Representación y Organización del conocimiento

Alumno: Carlos Alberto Gongora
DNI: 24978736

---

1. Introducción

El presente documento detalla la arquitectura del conocimiento que sustenta el sistema experto de senderismo y turismo en Tierra del Fuego. El objetivo principal de este sistema es organizar, procesar y presentar información relevante al respecto de la provincia, facilitando la toma de decisiones por parte del usuario para la planificación de sus actividades. La información ha sido cuidadosamente extraída de fuentes expertas (documentación oficial, guías turísticas especializadas) y estructurada para permitir una inferencia lógica y eficiente.

2. Estructura y Organización del Conocimiento

El conocimiento en este sistema experto se organiza de manera **modular y jerárquica**, agrupando la información por tipo de actividad turística. Esta modularidad permite una gestión clara de los datos y una escalabilidad futura del sistema.

2.1. Base de Conocimiento (Knowledge Base)

La base de conocimiento se almacena en **archivos JSON**, que actúan como repositorios estructurados de datos. Cada archivo representa una categoría principal de información, lo que facilita la lectura y el mantenimiento. Esta elección permite una representación de datos **clara y legible por humanos**, y es fácilmente parseable por el sistema.

- `senderos.json`: Contiene la información de cada sendero (nombre, dificultad, distancia, duración, ubicación, disponibilidad estacional, notas, URL GPX).
- `museums.json`: Almacena los detalles de los museos y sitios culturales (nombre, ubicación, dirección, horarios, tarifas, descripción).
- `winter_sports.json`: Guarda datos sobre actividades y ubicaciones de deportes de invierno (ubicación, actividades, notas).
- `nature.json`: Proporciona información sobre la flora y fauna por ecosistema (ecosistema, flora, fauna, notas).
- `fishing.json`: Incluye detalles sobre la pesca deportiva (ubicación, tipo, especies, notas sobre regulaciones y temporadas).

Organización Lógica: La información dentro de cada JSON se organiza como una **lista de objetos**. Cada objeto representa una "instancia" de un concepto turístico (ej. un sendero específico, un museo particular) y sus atributos relevantes.

```
// Ejemplo de estructura en senderos.json
[
  {
    "name": "Castorera",
    "difficulty": "Baja",
    "distance": "200 mts",
    "duration": "20 min",
    "location": "Parque Nacional Tierra del Fuego",
    "seasonal_availability": "Todo el año",
    "notes": "Cartelería interpretativa...",
    "gpx_url": "https://example.com/gpx/castorera.gpx"
  },
  // ... más senderos ...
]
```

2.2. Representación del Conocimiento (Modelos Pydantic)

Para una gestión robusta del conocimiento y la interacción con la API, se utilizan **modelos de datos** definidos con la librería **Pydantic** en `models.py`. Estos modelos garantizan:

- **Validación de Datos:** Aseguran que los datos de entrada y salida de la API cumplan con la estructura y tipos esperados.
- **Serialización/Deserialización:** Facilitan la conversión de datos JSON a objetos Python y viceversa.
- **Claridad de Estructura:** Proporcionan una documentación inherente de cómo se espera que luzcan los datos.

Cada tipo de entidad en la base de conocimiento (Sendero, Museum, WinterSportLocation, NatureInfo, FishingInfo) tiene un modelo Pydantic asociado, que define los **criterios y atributos** de cada elemento.

3. Lógica de Inferencia y Toma de Decisiones

El "cerebro" del sistema experto reside en el módulo `logic.py`, que implementa las reglas y los métodos de inferencia para procesar la información de la base de conocimiento.

3.1. Métodos de Inferencia

El sistema utiliza principalmente un enfoque de **inferencia basada en reglas (If-Then)** y **árboles de decisión implícitos** (a través de la lógica de filtrado anidada) para la resolución de problemas (ej. encontrar senderos adecuados) y la toma de decisiones (ej. recomendar medidas de seguridad).

a) Inferencia Basada en Reglas (If-Then)

Este es el método predominante, especialmente visible en la función `filter_senderos` y `is_sendero_available`.

