Search Engine

Erick Gonzalez Parada ID: 178145
Emiliano Ruiz Plancarte ID: 177478
Andre Francois Duhamel Gutierrez ID: 177315
Antonio Gutiérrez Blanco ID: 177442

May 5, 2025

Abstract

This document explores the design and implementation of a full-stack search engine using multiple programming languages and technologies. The objective was to extract information from selected web pages, process the textual data, and store it efficiently in a distributed SQL database system (CockroachDB). The processed data was analyzed using matrix operations and optimized through dimensionality reduction techniques such as Singular Value Decomposition (SVD). A backend API and graphical interface were also implemented to allow real-time query searches and display of ranked results. This multidisciplinary project combines web scraping, natural language processing (NLP), database engineering, and human-computer interaction, providing a comprehensive overview of modern data pipeline architecture in a search context [1, 4].

Keywords: Database Design, UML, Rust ,CockroachDB.

1 Objective

The objective of this project is to develop a modular search engine capable of parsing, processing, indexing, and retrieving relevant information from a set of ten online documents. This includes implementing a web scraper, performing text preprocessing such as stemming, designing a relational database schema, applying optimization algorithms, and deploying an interactive frontend for end users. The primary goal was to gain hands-on experience with full-stack data engineering and retrieval systems.

2 Theoretical Framework

Search engines operate through a series of phases: data collection (crawling), information extraction, indexing, and retrieval. Web scraping is a technique to automatically collect information from websites, often used when APIs are unavailable. In this project, data extraction is followed by Natural Language Processing (NLP) techniques such as tokenization and stemming to normalize terms.

Stemming reduces words to their root forms, increasing the likelihood of matches between query terms and document content. This project uses a Java-based stemmer to process JSON data scraped from target websites [4].

CockroachDB, a distributed SQL database built for cloud scalability and resilience, serves as the storage backend. Its distributed nature ensures fault tolerance and horizontal scalability, which is crucial for large-scale search systems [1].

To optimize document comparison, we employed Singular Value Decomposition (SVD), a linear algebra technique that reduces the dimensionality of term-document matrices, enabling latent semantic indexing (LSI). This allows the retrieval system to capture conceptual similarity even if query terms differ from those in the documents.

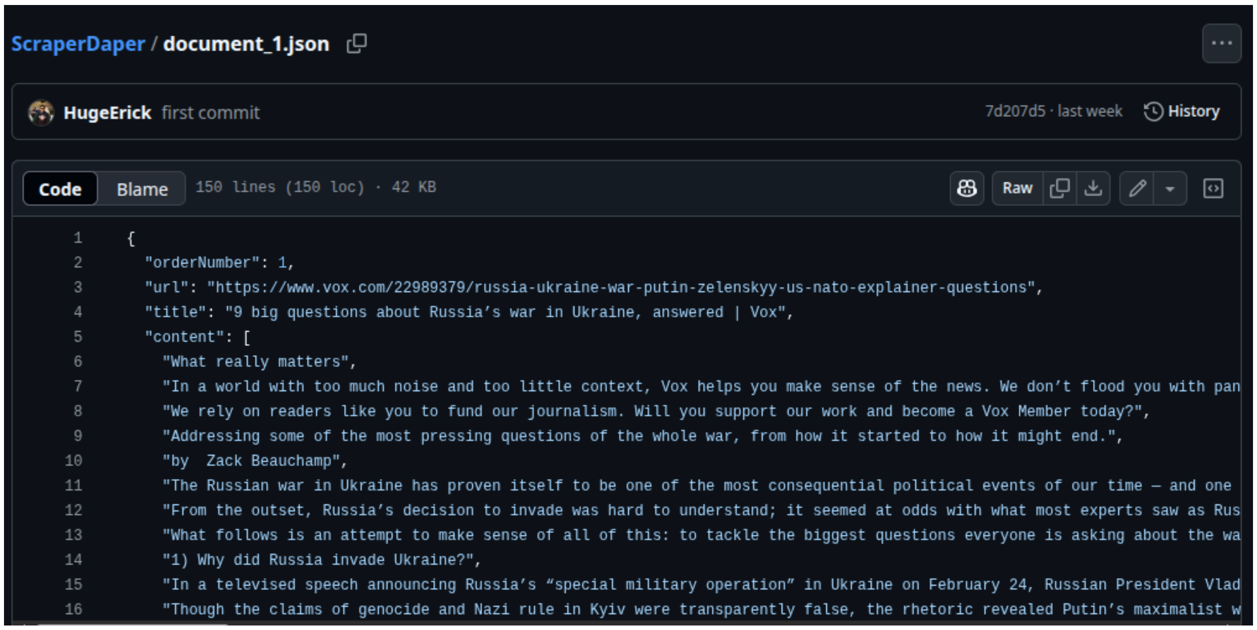
Finally, the user interface and API components were developed to allow for real-time querying and display of results. The backend provides endpoints for querying the database, while the frontend renders results in a structured, user-friendly format.

