

Sistema de Recuperación de Información sobre Wikipedia

Práctica 4 - Recuperación de Información

Jesús J. Cantero López

Curso 2024/25

Índice

1. Introducción	3
1.1. Objetivos	3
2. Entorno de trabajo	3
2.1. Backend (Python)	3
2.2. Frontend (JavaScript/TypeScript)	4
2.3. Contenerización	4
3. Estructura del proyecto	4
3.1. Descripción de Archivos del Backend	5
3.2. Ejecución del Sistema	5
3.2.1. Opción rápida: Docker	5
3.2.2. Opción manual	5
4. Colección Documental	5
4.1. Elección de la Fuente	5
4.2. Obtención de los Datos	6
4.3. Estadísticas del Corpus	6
5. Pipeline de Preprocesamiento	6
5.1. Análisis Léxico	6
5.2. Tokenización	7
5.3. Eliminación de Stopwords	7
5.4. Stemming	7
6. Modelo de Recuperación	7
6.1. Modelo Vectorial TF-IDF	7
6.1.1. Cálculo de TF (Term Frequency)	7
6.1.2. Cálculo de IDF (Inverse Document Frequency)	7
6.1.3. Peso TF-IDF	8
6.2. Similitud del Coseno	8
6.3. Índice Invertido	9
7. API REST (Backend)	9
7.1. Endpoints Implementados	9
7.2. Ejemplo de Respuesta de Búsqueda	9

8. Interfaz de Usuario (Frontend)	10
8.1. Características	10
8.2. Capturas de Pantalla	10
9. Problemas Encontrados y Soluciones	15
9.1. Error de Multiprocessing en Windows	15
9.2. MemoryError durante Indexación	15
9.3. Alto Consumo de RAM al Cargar Índices	15
9.4. Mezcla de Idiomas en Resultados	15
10. Conclusiones	16
10.1. Logros del Proyecto	16
10.2. Estadísticas Finales	16
10.3. Lecciones Aprendidas	16
11. Referencias	16

1. Introducción

En esta práctica se desarrolla un **sistema completo de Recuperación de Información (RI)** que permite buscar en una colección de más de 10 GB de artículos de Wikipedia. El sistema está compuesto por:

- **Backend:** API REST desarrollada con FastAPI que implementa el pipeline completo de RI
- **Frontend:** Interfaz web desarrollada con React, TypeScript y TailwindCSS
- **Corpus:** Artículos de Wikipedia en castellano, catalán y portugués (+10 GB)

El modelo de recuperación utilizado es el **modelo vectorial con TF-IDF y similitud del coseno**, implementando un índice invertido persistente que permite búsquedas eficientes sobre millones de documentos.

1.1. Objetivos

1. Implementar un backend con FastAPI que procese documentos, construya el índice e implemente la búsqueda
2. Desarrollar un frontend en React para enviar consultas y visualizar resultados
3. Utilizar una colección documental real de más de 10 GB
4. Documentar el proceso, problemas encontrados y soluciones aplicadas

2. Entorno de trabajo

2.1. Backend (Python)

Se ha utilizado **Python 3.11** con Anaconda para gestionar el entorno virtual.

```
conda create -n P4_Final_RI python=3.11
conda activate P4_Final_RI
pip install -r backend/requirements.txt
```

Las principales librerías utilizadas son:

- **fastapi:** Framework web asíncrono para la API REST
- **uvicorn:** Servidor ASGI para ejecutar FastAPI
- **nltk:** Procesamiento de lenguaje natural (stopwords, stemming)
- **pydantic:** Validación de datos y modelos

2.2. Frontend (JavaScript/TypeScript)

Se utilizó **Node.js 18+** con el siguiente stack:

- **Vite**: Bundler moderno (más rápido que create-react-app)
- **React 18**: Framework de interfaz de usuario
- **TypeScript**: Tipado estático para JavaScript
- **TailwindCSS**: Framework CSS utility-first
- **Lucide React**: Iconos

2.3. Contenerización

El sistema puede ejecutarse con **Docker** y **Docker Compose** para facilitar el despliegue:

```
docker-compose up
```

