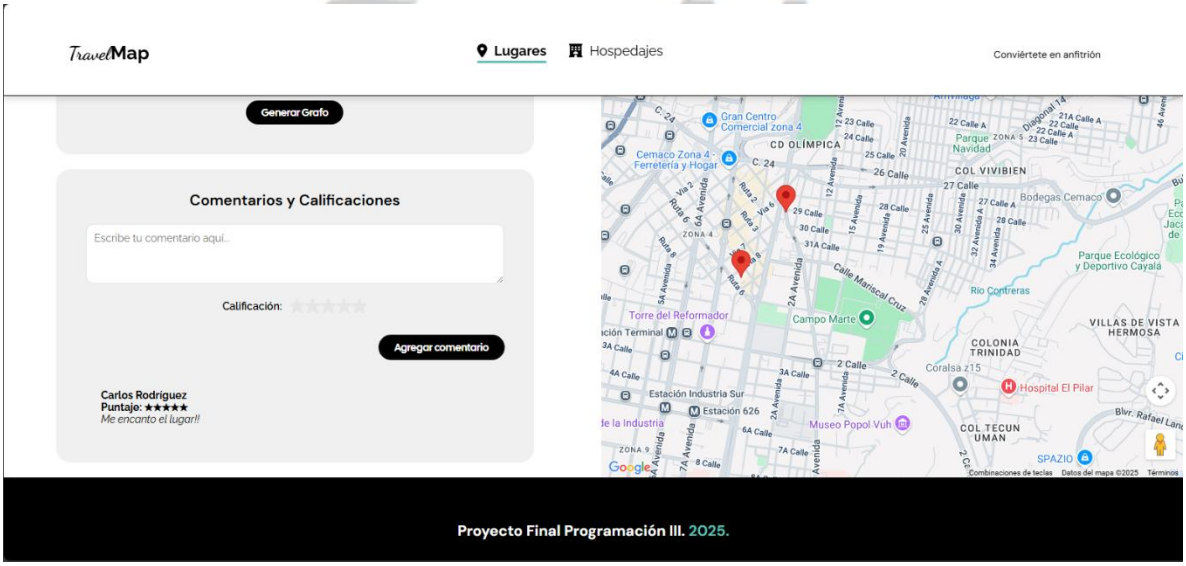
Herramienta	Descripción
Python 3.11	Lenguaje de programación principal del backend
Flask	Framework web ligero para la creación de la API
HTML/	lenguaje de etiquetas para desarrollo web
CSS	hoja de estilos para darle diseño a la página web
JS	lenguaje de programación para agregar interactividad al sitio web
Graphviz	Generación visual de estructuras de datos (Árbol B, rutas)
Google Maps API	Para renderizado de mapas y trazado de rutas en la interfaz
CSV	Medio para carga masiva de datos
VSCode / Postman / Git	IDE y herramientas de prueba e integración