3 Implementations (Solutions)

3.1 Web Scrapper

As a team we coded a javascript web scraper called ”ScraperDaper” that for the url’s processed it will give them an order number, url, title and content JSON sections.

`https://github.com/HugeErick/ScraperDaper`



```
{
  "orderNumber": 1,
  "url": "https://www.vox.com/22989379/russia-ukraine-war-putin-zelenskyy-us-nato-explainer-questions",
  "title": "9 big questions about Russia’s war in Ukraine, answered | Vox",
  "content": [
    "What really matters",
    "In a world with too much noise and too little context, Vox helps you make sense of the news. We don’t flood you with pan
    "We rely on readers like you to fund our journalism. Will you support our work and become a Vox Member today?",
    "Addressing some of the most pressing questions of the whole war, from how it started to how it might end.",
    "by  Zack Beauchamp",
    "The Russian war in Ukraine has proven itself to be one of the most consequential political events of our time — and one
    "From the outset, Russia’s decision to invade was hard to understand; it seemed at odds with what most experts saw as Rus
    "What follows is an attempt to make sense of all of this: to tackle the biggest questions everyone is asking about the wa
    "1) Why did Russia invade Ukraine?",
    "In a televised speech announcing Russia’s “special military operation” in Ukraine on February 24, Russian President Vlad
    "Though the claims of genocide and Nazi rule in Kyiv were transparently false, the rhetoric revealed Putin’s maximalist w
```

Figure 1: JSON sample used from the scraper

3.2 Stemmer

This is also a project we did, coded in Java in order to process text to get stemmed and meaningfull terms one of the integrations was a JSON integration so we can further process the JSONS of the scraper.

`https://github.com/HugeErick/Stemmer`

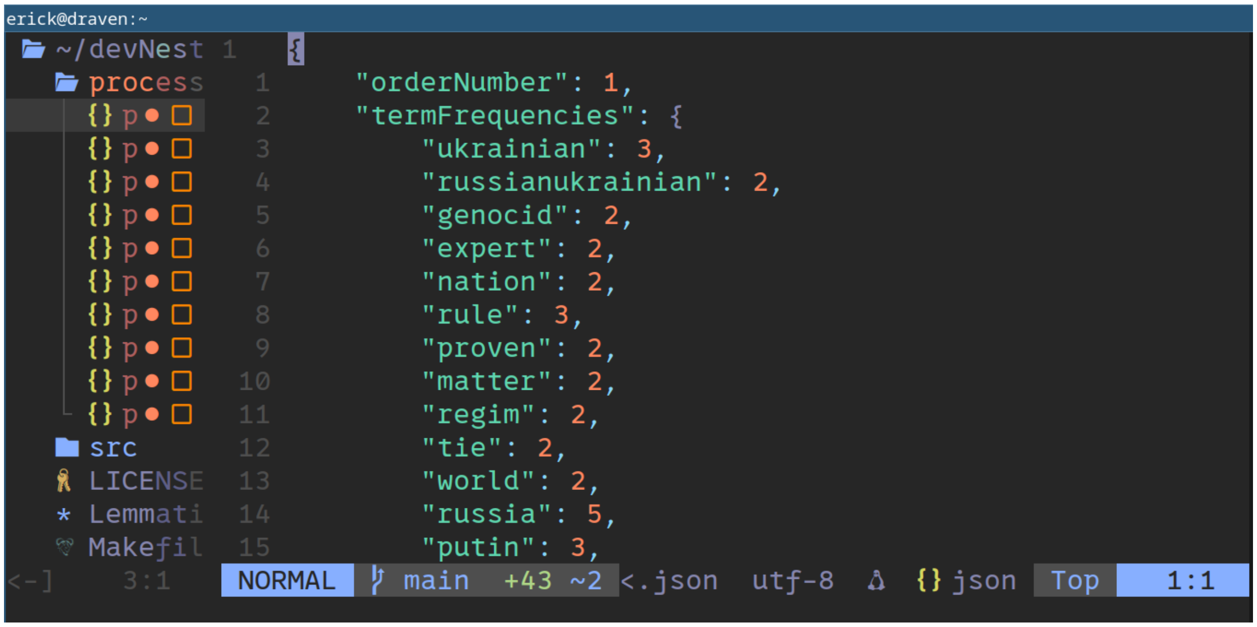

```
erick@draven:~
~/devNest          {
  process            "orderNumber": 1,
    {} p             "termFrequencies": {
    {} p                 "ukrainian": 3,
    {} p                 "russianukrainian": 2,
    {} p                 "genocid": 2,
    {} p                 "expert": 2,
    {} p                 "nation": 2,
    {} p                 "rule": 3,
    {} p                 "proven": 2,
    {} p                 "matter": 2,
    {} p                 "regim": 2,
  src                    "tie": 2,
  LICENSE                "world": 2,
  Lemmati                "russia": 5,
  Makefil                "putin": 3,
<-]   3:1  NORMAL  main  +43 ~2 <.json  utf-8  {} json  Top  1:1
```