3. Estructura del proyecto

La estructura de carpetas y archivos del proyecto es la siguiente:

```
P4_RI/
|-- backend/                               (API FastAPI)
|   |-- main.py                            (punto de entrada de la API)
|   |-- config.py                          (configuración centralizada)
|   |-- preprocessing.py                  (tokenización, stopwords, stemming)
|   |-- indexing.py                        (cálculo TF-IDF, índice invertido)
|   |-- build_index.py                   (indexación de castellano)
|   |-- build_index_ca.py                (indexación de catalán)
|   |-- build_index_pt.py                (indexación de portugués)
|   |-- wikipedia_loader.py            (carga de artículos JSON)
|   |-- persistent_index.py           (índice persistente en disco)
|   |-- requirements.txt
|-- frontend/                                (Interfaz React)
|   |-- src/
|   |   |-- App.tsx                      (componente principal)
|   |   |-- api.ts                       (cliente API)
|   |   |-- types.ts                     (tipos TypeScript)
|   |-- package.json
|-- datos/                                    (corpus de Wikipedia)
|   |-- extracted_es/                  (artículos castellano, ~5.8 GB)
|   |-- extracted_ca/                  (artículos catalán, ~1.9 GB)
|   |-- extracted_pt/                  (artículos portugués, ~2.8 GB)
|   |-- index/
|       |-- es/                         (índice castellano)
|       |-- ca/                         (índice catalán)
|       |-- pt/                          (índice portugués)
|-- doc_practica4/                           (documentación)
|-- Dockerfile                                (imagen Docker)
|-- docker-compose.yml                      (orquestación de servicios)
```

3.1. Descripción de Archivos del Backend

- `main.py`: API FastAPI con todos los endpoints de búsqueda y pipeline
- `preprocessing.py`: Funciones de tokenización, eliminación de stopwords y stemming
- `indexing.py`: Cálculo de TF, IDF, TF-IDF y construcción del índice invertido
- `build_index.py`: Script de indexación batch para castellano
- `build_index_ca.py`: Script de indexación batch para catalán
- `build_index_pt.py`: Script de indexación batch para portugués
- `wikipedia_loader.py`: Iterador sobre archivos JSON de WikiExtractor
- `persistent_index.py`: Clase para guardar/cargar el índice en disco

3.2. Ejecución del Sistema

3.2.1. Opción rápida: Docker

La forma más sencilla de ejecutar el sistema es mediante Docker. Desde el directorio raíz del proyecto:

```
docker compose up
```

Esto levantará automáticamente el backend y el frontend, dejando la plataforma lista para usar.

3.2.2. Opción manual

```
# 1. Instalar dependencias del backend
cd backend
pip install -r requirements.txt

# 2. Construir índices (por idioma, este paso ya está realizado)
python build_index.py      # Castellano (~3 horas)
python build_index_ca.py    # Catalán (~41 min)
python build_index_pt.py    # Portugués (~48 min)

# 3. Ejecutar backend
uvicorn main:app --reload

# 4. Instalar y ejecutar frontend
cd frontend
npm install
npm run dev
```

4. Colección Documental

4.1. Elección de la Fuente

Se eligió **Wikipedia** como fuente de datos por las siguientes razones:

- **Variedad temática:** Artículos de ciencia, historia, cultura, geografía, etc.
- **Experiencia realista:** Simula un buscador real que los usuarios conocen
- **Metadatos ricos:** Cada artículo incluye título, URL y contenido estructurado
- **Licencia libre:** CC BY-SA permite uso académico
- **Descarga sencilla:** Dumps oficiales disponibles en Wikimedia

4.2. Obtención de los Datos

Los datos se obtuvieron de los dumps oficiales de Wikimedia:

- Castellano: <https://dumps.wikimedia.org/eswiki/latest/>
- Catalán: <https://dumps.wikimedia.org/cawiki/latest/>
- Portugués: <https://dumps.wikimedia.org/ptwiki/latest/>

La extracción se realizó con **WikiExtractor**, que convierte el XML de Wikipedia a archivos JSON con una línea por artículo:

```
{"id": "12", "url": "https://es.wikipedia.org/wiki/...",  
"title": "Título", "text": "Contenido del artículo..."}
```

4.3. Estadísticas del Corpus

Idioma	Artículos	Términos únicos	Tiempo
Castellano	1,924,610	2,657,342	188.6 min
Catalán	743,723	1,574,860	41.1 min
Portugués	1,057,354	1,603,916	47.6 min
Total	3,725,687	-	4.6 horas

Cuadro 1: Estadísticas del corpus indexado.