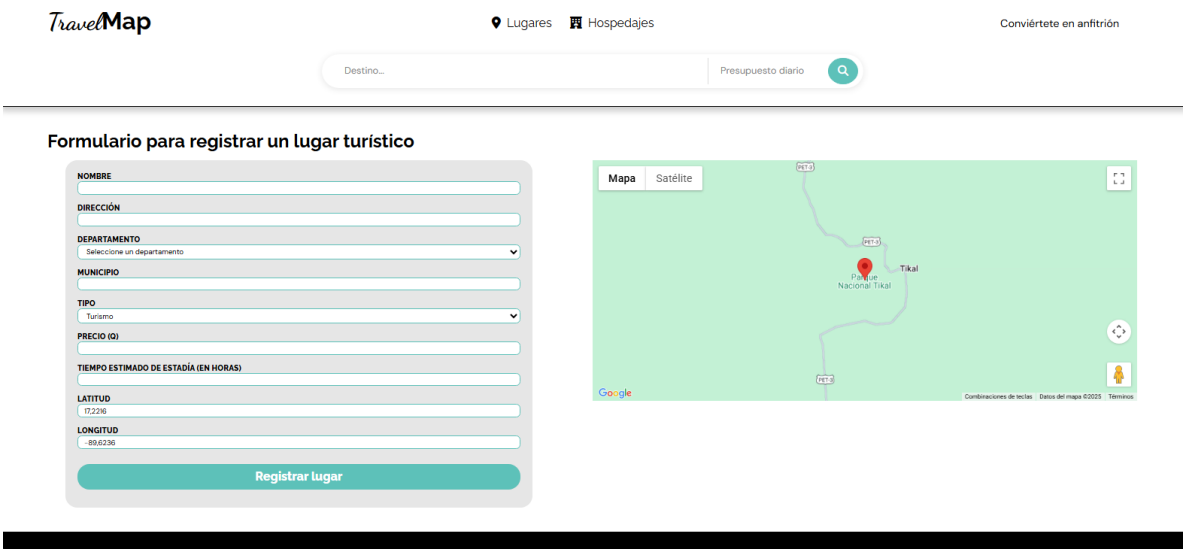
DIAGRAMAS DE MODELAMIENTO

Diagrama de Clases (Back-End)

El sistema incluye clases como:

- Lugar: representa un sitio turístico.
- Hospedaje: alojamientos disponibles.
- Calificacion: comentario y calificación de usuarios.
- BTree: árbol B personalizado para almacenar lugares y hospedajes.
- GrafoPonderado: grafo propio para rutas turísticas.





TravelMap

Lugares Hospedajes

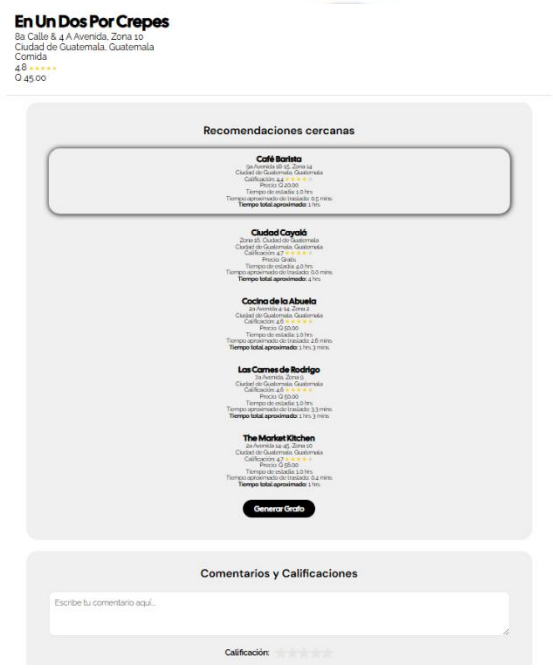
Conviértete en anfitrión

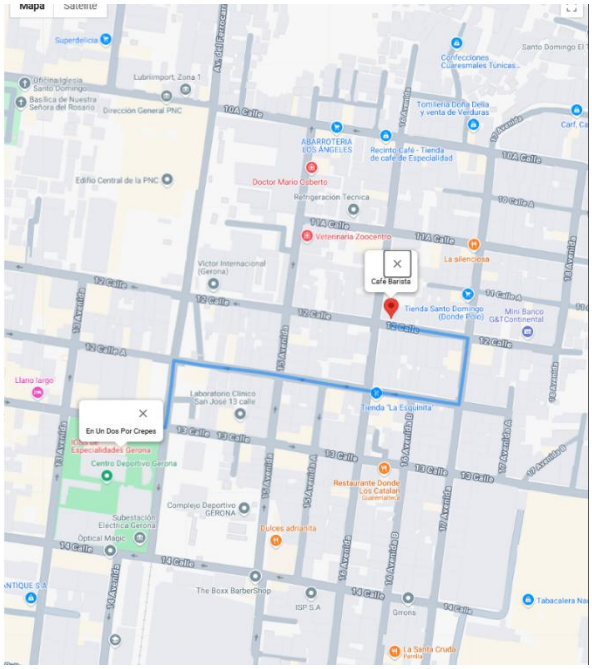
Destino... Presupuesto diario

¿De qué deseas convertirte en anfitrión?