Figure 2: JSON sample used from the stemmer

3.3 Database Design

The database schema was designed using an Entity-Relationship Model and then implemented in CockroachDB. The schema includes tables for documents, terms, and associations such as ‘has(term_id, document_id, frequency)‘.

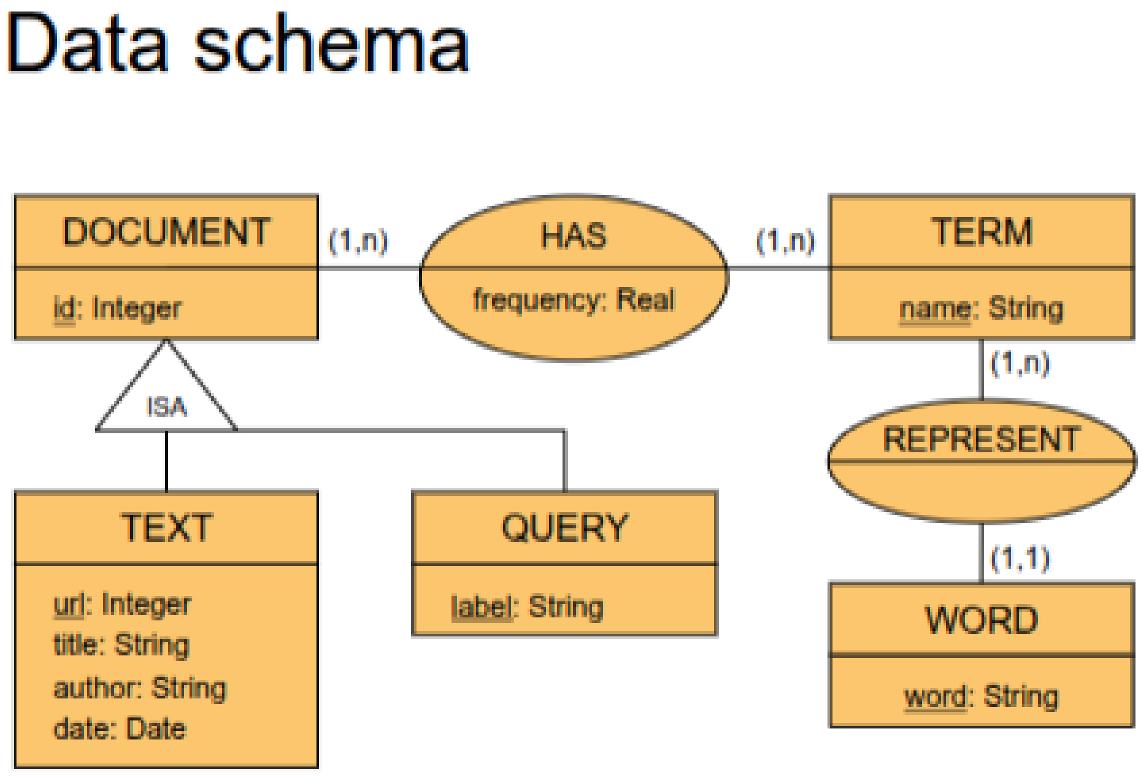


Figure 3: Staring UML Diagram

3.4 Database implementation and analysis

3.4.1 Creating of tables

The database schema was implemented using the following SQL script:

Listing 1: SQL Script for Creating Tables

```
-- Create the DOCUMENT table
CREATE TABLE IF NOT EXISTS DOCUMENT (
    id SERIAL PRIMARY KEY,
    url TEXT NOT NULL,
    title TEXT NOT NULL,
    author TEXT,
    date TIMESTAMP
);

-- Create the TERM table
CREATE TABLE IF NOT EXISTS TERM (
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL UNIQUE
);

-- Create the WORD table
CREATE TABLE IF NOT EXISTS WORD (
    id SERIAL PRIMARY KEY,
    word TEXT NOT NULL UNIQUE
);

-- Create the HAS relationship table (many-to-many between DOCUMENT and TERM)
CREATE TABLE IF NOT EXISTS HAS (
    document_id INTEGER REFERENCES DOCUMENT(id),
    term_id INTEGER REFERENCES TERM(id),
    frequency REAL NOT NULL,
   PRIMARY KEY (document_id, term_id)
);

-- Create the REPRESENTS relationship table (many-to-many between TERM and WORD)
CREATE TABLE IF NOT EXISTS REPRESENTS (
    term_id INTEGER REFERENCES TERM(id),
    word_id INTEGER REFERENCES WORD(id),
   PRIMARY KEY (term_id, word_id)
);

-- Create the SYNONYMS table
CREATE TABLE IF NOT EXISTS SYNONYMS (
    term_id INT8 NOT NULL REFERENCES TERM(id),
    synonym_name VARCHAR(255) NOT NULL,
   PRIMARY KEY (term_id, synonym_name)
);
```

3.4.2 Inserting data

Some examples of inserting data into the database:

Listing 2: Inserting Terms

```
INSERT INTO term (name) VALUES
('asamblea_general'), ('consejo_seguridad'), ('resolucion'), ('ucrania'), ('rusia'), ('esta
('union_europea'), ('invasion'), ('conflicto'), ('soberania'), ('integridad_territorial'),
('carta_naciones_unidas'), ('paz_duradera'), ('fuerzas_militares'), ('fronteras_internacion
('voto_abstencion'), ('enmiendas'), ('embajador'), ('derechos_humanos'), ('ayuda_humanitar
('ninos_afectados'), ('refugio'), ('escuelas_danadas'), ('sistema_energetico'), ('calefacci
('muertos'), ('heridos'), ('crisis_humanitaria'), ('confrontacion_nuclear'), ('orden_inter
('mito_capacidades_militares'), ('guerra_civil'), ('retirada_tropas'), ('camino_paz'), ('v
('dorothy_shea'), ('vassily_nebenzia'), ('barbara_woodward'), ('donald_trump'), ('kiev'),
('naciones_unidas'), ('oficina_coordinacion_asuntos_humanitarios'), ('ocha'), ('resolucion
('principios_carta_onu'), ('seguridad_internacional'), ('espectro_destruccion_total'), ('c
('propuestas_enmiendas'), ('apoyo_internacional'), ('abstenciones'), ('votos_favor'), ('vo
ON CONFLICT (name) DO NOTHING;
```