El tamaño total de los datos extraídos es de **10.5 GB**, superando el requisito mínimo de 10 GB.

5. Pipeline de Preprocesamiento

El preprocesamiento de documentos y consultas sigue el siguiente pipeline:

5.1. Análisis Léxico

Normalización del texto: conversión a minúsculas y limpieza de caracteres especiales.

```
1 def normalize_text(text: str) -> str:  
2     text = text.lower()  
3     text = re.sub(r'^\w\s+', ' ', text)  
4     return text
```

Listing 1: Normalización de texto

5.2. Tokenización

Extracción de palabras mediante expresiones regulares:

```

1 def tokenize_text(text: str) -> list[str]:
2     text = normalize_text(text)
3     tokens = re.findall(r'\w+', text)
4     return tokens

```

Listing 2: Tokenización

5.3. Eliminación de Stopwords

Filtrado de palabras vacías usando NLTK. Se soportan tres idiomas:

```

1 def remove_stopwords_tokens(tokens, language="spanish"):
2     stop_words = set(stopwords.words(language))
3     return [t for t in tokens if t not in stop_words]

```

Listing 3: Eliminación de stopwords

5.4. Stemming

Reducción de palabras a su raíz usando el algoritmo Snowball:

```

1 def lemmatize_or_stem_tokens(tokens, language="spanish"):
2     stemmer = SnowballStemmer(language)
3     return [stemmer.stem(t) for t in tokens]

```

Listing 4: Stemming con Snowball

6. Modelo de Recuperación

6.1. Modelo Vectorial TF-IDF

El sistema utiliza el **modelo vectorial** con ponderación **TF-IDF** (Term Frequency - Inverse Document Frequency). Cada documento y consulta se representa como un vector en un espacio de términos.

6.1.1. Cálculo de TF (Term Frequency)

La frecuencia normalizada del término t en el documento d :

$$\text{TF}(t, d) = \frac{f(t, d)}{|d|} \quad (1)$$

Donde $f(t, d)$ es el número de ocurrencias del término y $|d|$ es el número total de tokens en el documento.

6.1.2. Cálculo de IDF (Inverse Document Frequency)

Mide la importancia del término en la colección. Se usa IDF suavizado:

$$\text{IDF}(t) = \log \frac{N + 1}{df_t + 1} + 1 \quad (2)$$

Donde N es el número total de documentos y df_t es el número de documentos que contienen el término.

6.1.3. Peso TF-IDF

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t) \quad (3)$$

```

1 # Calcular TF
2 tf = count / n_tokens
3
4 # Calcular IDF suavizado
5 idf[term] = log((doc_count + 1) / (df_t + 1)) + 1.0
6
7 # Peso TF-IDF
8 tfidf = tf * idf[term]

```

Listing 5: Cálculo de TF-IDF en el índice

6.2. Similitud del Coseno

Para calcular la relevancia de un documento respecto a una consulta, se utiliza la **similitud del coseno**:

$$\text{sim}(q, d) = \frac{\vec{q} \cdot \vec{d}}{\|\vec{q}\| \times \|\vec{d}\|} = \frac{\sum_{i=1}^n q_i \times d_i}{\sqrt{\sum_{i=1}^n q_i^2} \times \sqrt{\sum_{i=1}^n d_i^2}} \quad (4)$$

