Backend – Sistema de Recuperación de Información

Este directorio contiene el backend del sistema de recuperación de información. Está implementado en Python con FastAPI y cubre todo el flujo: preprocesado, indexación por bloques (SPIMI) y búsqueda basada en TF-IDF.

Características clave:

- Indexación por bloques con escritura en disco (postings + índices de offset).
- Detección automática de idioma por documento (Lingua) para stopwords y stemming dinámicos.
- Búsquedas eficientes leyendo postings por offset y resolviendo metadatos con SQLite.
- Paralelismo controlado con backpressure y reciclado opcional de workers.

Tecnologías utilizadas

- Python 3.11+ recomendado (el reciclado de workers depende de soporte del runtime).
- FastAPI + Uvicorn.
- NLTK (stopwords, SnowballStemmer).
- Lingua Language Detector (offline).
- Pydantic + pydantic-settings.
- SQLite (índice de offsets del doc_store).

Probado en Arch Linux, pero portable a cualquier sistema compatible con Python.

Estructura del backend

```
backend/
├── app/
│   ├── main.py             # Punto de entrada de FastAPI
│   ├── api/
│   │   └── routes.py       # Definición de endpoints
│   ├── core/
│   │   └── config.py       # Configuración global
│   ├── schemas/
│   │   ├── requests.py     # Modelos de entrada (Pydantic)
│   │   └── responses.py    # Modelos de salida (Pydantic)
│   ├── services/
│   │   ├── preprocess.py   # Pipeline de preprocesado (por idioma)
│   │   ├── langdetect.py   # Detección automática de idioma
│   │   ├── indexer.py      # Indexación por bloques (SPIMI)
│   │   └── searcher.py     # Motor de búsqueda (bloques)
│   └── storage/
│       └── paths.py        # Gestión de rutas de datos
├── scripts/
│   ├── install.sh          # Instalación del entorno y dependencias
│   ├── dev.sh              # Arranque del servidor en modo desarrollo
│   ├── download_nltk.py    # Descarga de recursos NLTK
```

```
│   ├── smoke_test.sh        # Pruebas rápidas de endpoints
│   └── wikiextractor_to_jsonl.py # Conversión WikiExtractor -> JSONL
├── requirements.txt
└── README.md
```

Estructura de datos (fuera de `backend/`)

```
data/
├── raw/
│   └── corpus.jsonl         # Corpus de entrada por defecto
├── processed/               # Reservado para futuras etapas
└── indexes/
    ├── index.meta.json      # Metadatos del índice (formato block)
    ├── index.postings       # Postings por término (archivo de texto
binario)
    ├── index.terms.json     # Diccionario term -> [offset, length]
    ├── doc_store.jsonl      # Metadatos por documento (una línea por doc)
    └── doc_store.sqlite     # doc_id -> offset en doc_store.jsonl
```

Durante la indexación también se generan temporalmente:

- `data/indexes/blocks/` con los bloques parciales.
- `data/indexes/doc_store_parts/` con fragmentos del doc_store.

Si `INDEX_KEEP_BLOCKS=false` estos directorios se eliminan al finalizar.

Instalación

Desde el directorio `backend/`:

```
chmod +x scripts/*.sh
./scripts/install.sh
```

Este script:

1. Crea el entorno virtual `.venv`.
2. Instala dependencias.
3. Descarga recursos de NLTK.

Ejecución del backend

```
./scripts/dev.sh
```

API en:

```
http://localhost:8000
```

Swagger UI:

```
http://localhost:8000/docs
```

Configuración (app/core/config.py)

La configuración se carga desde `.env` (opcional) y variables de entorno.

Parámetros principales

- `DEFAULT_LANGUAGE`: idioma por defecto para endpoints de preprocesado.
- `DEFAULT_QUERY_LANGUAGE`: idioma por defecto si la consulta es “unknown”.
- `TOP_K`: número de resultados para `/search`.
- `MIN_TOKEN_LEN`: longitud mínima para filtrar tokens.
- `MIN_DF`: frecuencia mínima de documento para conservar un término.
- `MAX_DF_RATIO`: ratio máximo (respecto a N) para conservar un término.
- `INDEX_WORKERS`: número de procesos para indexación.
- `INDEX_BLOCK_DOCS`: documentos por bloque.
- `INDEX_MAX_IN_FLIGHT`: máximo de tareas en vuelo (0 = auto = 2 * workers).
- `INDEX_MAX_TASKS_PER_CHILD`: reciclado de workers (0 = desactivado).
- `INDEX_KEEP_BLOCKS`: conserva bloques y doc_store_parts si es True.

Rutas

- `DATA_DIR`, `RAW_DIR`, `PROCESSED_DIR`, `INDEX_DIR`.

Flujo de indexación

El endpoint `POST /index` construye un índice por bloques (SPIMI) desde un corpus JSONL:

1. **Selección del corpus**
    - Usa `corpus_path` si se proporciona.
    - Si no, usa `data/raw/corpus.jsonl`.
2. **Particionado por offsets (sin cargar documentos)**
    - El proceso principal recorre el fichero en binario para calcular rangos `(start, end)` con `INDEX_BLOCK_DOCS` líneas por bloque.
    - Esto evita cargar millones de documentos en memoria.