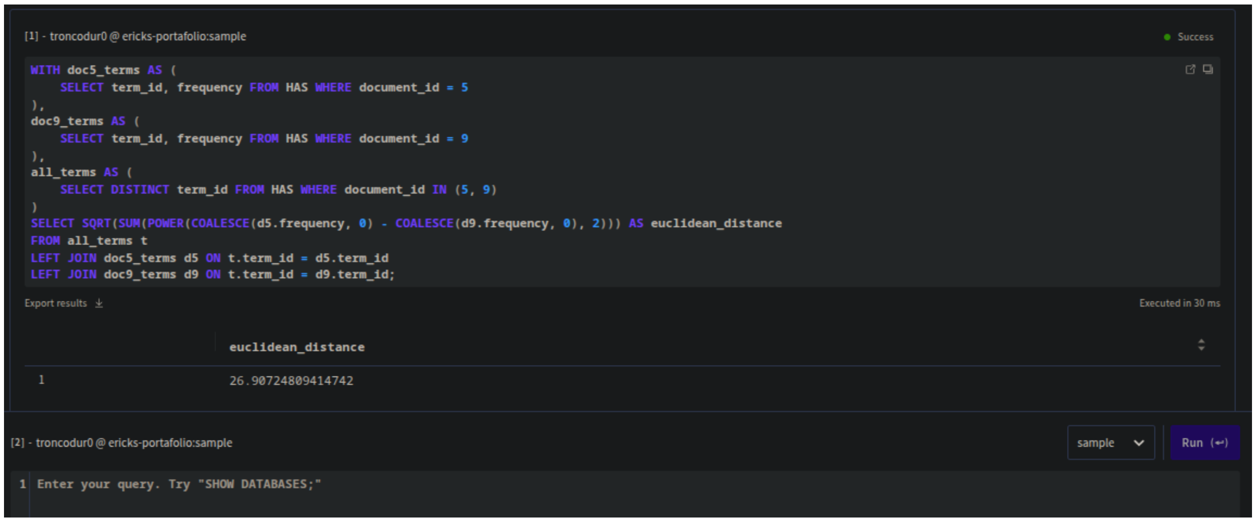
Listing 3: Inserting Synonyms

```
INSERT INTO synonyms (term_id, synonym_name)
VALUES
-- Asamblea General
((SELECT id FROM term WHERE name = 'asamblea_general'), 'organismo_onu'),
-- Consejo de Seguridad
((SELECT id FROM term WHERE name = 'consejo_seguridad'), 'organo_seguridad_onu'),
-- Resolucion
((SELECT id FROM term WHERE name = 'resolucion'), 'declaracion_oficial'),
-- Ucrania
((SELECT id FROM term WHERE name = 'ucrania'), 'ukraine'),
-- Rusia
((SELECT id FROM term WHERE name = 'rusia'), 'russia'),
-- Estados Unidos
((SELECT id FROM term WHERE name = 'estados_unidos'), 'eeuu'),
-- Union Europea
((SELECT id FROM term WHERE name = 'union_europea'), 'ue'),
-- Invasion
((SELECT id FROM term WHERE name = 'invasion'), 'ataque_militar'),
-- Conflicto
((SELECT id FROM term WHERE name = 'conflicto'), 'guerra'),
-- Soberania
((SELECT id FROM term WHERE name = 'soberania'), 'autonomia_estatal'),
-- Integridad Territorial
((SELECT id FROM term WHERE name = 'integridad_territorial'), 'unidad_nacional'),
-- Hostilidades
((SELECT id FROM term WHERE name = 'hostilidades'), 'combate_armado'),
-- Carta Naciones Unidas
((SELECT id FROM term WHERE name = 'carta_naciones_unidas'), 'fundamentos_onu'),
-- Paz Duradera
((SELECT id FROM term WHERE name = 'paz_duradera'), 'cese_hostilidades'),
-- Fuerzas Militares
((SELECT id FROM term WHERE name = 'fuerzas_militares'), 'tropas_combate'),
-- Fronteras Internacionalmente Reconocidas
((SELECT id FROM term WHERE name = 'fronteras_internacionalmente_reconocidas'), 'limites_le
-- Voto Abstencion
((SELECT id FROM term WHERE name = 'voto_abstencion'), 'neutralidad_votacion'),
-- Enmiendas
((SELECT id FROM term WHERE name = 'enmiendas'), 'modificaciones_propuesta'),
-- Embajador
((SELECT id FROM term WHERE name = 'embajador'), 'representante_diplomatico'),
-- Derechos Humanos
((SELECT id FROM term WHERE name = 'derechos_humanos'), 'proteccion_ciudadanos'),
-- Ayuda Humanitaria
((SELECT id FROM term WHERE name = 'ayuda_humanitaria'), 'asistencia_emergencia'),
-- Victimas Civiles
((SELECT id FROM term WHERE name = 'victimas_civiles'), 'danos_poblacion_no_combatiente'),
-- Ninos Afectados
((SELECT id FROM term WHERE name = 'ninos_afectados'), 'infancia_impactada'),
-- Refugio
((SELECT id FROM term WHERE name = 'refugio'), 'alojamiento_temporal'),
-- Escuelas Danadas
((SELECT id FROM term WHERE name = 'escuelas_danadas'), 'infraestructura_educativa_destruid
-- Sistema Energetico
((SELECT id FROM term WHERE name = 'sistema_energetico'), 'red_electrica'),
-- Calefaccion
((SELECT id FROM term WHERE name = 'calefaccion'), 'suministro_calor'),
-- Electricidad
((SELECT id FROM term WHERE name = 'electricidad'), 'energia_electrica'),
-- Muertos
((SELECT id FROM term WHERE name = 'muertos'), 'caidos'),
-- Heridos
((SELECT id FROM term WHERE name = 'heridos'), 'lesionados'),
-- Crisis Humanitaria
((SELECT id FROM term WHERE name = 'crisis_humanitaria'), 'emergencia_social'),
-- Confrontacion Nuclear
((SELECT id FROM term WHERE name = 'confrontacion_nuclear'), 'amenaza_atomic'),
-- Orden Internacional
((SELECT id FROM term WHERE name = 'orden_internacional'), 'normas_globales'),
-- Solidaridad
((SELECT id FROM term WHERE name = 'solidaridad'), 'apoyo_mutuo'),