Las normas de los documentos se precisan durante la indexación para acelerar las búsquedas.

```

1 def search(self, query: str, top_k: int = 10):
2     # Preprocesar consulta
3     query_tokens = preprocess_document(query, self.language)
4
5     # Calcular vector TF-IDF de la consulta
6     query_vec = {}
7     for term, count in Counter(query_tokens).items():
8         if term in self.idf:
9             tf = count / len(query_tokens)
10            query_vec[term] = tf * self.idf[term]
11
12     # Calcular similitud con cada documento
13     scores = defaultdict(float)
14     query_norm = sqrt(sum(w*w for w in query_vec.values()))
15
16     for term, weight in query_vec.items():
17         if term in self.inverted_index:
18             for doc_id, doc_weight in self.inverted_index[term]:
19                 scores[doc_id] += weight * doc_weight
20
21     # Normalizar por normas de documentos
22     for doc_id in scores:
23         scores[doc_id] /= (query_norm * self.doc_norms[doc_id])
24
25     # Ordenar por score descendente
26     return sorted(scores.items(), key=lambda x: x[1], reverse=True)[:top_k]

```

Listing 6: Cálculo de similitud coseno

6.3. Índice Invertido

El índice invertido es la estructura de datos principal que permite búsquedas eficientes. Para cada término, almacena una lista de documentos que lo contienen junto con su peso TF-IDF:

```
{
  "madrid": [("doc_123", 0.045), ("doc_456", 0.032), ...],
  "España": [("doc_789", 0.067), ("doc_123", 0.041), ...],
  ...
}
```

Los postings están ordenados por peso descendente para optimizar las búsquedas.

7. API REST (Backend)

7.1. Endpoints Implementados

La API FastAPI expone los siguientes endpoints:

Método	Endpoint	Descripción
GET	/	Información de la API
GET	/stats	Estadísticas del índice cargado
GET	/search?q={query}	Búsqueda de documentos
GET	/document/{id}	Obtener documento por ID
POST	/lexical_analysis	Normalización de texto
POST	/tokenize	Tokenización
POST	/remove_stopwords	Eliminación de stopwords
POST	/lemmatize	Stemming/Lematización
POST	/analyze_query	Pipeline completo de análisis

Cuadro 2: Endpoints de la API REST.

7.2. Ejemplo de Respuesta de Búsqueda

GET /search?q=historia+de+España&lang=es&top_k=5

```
{
  "query": "historia de España",
  "language": "es",
  "results": [
    {
      "id": "12345",
      "title": "Historia de España",
      "url": "https://es.wikipedia.org/wiki/Historia_de_España",
      "score": 0.847,
      "snippet": "La historia de España abarca el período..."
    },
    ...
  ],
  "total_results": 5,
  "search_time_ms": 234
}
```

8. Interfaz de Usuario (Frontend)

El frontend está desarrollado con **React** y **TypeScript**, utilizando **TailwindCSS** para los estilos.

8.1. Características

- **Selector de idioma:** Permite elegir entre castellano, catalán y portugués
- **Barra de búsqueda:** Campo de texto con botón de búsqueda
- **Indicador de carga:** Spinner mientras se realiza la búsqueda
- **Lista de resultados:** Muestra los 20 primeros resultados relevantes con título, score, snippet y enlace a Wikipedia
- **Estadísticas dinámicas:** Los indicadores de número de documentos, vocabulario y estado del índice se actualizan automáticamente después de cada consulta en cada idioma
- **Gestión de errores:** Mensajes cuando el backend no está disponible