Lugar

Hospedaje





ESTRUCTURA TÉCNICA DEL SISTEMA

3.1 Backend (Python/Flask)

- /api/lugares: consulta y registro de lugares
- /api/hospedajes: administración de hospedajes
- /api/rutas: generación de rutas mediante Dijkstra
- /api/calificaciones: guarda y muestra comentarios

Estructura modular dividida por:

- modelos/: clases Lugar, Calificacion, BTree, GrafoPonderado
- carga_csv.py: parser y validador de archivos CSV
- graficador.py: generación visual con Graphviz

3.2 Frontend (React)

- Formulario para agregar lugares y hospedajes
- Vista de lugares recomendados
- Interacción con mapa de rutas
- Sección de comentarios y calificaciones

EXPLICACIÓN DEL CÓDIGO PRINCIPAL DE TRAVELMAP

1. app.py – Servidor Flask principal

Este archivo contiene todas las rutas del backend.

```
from flask import Flask, render_template, request, jsonify
from backend.arbolB import BTree
from backend.carga_csv import cargar_lugares_csv, cargar_calificaciones_csv
from backend.modelos.GrafoPonderado import GrafoPonderado
```

Objetivo:

Inicializa la aplicación, crea árboles y grafos, y gestiona las rutas API para toda la lógica del sistema.

```
app = Flask(__name__)
arbol_lugares = BTree(grado=5)
arbol_hospedaje = BTree(grado=5)
grafo = GrafoPonderado()
```

Rutas importantes:

Carga de lugares desde CSV

```
@app.route('/api/cargar-lugares/cargacsv', methods=['POST'])
def cargar_csv():
    archivo = request.files['archivo']
    cargar_lugares_csv(archivo, arbol_lugares)
    return jsonify({'mensaje': 'Lugares cargados correctamente.'})
```

Consultar todos los lugares del árbol

```
@app.route('/api/lugares', methods=['GET'])
def obtener_lugares():
    return jsonify(arbol_lugares.recorrer())
```

2. arbolB.py – Árbol B desde cero

Propósito:

Estructura de datos eficiente para insertar y buscar lugares ordenadamente.

```
class BTree:
    def __init__(self, grado):
        self.grado = grado
        self.raiz = NodoB(grado)
```


Insertar lugar:

```
def insertar(self, lugar):
    nodo_raiz = self.raiz
    if nodo_raiz.esta_lleno():
        nuevo = NodoB(self.grado)
        nuevo.hijos.append(self.raiz)
        nuevo.dividir(0)
        self.raiz = nuevo
    self.raiz.insertar_no_lleno(lugar)
```

La lógica divide el nodo si está lleno, asegurando balance.

3. GrafoPonderado.py – Representación del mapa

Estructura:

```
class GrafoPonderado:
    def __init__(self):
        self.vertices = {}  # {lugar: {vecino: peso}}
```

Agregar conexión:

```
def agregar_arista(self, origen, destino, peso):
    if origen not in self.vertices:
        self.vertices[origen] = {}
    self.vertices[origen][destino] = peso
```

Dijkstra:

```
def dijkstra(self, origen):
    distancias = {v: float('inf') for v in self.vertices}
    distancias[origen] = 0
    visitados = set()
    while visitados != set(self.vertices):
        actual = min((n for n in distancias if n not in visitados), key=lambda n:
distancias[n])
        for vecino, peso in self.vertices[actual].items():
            nueva_dist = distancias[actual] + peso
            if nueva_dist < distancias[vecino]:
                distancias[vecino] = nueva_dist
        visitados.add(actual)
    return distancias
```

4. carga_csv.py – Carga masiva

```
def cargar_lugares_csv(archivo, arbol_b):
```

```
    decoded = archivo.read().decode('utf-8')
    datos = csv.reader(io.StringIO(decoded))
    for fila in datos:
        lugar = Lugar(nombre=fila[0], tipo=fila[1], ...)
        arbol_b.insertar(lugar)
```

Lee línea por línea y genera objetos Lugar o Hospedaje que se insertan en el árbol.

5. Lugar.py / Hospedaje.py – Modelo de entidades


```
class Lugar:
    def __init__(self, nombre, tipo, departamento, presupuesto):
        self.nombre = nombre
        self.tipo = tipo
        self.departamento = departamento
        self.presupuesto = presupuesto
```

