TFG: Recomendación y visualización de rutas turísticas basado en IA

Resumen

El objetivo de este TFG es diseñar e implementar una aplicación que recomiende y visualice rutas turísticas personalizadas.
El sistema contará con un **motor de recomendación basado en ML clásico** y, en una segunda fase, con una **interfaz conversacional con LLMs + RAG** para que el usuario interactúe en lenguaje natural.

El flujo de desarrollo será incremental:

1. **Primero** se implementa y prueba toda la lógica de recomendación en **terminal** (entrada y salida en consola).
2. **Después**, se añade la capa conversacional con LLM + RAG.
3. **Por último**, se desarrolla la interfaz web (visualización en mapas + chat).

Motivación

La planificación de viajes suele ser compleja: implica consultar múltiples fuentes, filtrar listas largas de POIs y utilizar interfaces rígidas. Este proyecto busca **modernizar esa experiencia**, permitiendo que el usuario pueda expresar sus preferencias de forma libre (*"Quiero una tarde cultural terminando en un sitio de tapas"*), y que el sistema genere rutas personalizadas de forma automática.

El valor añadido está en la combinación de **IA clásica (ML)**, que proporciona un recomendador sólido y medible, y **IA generativa (LLM + RAG)**, que aporta interacción natural y explicaciones comprensibles al usuario.

Objetivos

- Procesar y estructurar un dataset de rutas históricas (std_2018 de Semantic Trails + venues de Foursquare).
- Implementar un **sistema de recomendación en Python** con métodos de ML clásico.
- Soportar dos tipos de recomendación:
    1. **Según perfil del usuario** (si aparece en el dataset y tiene historial).
    2. **Según información proporcionada en el momento** (categorías, duración, etc.).
- Probar primero en **terminal**, con entrada de parámetros o texto.
- Integrar un **chatbot con LLM + RAG** para que el usuario pueda introducir sus preferencias en lenguaje natural.

- Implementar una **interfaz web** para visualizar las rutas en mapas y mostrar la interacción conversacional.

Procesamiento de dataset

Fuentes de datos:

- Semantic Trails Dataset (std_2018): rutas históricas de usuarios en Foursquare.
- Foursquare API: permite obtener coordenadas, categorías, horarios y más información de cada venue_id.

Pasos de procesamiento (ETL):

1. Descargar y explorar std_2018. Columnas principales: trail_id, user_id, venue_id, timestamp, schema_category.
2. Mapear cada venue_id con sus coordenadas (lat, lon), nombre, categoría y horarios. Esto se puede hacer con:
    - venues.csv del propio proyecto.
    - API de Foursquare (si faltan datos).
3. Crear un **dataset unificado** con campos:
    - poi_id, name, lat, lon, city, country, category, schema_category, description, opening_hours.
    - trail_id y user_id para enlazar a rutas históricas.
4. Crear embeddings de descripciones/categorías (ejemplo: sentence-transformers/all-MiniLM-L6-v2).
5. Guardar embeddings en una base vectorial (FAISS o Qdrant).

Resultado esperado de esta fase:

- Dataset estructurado y enriquecido.
- Base de conocimiento vectorial lista para búsqueda semántica.
- Scripts reproducibles de limpieza y unión de datos.

Tipos de recomendación

El sistema ofrecerá **dos tipos de recomendación**, según si el usuario existe en el dataset o no:

1. **Usuario existente (perfil con historial)**
    - Si el user_id aparece en std_2018, se usa su historial de trails.
    - Se aplica **colaborative filtering** (similaridad con otros usuarios, factorización matricial).

- El usuario podrá recibir:
    - Recomendaciones de POIs no visitados pero comunes entre usuarios similares.
    - Nuevas rutas basadas en su comportamiento histórico.

2. **Usuario nuevo (sin perfil registrado)**
    - El sistema solo usará la información proporcionada (categoría deseada, duración, ciudad, etc.).
    - En primera instancia, la interacción será en **terminal** (el usuario introduce texto o parámetros).
    - Posteriormente, el chatbot permitirá hacerlo en lenguaje natural.
    - Se aplican técnicas de **content-based filtering** y **clustering** para construir la ruta.

Métodos de ML clásico a implementar

1. Content-based filtering

- Se representa cada POI con características (categoría, localización, horarios).
- Se buscan POIs similares a la consulta del usuario mediante k-NN.
- Útil para **usuarios nuevos**.

2. Collaborative filtering

- Matriz usuario–POI con check-ins de std_2018.
- Factorización matricial (SVD/ALS) para predecir POIs de interés.
- Útil para **usuarios con historial**.