8.2. Capturas de Pantalla

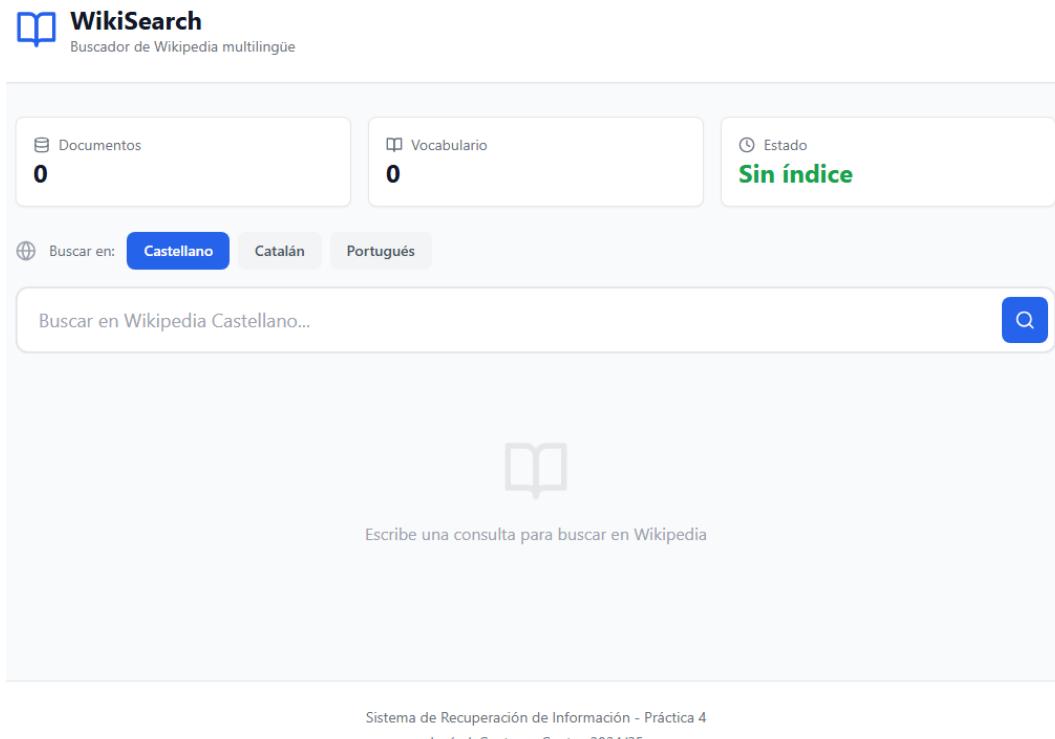


Figura 1: Interfaz principal del buscador WikiSearch.

The screenshot shows the WikiSearch search interface. At the top, there are three summary boxes: 'Documentos 1.924.610', 'Vocabulario 2.657.342', and 'Estado Activo'. Below these are language selection buttons for 'Castellano' (highlighted in blue), 'Catalán', and 'Portugués'. A search bar contains the query 'Cervantes Quijote'. Below the search bar, it says '7264 resultados encontrados en 116931.59 ms'. The results are listed in three cards:

- #1 Score: 0.7580 Cervantismo**
El cervantismo es el estudio de la obra y figura de Miguel de Cervantes, autor de "Don Quijote de la Mancha". Historia. El escritor José María de Pereda afirmaba que: Hay quienes proponen que el origen del cervantismo habría que situarlo en la "Vida de Cervantes" que Mayans redactó por encargo de Lo...
- #2 Score: 0.7192 Don Quijote de la Mancha**
Don Quijote de la Mancha es una novela escrita por el español Miguel de Cervantes Saavedra. Publicada su primera parte con el título de El ingenioso hidalgo don Quijote de la Mancha a comienzos de 1605, es la obra más destacada de la literatura española y una de las principales de la literatura univ...
- #3 Score: 0.6532 Miguel de Cervantes**
Miguel de Cervantes Saavedra (Alcalá de Henares, 29 de septiembre de 1547-Madrid, 22 de abril de 1616) fue un novelista, poeta, dramaturgo y...

Figura 2: Primera búsqueda en castellano (carga inicial del índice).

The screenshot shows the WikiSearch search interface. At the top, there are three summary boxes: 'Documentos 1.924.610', 'Vocabulario 2.657.342', and 'Estado Activo'. Below these are language selection buttons for 'Castellano' (highlighted in blue), 'Catalán', and 'Portugués'. A search bar contains the query 'Ceuta'. Below the search bar, it says '2377 resultados encontrados en 313.27 ms'. The results are listed in three cards:

- #1 Score: 0.8360 Ceuta**
Ceuta es una ciudad autónoma española, situada en la península ibérica, en la orilla africana del estrecho de Gibraltar, en su lado oriental. Está bañada por las aguas del mar Mediterráneo, mientras que al oeste y suroeste limita con Marruecos. Su población es de . Con una extensión superficial de...
- #2 Score: 0.7858 Francisco Olivencia Ruiz**
Francisco Olivencia Ruiz (Ceuta, -ibidem,) fue un político y parlamentario español, diputado en Cortes Generales durante la I Legislatura (gobiernos de Adolfo Suárez y Leopoldo Calvo Sotelo) y senador por Ceuta en las Legislaturas V y VI. Abogado de profesión, fue considerado un firme defensor de...
- #3 Score: 0.7849 Nicolás Fernández Cucurull**
Nicolás Fernández Cucurull (Ceuta, 9 de julio de 1963) es un político español. Biografía. Nacido en Ceuta en 1963, es licenciado en Derecho y en...