6. calificacion.py – Comentarios y ratings

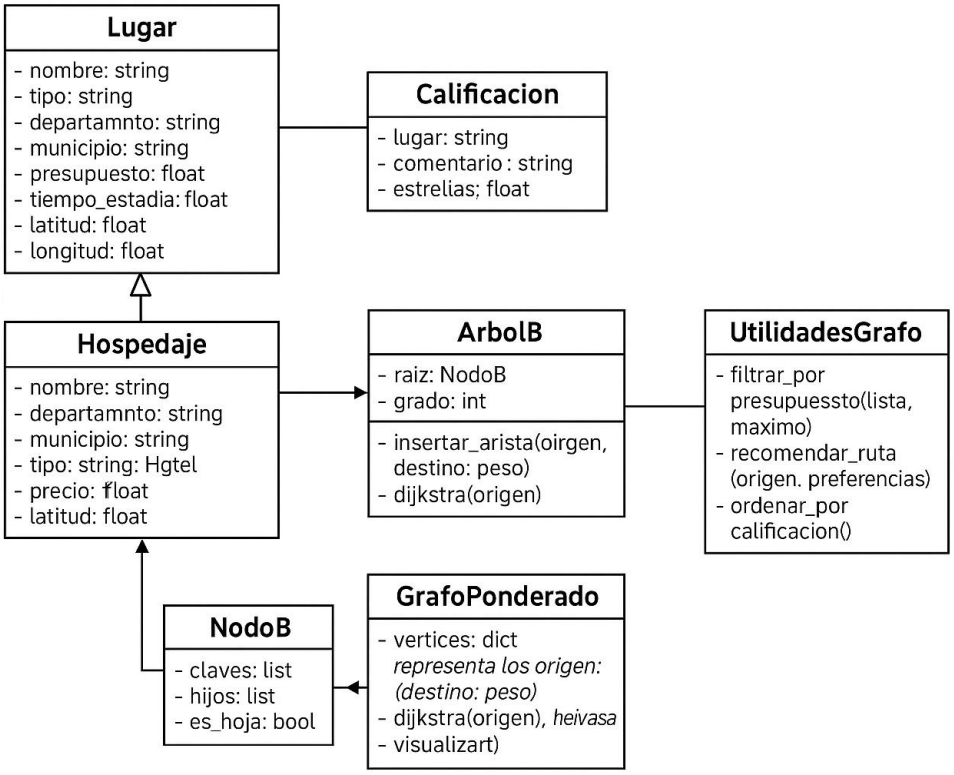
```
class Calificacion:
    def __init__(self, nombre_lugar, comentario, estrellas):
        self.nombre_lugar = nombre_lugar
        self.comentario = comentario
        self.estrellas = estrellas
```

Y se guardan en CSV así:

```
def guardar_calificacion_en_csv(nombre_lugar, comentario, estrellas):
```

```
  with open('data/calificaciones.csv', 'a', newline='', encoding='utf-8') as archivo:
      writer = csv.writer(archivo)
      writer.writerow([nombre_lugar, comentario, estrellas])
```

DIAGRAMA DE CLASES



ALGORITMO DE RECOMENDACIONES – *TravelMap*

Estructura de datos utilizada

Grafo ponderado

El sistema utiliza un **grafo dirigido y ponderado**, donde:

- Cada **nodo** representa un **lugar turístico o hospedaje**.
- Cada **arista** representa una **conexión posible** entre dos lugares.
- Cada arista tiene un **peso**, que representa tiempo estimado de traslado.

```
grafo = {
"Lugar A": {"Lugar B": 2.5, "Lugar C": 1.0},
"Lugar B": {"Lugar D": 3.2},
}
```

Algoritmo usado: Dijkstra

Se implementa el **algoritmo de Dijkstra** para encontrar la **ruta más corta** (en tiempo) desde un origen a todos los destinos.

¿Cómo funciona?

1. Se asigna distancia 0 al nodo de origen y ∞ a los demás.
2. Se elige el nodo no visitado con menor distancia.
3. Se actualizan las distancias de sus vecinos si se encuentra un camino más corto.
4. Se repite hasta visitar todos los nodos.