3. Clustering de POIs

- Agrupa POIs en "tipos de experiencias" (ejemplo: cultural, gastronómica, ocio).
- Algoritmo: KMeans o DBSCAN con coordenadas + categorías.
- Permite recomendar grupos de lugares similares.

4. Optimización de rutas

- Una vez seleccionados los POIs candidatos:
    - Heurística de vecino más cercano para ordenarlos.
    - Ajustes por horarios (si está cerrado, se penaliza).
    - Llamada a **Geoapify Routing API** para obtener la ruta realista (distancias, tiempos).

Flujo de recomendación

1. **Usuario existente (con perfil en el dataset)**
    - Se busca el user_id en std_2018.
    - Se cargan sus trails previos (secuencias de POIs).
    - Se aplica **collaborative filtering** para recomendar POIs similares a los visitados por usuarios parecidos.
    - Se genera un itinerario nuevo a partir de esas recomendaciones.
2. **Usuario nuevo (sin historial)**
    - El usuario introduce sus preferencias (categoría, tiempo disponible, ciudad, etc.).
    - En primera instancia, lo hace por **terminal**.
    - Cuando se integre el chatbot, lo hará en lenguaje natural.
    - El sistema aplica **content-based filtering** y/o clustering de POIs para generar la ruta.
3. **Ruta final optimizada**
    - Se seleccionan POIs candidatos (por historial o por preferencias).
    - Se ordenan con heurísticas (vecino más cercano, corrección por horarios).
    - Se consulta la **Geoapify Routing API** para obtener distancias y tiempos reales, devolviendo la ruta en formato GeoJSON para su visualización en mapas.

Fases del proyecto

Fase 1 – Procesamiento de dataset (septiembre – noviembre)

- ETL con std_2018 y enriquecimiento con venues/Foursquare API.
- Creación del dataset estructurado con coordenadas, categorías y descripciones.
- Creación de embeddings y base vectorial.
- Visualización inicial de rutas históricas en mapa (Folium o Streamlit).

Resultado: dataset limpio y reproducible + demo de rutas reales en mapa.

Fase 2 – Implementación ML clásico (noviembre – diciembre)

- Desarrollo de un recomendador en Python ejecutable en **terminal**.
- Implementación de:
    - Content-based filtering.
    - Collaborative filtering (usuarios existentes).

    - Clustering de POIs.
- Optimización de rutas con heurísticas simples + Geoapify Routing API.

Resultado: recomendador funcional en consola (entrada de parámetros, salida de POIs + ruta).

Fase 3 – Backend y API (diciembre – enero, Navidades)

- Desarrollo del backend con **FastAPI**.
- Endpoints:
    - /recommend (recibe parámetros o consulta textual, devuelve POIs + ruta).
    - /user/<id> (recomendación para usuario existente).
- Integración de dataset y modelos ML en el backend.

Resultado: backend con API REST listo para ser usado por el frontend.

Fase 4 – Chatbot y RAG (enero – febrero, Florida)

- Añadir LLM para procesar consultas en lenguaje natural.
- Implementar RAG: búsqueda de POIs relevantes en la base vectorial.
- Construir prompts que combinen:
    - Preferencias del usuario.
    - Contexto de POIs recuperados.
- El LLM devuelve explicaciones naturales de las rutas.

Resultado: interacción conversacional con el recomendador (por texto/chat).

Fase 5 – Interfaz web y visualización (febrero – marzo, Florida)

- Interfaz web con **Streamlit** (rápido) o **React + Leaflet** (más avanzado).
- Componentes:
    - Chat con el sistema.
    - Mapa interactivo mostrando rutas históricas y recomendadas.
- Conexión frontend ↔ backend (API REST).

Resultado: demo visual del sistema con conversación y mapa.

Fase 6 – Pruebas, evaluación y memoria (marzo – inicios abril)

- Comparar rutas recomendadas vs. rutas históricas:

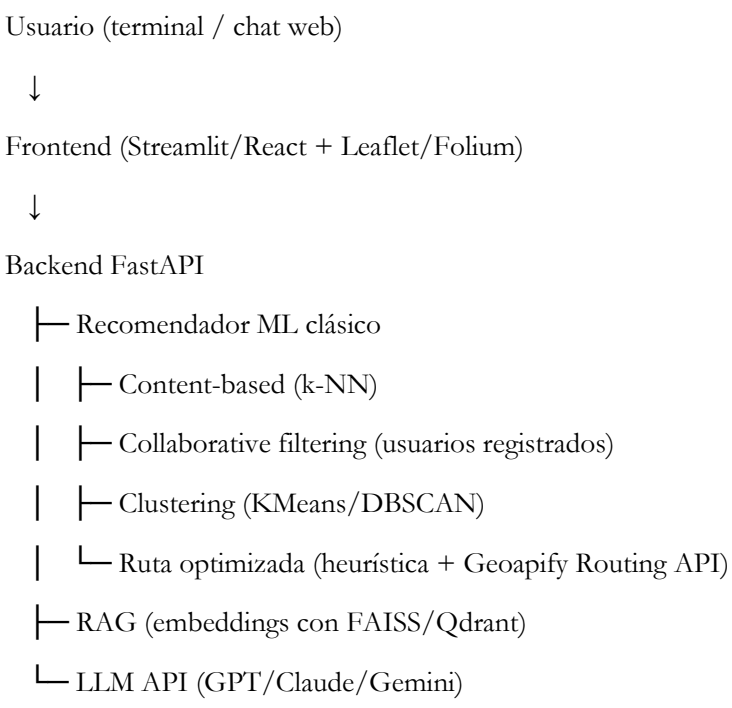
    - Cobertura de categorías pedidas.
    - Similitud de itinerarios (categorías, secuencias de POIs).
- Evaluar eficiencia de las rutas (distancia, duración).
- Ajustes de modelos, prompts y lógica de recuperación.
- Redacción de la memoria del TFG.
- Preparación de un vídeo demo (2–3 min).

Resultado: TFG finalizado a principios de abril, listo para defensa en mayo.

Cronograma

- **Septiembre – Octubre**: Procesamiento inicial del dataset (std_2018, venues, Foursquare API).
- **Noviembre**: Dataset estructurado + embeddings listos.
- **Diciembre**: Recomendador ML clásico en terminal.
- **Navidades (finales diciembre – enero)**: Backend FastAPI con API REST.
- **Enero – Febrero (Florida, remoto)**: Chatbot con LLM + RAG, pruebas iniciales.
- **Febrero – Marzo (Florida)**: Desarrollo de interfaz web (visualización en mapas + chat).
- **Marzo – Inicios abril**: Evaluación + memoria + vídeo.
- **Abril**: Entrega final.
- **Mayo**: Defensa del TFG.

Arquitectura



Estructura de carpetas propuesta

```
tfg-tourism-route-recommendation/
├── data/                    # Todo lo relacionado con datos
│   ├── raw/                 # Datos originales (std_2018.csv, venues.csv)
│   ├── external/            # Datos extra (llamadas a Foursquare API, Geoapify)
│   ├── processed/           # Dataset limpio y estructurado listo para ML
│   └── embeddings/          # Vectores generados (FAISS/Qdrant dumps)
│
├── notebooks/               # Notebooks Jupyter para exploración y prototipos
│   ├── 01_exploration.ipynb     # Análisis inicial de std_2018
│   ├── 02_etl_dataset.ipynb     # Limpieza y unión con venues/Foursquare
│   ├── 03_recommender_baseline.ipynb # ML clásico en terminal
│   └── 04_embeddings_rag.ipynb  # Creación de embeddings para RAG
│
├── src/                     # Código principal del proyecto
│   ├── etl/                 # Scripts de ETL (procesamiento dataset)
│   │   └── preprocess.py
│   ├── recommender/         # Lógica del recomendador clásico
│   │   ├── content_based.py
│   │   ├── collaborative.py
│   │   ├── clustering.py
│   │   └── routing.py       # Conexión con Geoapify API
│   ├── rag/                 # Módulo de RAG (embeddings + recuperación)
│   │   ├── retriever.py
│   │   └── index_faiss.py
│   ├── llm/                 # Conexión con APIs de LLM (GPT, Claude, etc.)
│   │   └── chatbot.py
│   ├── api/                 # Backend con FastAPI
│   │   ├── main.py          # Endpoints principales
│   │   └── routes/          # Rutas de API (endpoints REST)
│   └── web/                 # Frontend (Streamlit o React/Leaflet)
│       └── app.py
│
├── tests/                   # Pruebas unitarias y de integración
│   ├── test_recommender.py
│   └── test_rag.py
├── docs/                    # Documentación, diagramas, artículos
│   ├── diagrams/            # Arquitectura, flujos, esquemas
│   └── references/          # Papers, artículos, apuntes
├── reports/                 # Resultados y entregables intermedios
│   ├── figures/             # Gráficas, imágenes para la memoria
│   └── metrics/             # Tablas de evaluación
│
├── requirements.txt         # Dependencias del proyecto
```

```
├─ README.md                  # Explicación inicial del repositorio
├─ .gitignore                 # Archivos a ignorar en GitHub
└─ docker-compose.yml         # (Opcional) para levantar Qdrant, FastAPI, etc.
```