Figura 3: Segunda búsqueda en castellano con índice ya cargado (313.27 ms).

The screenshot shows the WikiSearch search interface. At the top, there are three summary boxes: 'Documentos 743.723', 'Vocabulario 1.574.860', and 'Estado Activo'. Below this, a search bar contains the query 'Carles Puigdemont'. Underneath the search bar, it says '10.240 resultados encontrados en 139713.98 ms'. The results are listed in cards:

- #1 Score: 0.5481**
Carles Puigdemont i Casamajó
és un periodista i polític català, 130è president de la Generalitat de Catalunya, i president de Junts per Catalunya des d'octubre de 2024. Biografia. Es va iniciar al periodisme en diversos mitjans comarcals. Més tard va esdevenir director de l'Agència Catalana de Notícies, així com del setmanari "...".
- #2 Score: 0.3745**
Llista d'estats visitats pels membres del Govern de Catalunya a l'exili
Poc després que el fiscal general de l'Estat espanyol, José Manuel Maza, anunciés el 30 d'octubre la presentació d'una querella davant l'Audiència Nacional per rebel·lió, sedició i malversació Carles Puigdemont, president de la Generalitat, i la resta del Govern de la república proclamada al Parlament...
- #3 Score: 0.3642**
Detenció de Carles Puigdemont a Sardenya
El 23 de setembre de 2021, el president de la Generalitat de Catalunya a l'exili i actual eurodiputat català Carles Puigdemont fou detingut a...

Figura 4: Primera búsqueda en catalán (carga inicial del índice).

The screenshot shows the WikiSearch search interface. At the top, there are three summary boxes: 'Documentos 743.723', 'Vocabulario 1.574.860', and 'Estado Activo'. Below this, a search bar contains the query 'emple Expiatori de la Sagrada Família'. Underneath the search bar, it says '13.780 resultados encontrados en 22.53 ms'. The results are listed in cards:

- #1 Score: 0.3793**
Jordi Faulí i Oller
és un arquitecte català, titulat el 1992. És l'actual arquitecte director i coordinador del Temple Expiatori de la Sagrada Família. En la seva joventut va ser membre dels equips directius de Minyons Escoltes i Guies de Catalunya. El 1990 es va incorporar a les obres del temple dintre de l'equip diri...
- #2 Score: 0.3188**
Josep Manyanet i Vives
Josep Manyanet i Vives (Trem普, Pallars Jussà, 7 de gener del 1833 - Sant Andreu de Palomar, Barcelona, Barcelonès, 17 de desembre del 1901) fou un prevere, fundador dels Fills de la Sagrada Família i les Missioneres Filles de la Sagrada Família de Natzaret, i apòstol de la devoció a la Sagrada Famili...
- #3 Score: 0.3106**
Junta Constructora del Temple Expiatori de la Sagrada Família
La Junta Constructora del Temple Expiatori de la Sagrada Família és una entitat creada el 1895 per a promoure l'obra i la figura d'Antoni Gaudí a...

Figura 5: Segunda búsqueda en catalán con índice ya cargado (22.53 ms).

The screenshot shows the WikiSearch search interface. At the top, there are three summary boxes: 'Documentos 1.057.354', 'Vocabulario 1.603.916', and 'Estado Activo'. Below these are language selection buttons: 'Buscar en: Castellano', 'Catalán', and 'Portugués' (which is highlighted). A search bar contains the query 'Saudade'. The results page indicates 2307 results found in 76086.27 ms. The first result is '#1 Score: 0.6323 Saudade', with a snippet describing it as a Portuguese word for longing. The second result is '#2 Score: 0.5966 Saudades de casa', with a snippet about homesickness. The third result is '#3 Score: 0.5182 Saudades (Santa Catarina)', with a snippet about a municipality in Brazil.