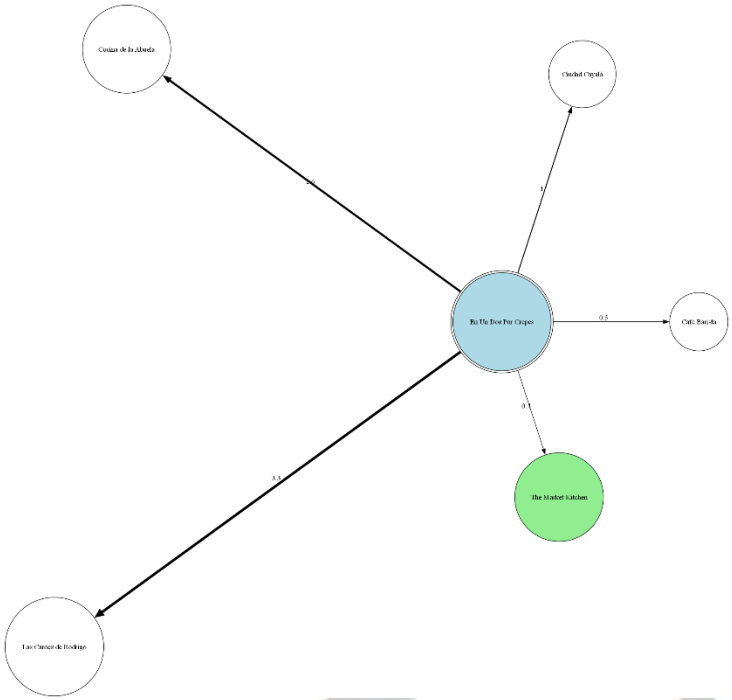


Diagrama de flujo del algoritmo

[Inicio]

↓

[Seleccionar lugar de origen]

↓

[Ejecutar Dijkstra]

↓

[Obtener rutas con tiempo mínimo]

↓

[Filtrar lugares por tipo, calificación y presupuesto]

↓

[Ordenar resultados]

↓

[Mostrar en mapa y lista]

↓

[Fin]

ESTRUCTURA DEL CÓDIGO FUENTE – SISTEMA *TRAVELMAP*


```
proyectoFinal/
│
├── backend/                ← Lógica del sistema (algoritmos y estructuras)
│   ├── modelos/            ← Clases de datos y estructuras
│   │   ├── arbolB.py       ← Implementación de Árbol B
│   │   ├── calificacion.py ← Modelo de calificación individual
│   │   ├── calificaciones.py ← Operaciones sobre múltiples calificaciones
│   │   ├── GrafoPonderado.py ← Implementación de grafo ponderado
│   │   ├── lugar.py        ← Clase `Lugar` con sus atributos
│   │   ├── recomendacion.py ← Lógica para generar rutas recomendadas
│   │   ├── utilidadesGrafo.py ← Métodos auxiliares de grafos
│   │   └── __init__.py     ← Inicializador del paquete
│   │
│   ├── carga_csv.py        ← Funciones para carga masiva desde CSV
```

```
│   └── __init__.py          ← Inicializador del backend
│
├── data/                    ← Archivos de datos
│   ├── datos.csv            ← Lugares turísticos y hospedajes
│   ├── ratings.csv          ← Comentarios y calificaciones de usuarios
│   └── ratings.json         ← Calificaciones estructuradas (JSON)
│
├── static/                  ← Archivos estáticos del frontend
│   ├── css/                 ← Hojas de estilo
│   └── js/                  ← Scripts del lado cliente
│
├── templates/               ← Plantillas HTML (Flask)
│   ├── index.html           ← Página principal
│   ├── cargar.html          ← Interfaz de carga de CSV
│   └── detalles.html        ← Vista detallada de un lugar
│
├── app.py                   ← Servidor Flask: enrutamiento y lógica web
├── gulpfile.js              ← Automatizador de tareas frontend (opcional)
├── package.json             ← Dependencias del frontend (React o JS)
├── package-lock.json        ← Control de versiones JS
├── README.md                ← Instrucciones de uso e instalación
├── requirements.txt         ← Librerías necesarias para Python
└── venv/                    ← Entorno virtual de Python
```

Detalle por módulo

backend/modelos/

Contiene las clases clave:

- Lugar, Hospedaje: entidades del sistema.
- BTree: estructura jerárquica para almacenar lugares.
- GrafoPonderado: grafo dirigido con pesos para rutas.
- Calificacion: modelo de comentarios de usuarios.
- UtilidadesGrafo: filtros y lógica auxiliar para recomendaciones.

backend/carga_csv.py

- Funciones que leen y validan datos desde archivos CSV.
- Inserta datos automáticamente en el Árbol B.

app.py

- Configura Flask, define las rutas (@app.route), responde peticiones del frontend.
- Conecta todos los módulos del sistema.

templates/ + static/

- Interfaz del usuario.
- HTML con integración de mapas, formularios, comentarios y rutas.

data/

- Fuente de información cargada en estructuras del backend.
- datos.csv: contiene todos los lugares y hospedajes.
- ratings.csv: contiene comentarios escritos por usuarios.