3. **Paralelismo controlado**
    - Se usa `ProcessPoolExecutor` con `INDEX_WORKERS`.
    - Se limita el número de tareas en vuelo (`INDEX_MAX_IN_FLIGHT`).
    - Opcionalmente se reciclan workers con `INDEX_MAX_TASKS_PER_CHILD`.
4. **Procesado por worker (SPIMI)** Cada worker:
    - Lee su rango del JSONL y parsea documentos.
    - Aplica el pipeline:
        1. `lexical_analysis` (normalización y minúsculas).
        2. `detect_language` (Lingua).
        3. `tokenize`.
        4. `remove_stopwords` (según idioma).
        5. `filter_meaningful` (longitud mínima y no numéricos).
        6. `lemmatize_or_stem` (SnowballStemmer).
    - Calcula TF por documento y genera un índice invertido del bloque.
    - Guarda:
        - `blocks/block_XXXXXX.jsonl` con líneas `term\t[[doc_id, tf], ...]`.
        - `doc_store_parts/doc_store_XXXXXX.jsonl` con metadatos de documentos.
5. **Finalización (merge)**
    - `doc_store_parts` se concatena en `doc_store.jsonl`.
    - Se crea `doc_store.sqlite` con `doc_id -> offset`.
    - Se calcula DF por término y se filtra con `MIN_DF` y `MAX_DF_RATIO`.
    - Se fusionan los bloques con un heap en `index.postings`.
    - Se escribe `index.terms.json` con los offsets para cada término.
    - Se genera `index.meta.json` con toda la metadata del índice.
6. **Limpieza**
    - Si `INDEX_KEEP_BLOCKS` es `False`, se eliminan los bloques temporales.

Formato de los archivos de índice

- `index.meta.json`:
    - `format`: "block"
    - `N`: número total de documentos
    - `vocab_size`: tamaño del vocabulario
    - `postings_path`, `terms_index_path`, `doc_store_path`
    - `doc_index_path` + `doc_index_type` (sqlite)
- `index.postings`:
    - Archivo de texto binario.
    - Cada línea: `term\t[[doc_id, tf], ...]`
- `index.terms.json`:

    - Diccionario `term -> [offset, length]` para acceder rápidamente al postings.
- `doc_store.jsonl`:
    - Una línea por doc con `doc_id`, `title`, `url`, `snippet`.
- `doc_store.sqlite`:
    - Tabla `doc_index(doc_id TEXT PRIMARY KEY, offset INTEGER)`.

Endpoints disponibles

Preprocesado

Todos los endpoints de preprocesado son útiles para depuración y demo.

- `POST /lexical_analysis`
    - Entrada: `{ "document": "..." }`
    - Salida: `{ "normalized": "..." }`
- `POST /tokenize`
    - Entrada: `{ "document": "..." }`
    - Salida: `{ "tokens": ["..."] }`
- `POST /remove_stopwords`
    - Entrada: `{ "tokens": ["..."] }`
    - Salida: `{ "tokens": ["..."] }`
- `POST /lemmatize`
    - Entrada: `{ "tokens": ["..."] }`
    - Salida: `{ "tokens": ["..."] }`

La ponderación y la selección de términos no se exponen como endpoints. Se realizan internamente durante la indexación (TF por documento y filtrado por DF) y en tiempo de consulta (IDF).

Indexación

- `POST /index`
    - Entrada: `{ "corpus_path": "/ruta/opcional/corpus.jsonl" }`
    - Si no se especifica `corpus_path`, se usa `data/raw/corpus.jsonl`.
    - Respuesta:
        - `indexed_docs`
        - `vocab_size`
        - `index_path` (normalmente `data/indexes/index.meta.json`)

Ejemplo:

```
```

```
curl -s -X POST "http://localhost:8000/index" \
  -H "Content-Type: application/json" \
  -d '{"corpus_path":"data/raw/corpus.jsonl"}'
```

Búsqueda

- `GET /search?query=texto`
  - Aplica preprocesado similar al de indexación.
  - Si la detección de idioma es `unknown`, usa `DEFAULT_QUERY_LANGUAGE`.
  - Calcula TF-IDF en tiempo de consulta y devuelve `TOP_K` resultados.

Ejemplo:

```
curl -s "http://localhost:8000/search?query=andorra"
```

Scripts útiles

- `scripts/dev.sh`: arranca el servidor.
- `scripts/install.sh`: prepara entorno y dependencias.
- `scripts/download_nltk.py`: descarga stopwords/stemmers.
- `scripts/smoke_test.sh`: llamadas rápidas de validación.
- `scripts/wikiextractor_to_jsonl.py`: convierte dumps procesados con WikiExtractor a JSONL (ejemplo en inglés).

Notas importantes

- El índice debe existir antes de ejecutar `/search`.
- La calidad del resultado depende del corpus (incluye páginas de desambiguación).
- Se puede ajustar `MIN_DF`, `MAX_DF_RATIO`, `INDEX_BLOCK_DOCS` y `INDEX_WORKERS` para controlar calidad y rendimiento.