Figura 6: Primera búsqueda en portugués (carga inicial del índice).

The screenshot shows the WikiSearch search interface. At the top, there are three summary boxes: 'Documentos 1.057.354', 'Vocabulario 1.603.916', and 'Estado Activo'. Below these are language selection buttons: 'Buscar en: Castellano', 'Catalán', and 'Portugués' (which is highlighted). A search bar contains the query 'Cristóbal Colón'. The results page indicates 4836 results found in 16.46 ms. The first result is '#1 Score: 0.6154 Cristóbal Colón (cruzador)', with a snippet about a Spanish ship. The second result is '#2 Score: 0.5471 San Cristóbal (municipio da Venezuela)', with a snippet about a Venezuelan town. The third result is '#3 Score: 0.4890 San Cristóbal (distrito de Lucanas)', with a snippet about a Peruvian district.

Figura 7: Segunda búsqueda en portugués con índice ya cargado (16.46 ms).

Las figuras anteriores demuestran la diferencia de rendimiento entre la primera búsqueda (que incluye la carga del índice desde disco) y las búsquedas posteriores (que utilizan el índice ya cargado en memoria). Los tiempos de respuesta se reducen drásticamente: de varios segundos en la carga inicial a apenas **313 ms** en castellano, **22 ms** en catalán y **16 ms** en portugués. Esta mejora evidencia la eficiencia del índice invertido en memoria para búsquedas en tiempo real.

```
(base) PS E:\P4_RI\backend> conda activate P4_Final_RI
(P4_Final_RI) PS E:\P4_RI\backend> uvicorn main:app --reload
INFO: Will watch for changes in these directories: ['E:\\\\P4_RI\\\\backend']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [21400] using StatReload
INFO: Started server process [21580]
INFO: Waiting for application startup.
Indices disponibles: es, ca, pt
Los indices se cargan bajo demanda en la primera busqueda.
INFO: Application startup complete.
```

Figura 8: Consola del backend FastAPI en ejecución.

```
(base) PS E:\P4_RI\frontend> conda activate P4_Final_RI
(P4_Final_RI) PS E:\P4_RI\frontend> npm run dev
> wikisearch-frontend@1.0.0 dev
> vite

The CJS build of Vite's Node API is deprecated. See https://vite.dev/guide/troubleshooting.html#vite-cjs-node-api-deprecated-for-more-details.
Re-optimizing dependencies because vite config has changed

VITE v5.4.21 ready in 566 ms

→ Local: http://localhost:3000/
→ Network: use --host to expose
→ press h + enter to show help
```

Figura 9: Consola del frontend Vite/React en ejecución.