- **Reglas de Filtrado de Senderos:**
  - **IF** el usuario selecciona una `dificultad` **AND** la `dificultad` del sendero NO coincide, **THEN** se descarta el sendero.
  - **IF** el usuario selecciona una `ubicación` **AND** la `ubicación` del sendero NO coincide, **THEN** se descarta el sendero.
  - **IF** el usuario selecciona una `categoría de duración` **AND** la `duración` del sendero NO cae en esa categoría, **THEN** se descarta el sendero.
  - **IF** el usuario especifica una `fecha planificada` **OR** una `estación`, **AND** el sendero NO está disponible en ese período (según su `seasonal_availability`), **THEN** se descarta el sendero.
  - **ELSE (IF** todas las condiciones anteriores son VERDADERAS**)**, **THEN** se incluye el sendero en la lista de resultados.
- **Reglas de Disponibilidad Estacional (`is_sendero_available`):**
  - **IF** la `disponibilidad_sendero` es "Todo el año" **OR** "Habilitado", **THEN** el sendero está disponible.
  - **IF** el usuario proporciona una `fecha_planificada`:
    - **AND** `disponibilidad_sendero` es "Noviembre a Abril" **AND** `mes_planificado` está entre Nov y Abril, **THEN** disponible.
  - **IF** el usuario selecciona una `estación`:
    - **AND** `disponibilidad_sendero` está entre las `disponibilidades_permitidas_por_estacion`, **THEN** disponible.
  - **IF** `disponibilidad_sendero` es "Condicionado a clima", **THEN** asumir disponible (con advertencia de verificación manual).
- **Reglas de Recomendaciones de Seguridad (`get_safety_recommendations`):**
  - **IF** la `dificultad` seleccionada es "Alta", **THEN** incluir recomendaciones específicas de registro y guía.
  - **IF** la `dificultad` es "Media" **OR** "Alta", **THEN** incluir recomendaciones de ir acompañado y equipo esencial.
  - **ELSE** (siempre), **THEN** incluir recomendaciones generales de clima, horario y emergencia.

b) Árbol de Decisión (Representación Conceptual)

Aunque no se implementa explícitamente como un algoritmo de "árbol de decisión" entrenado, la lógica de filtrado del sistema sigue un patrón de árbol de decisión:

1. **Nodo Raíz:** Elección de la categoría de actividad (Senderismo, Museos, etc.).
2. **Nodos Intermedios (para Senderismo):** Los filtros sucesivos actúan como nodos de decisión (Dificultad, Ubicación, Duración, Estación/Fecha). Cada decisión en un filtro lleva a un subconjunto de opciones.

3. **Nodos Hoja:** La presentación de la información final (lista de senderos, información de museos, etc.) o un mensaje de "no se encontraron resultados".

Este enfoque permite al sistema reducir el espacio de búsqueda y presentar al usuario la información más relevante de manera estructurada.

3.2. Mecanismos de Recuperación Directa

Para categorías como Museos, Deportes de Invierno, Naturaleza y Pesca, la inferencia es más directa. El sistema realiza una **recuperación completa de la categoría** y la presenta, ya que no hay filtros complejos aplicados a esos conjuntos de datos en la implementación actual. Las funciones `get_all_museums()`, `get_all_winter_sports_locations()`, etc., demuestran esta recuperación directa de información predefinida.

4. Lógica de Organización del Conocimiento

La organización del conocimiento dentro del sistema se basa en los siguientes principios:

- **Modularidad por Dominio:** El conocimiento se divide en módulos (`senderos.json`, `museums.json`, etc.) que corresponden a dominios turísticos específicos. Esto facilita el mantenimiento, la comprensión y la extensión del sistema. Si se añade un nuevo tipo de actividad (ej. buceo), solo se necesitaría un nuevo archivo JSON y un endpoint asociado.
- **Jerarquización Implícita:** La interfaz de usuario (y los endpoints de la API) reflejan una jerarquía. El usuario primero elige una categoría general de actividad (nivel superior), y luego, si es aplicable (como en Senderismo), se le presentan filtros más específicos (niveles inferiores).
- **Relación Concepto-Atributo:** Dentro de cada módulo, los conceptos (ej. "Sendero", "Museo") se definen por un conjunto de atributos (`name`, `difficulty`, `hours`, etc.). Esta relación es clave para que las reglas de inferencia puedan operar sobre los datos.
- **Coherencia de Datos:** Los modelos Pydantic (`models.py`) aseguran que la estructura de los datos sea consistente en toda la aplicación, lo cual es vital para que la lógica de inferencia funcione correctamente.
- **Separación de Preocupaciones:** La base de conocimiento (`data/`), la lógica de inferencia (`logic.py`), la definición de modelos (`models.py`) y la interfaz API (`main.py`) están claramente separadas. Esto mejora la mantenibilidad del código y permite a los desarrolladores centrarse en aspectos específicos del sistema sin afectar otros.

5. Conclusión

La arquitectura del conocimiento del sistema experto de turismo en Tierra del Fuego se basa en una sólida **estructura modular de datos JSON**, una **validación rigurosa con Pydantic**, y una **lógica de inferencia basada en reglas (if-then) y árboles de decisión implícitos**. Esta organización permite al sistema procesar de manera eficiente las consultas

de los usuarios, filtrar información compleja y proporcionar recomendaciones pertinentes. La clara separación de componentes y la estructura jerárquica del conocimiento garantizan que el sistema sea robusto, fácil de mantener y adaptable a futuras expansiones, permitiendo ofrecer una experiencia turística informada y segura en el "Fin del Mundo".