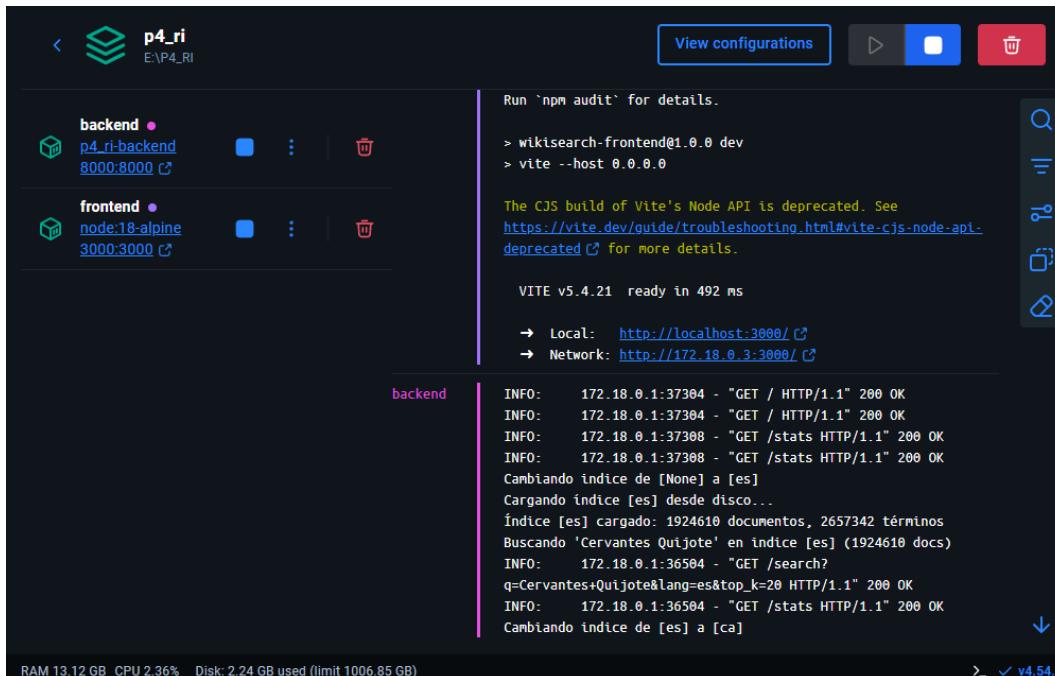


Figura 10: Backend y Frontend ejecutado en Docker.

9. Problemas Encontrados y Soluciones

Durante el desarrollo del sistema se encontraron varios problemas técnicos que requirieron soluciones específicas.

9.1. Error de Multiprocessing en Windows

Problema: WikiExtractor fallaba con el error `ValueError: cannot find context for 'fork'`.

Causa: WikiExtractor usa `fork` para multiprocessing, que solo existe en Linux/Mac.

Solución: Usar WSL (Windows Subsystem for Linux) para la extracción con 4 procesos paralelos.

9.2. MemoryError durante Indexación

Problema: El script de indexación fallaba con `MemoryError` al procesar más de 3 millones de documentos.

Causa: El índice invertido completo no cabía en RAM (20-40 GB necesarios).

Solución: Dividir la indexación en scripts separados por idioma, cada uno guardando en su propio directorio (`index/es/`, `index/ca/`, `index/pt/`).

9.3. Alto Consumo de RAM al Cargar Índices

Problema: Cargar el índice de castellano consume 18 GB de RAM en pico.

Causa: El archivo del índice invertido (.pkl) contiene 2.6 millones de términos con millones de postings.

Solución: Los índices se cargan bajo demanda (no al iniciar el servidor). Para Docker, se recomienda usar los índices de catalán o portugués, que son más ligeros.

9.4. Mezcla de Idiomas en Resultados

Problema: Búsquedas en castellano devolvían resultados en portugués y catalán.

Causa: El índice original fue construido antes de separar los scripts por idioma, mezclando documentos.

Solución: Reconstruir el índice de castellano usando solo `extracted.es`.

10. Conclusiones

10.1. Logros del Proyecto

Se ha desarrollado un sistema completo de Recuperación de Información que cumple con todos los requisitos de la práctica:

- **Backend funcional:** API REST con FastAPI que implementa el pipeline completo de RI
- **Frontend moderno:** Interfaz web con React, TypeScript y TailwindCSS
- **Corpus significativo:** Más de 10 GB de artículos de Wikipedia en 3 idiomas
- **Índice eficiente:** Índice invertido con TF-IDF y similitud coseno
- **Despliegue con Docker:** Sistema containerizado para fácil despliegue

10.2. Estadísticas Finales

Métrica	Valor
Total de artículos indexados	3,725,687
Tamaño del corpus	10.5 GB
Tiempo total de indexación	4.6 horas
Idiomas soportados	3 (ES, CA, PT)
Endpoints de la API	9

Cuadro 3: Estadísticas finales del sistema.

10.3. Lecciones Aprendidas

- **Escalabilidad:** Trabajar con millones de documentos requiere estrategias de memoria cuidadosas
- **Separación por idiomas:** Mantener índices separados simplifica el mantenimiento y reduce el consumo de RAM
- **WSL para Windows:** Herramientas de Linux como WikiExtractor funcionan mejor en WSL que en Windows nativo
- **Docker:** Facilita el despliegue pero tiene limitaciones de memoria que afectan a índices grandes

11. Referencias

- Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.
- FastAPI Documentation: <https://fastapi.tiangolo.com/>
- React Documentation: <https://react.dev/>
- Wikimedia Downloads: <https://dumps.wikimedia.org/>
- NLTK Documentation: <https://www.nltk.org/>