-- Mito Capacidades Militares
((SELECT id FROM term WHERE name = 'mito_capacidades_militares'), 'falacia_potencia_rusa')
-- Guerra Civil
((SELECT id FROM term WHERE name = 'guerra_civil'), 'conflicto_interno'),
-- Retirada Tropas
((SELECT id FROM term WHERE name = 'retirada_tropas'), 'desmovilizacion_militar'),
-- Camino Paz
((SELECT id FROM term WHERE name = 'camino_paz'), 'proceso_negociacion'),
-- Vetos
((SELECT id FROM term WHERE name = 'vetos'), 'bloqueo_decisiones'),
-- Mariana Betsa
((SELECT id FROM term WHERE name = 'mariana_betsa'), 'diplomatica_ucraniana'),
-- Dorothy Shea
((SELECT id FROM term WHERE name = 'dorothy_shea'), 'representante_usa'),
-- Vassily Nebenzia
((SELECT id FROM term WHERE name = 'vassily_nebenzia'), 'representante_ruso'),
-- Barbara Woodward
((SELECT id FROM term WHERE name = 'barbara_woodward'), 'representante_reino_unido'),
-- Donald Trump
((SELECT id FROM term WHERE name = 'donald_trump'), 'presidente_usa'),
-- Kiev
((SELECT id FROM term WHERE name = 'kiev'), 'kyiv'),
-- Moscu
((SELECT id FROM term WHERE name = 'moscu'), 'moscow'),
-- ONU
((SELECT id FROM term WHERE name = 'onu'), 'naciones_unidas'),
-- Oficina Coordinacion Asuntos Humanitarios
((SELECT id FROM term WHERE name = 'oficina_coordinacion_asuntos_humanitarios'), 'ocha'),
-- Resolucion Pacifica Disputas
((SELECT id FROM term WHERE name = 'resolucion_pacifica_disputas'), 'negociacion_conflictos
-- Principios Carta ONU
((SELECT id FROM term WHERE name = 'principios_carta_onu'), 'fundamentos_internacionales')
-- Seguridad Internacional
((SELECT id FROM term WHERE name = 'seguridad_internacional'), 'proteccion_global'),
-- Espectro Destruccion Total
((SELECT id FROM term WHERE name = 'espectro_destruccion_total'), 'riesgo_extincion_masiva
-- Causa Profunda Conflicto
((SELECT id FROM term WHERE name = 'causa_profunda_conflicto'), 'motivaciones_politicas'),
-- Propuestas Enmiendas
((SELECT id FROM term WHERE name = 'propuestas_enmiendas'), 'modificaciones_texto'),
-- Apoyo Internacional
((SELECT id FROM term WHERE name = 'apoyo_internacional'), 'alianzas_globales'),
-- Abstenciones
((SELECT id FROM term WHERE name = 'abstenciones'), 'neutralidad_miembros'),
-- Votos Favor
((SELECT id FROM term WHERE name = 'votos_favor'), 'aprobacion_propuesta'),
-- Votos Contra
((SELECT id FROM term WHERE name = 'votos_contra'), 'rechazo_propuesta')
ON CONFLICT (term_id, synonym_name) DO NOTHING;
```

Listing 4: Inserting Frequencies

```
INSERT INTO has (document_id, term_id, frequency)
SELECT
  10,
  id,
  CASE
    WHEN name IN ('asamblea_general', 'consejo_seguridad', 'resolucion', 'ucrania', 'rusia
    WHEN name IN ('union_europea', 'invasion', 'conflicto', 'soberania', 'integridad_terri
     ELSE 2
  END
FROM term
WHERE name IN (
  'asamblea_general', 'consejo_seguridad', 'resolucion', 'ucrania', 'rusia', 'estados_unid
  'union_europea', 'invasion', 'conflicto', 'soberania', 'integridad_territorial', 'hostili
  'carta_naciones_unidas', 'paz_duradera', 'fuerzas_militares', 'fronteras_internacionalme
  'voto_abstencion', 'enmiendas', 'embajador', 'derechos_humanos', 'ayuda_humanitaria', 'v
  'ninos_afectados', 'refugio', 'escuelas_danadas', 'sistema_energetico', 'calefaccion', '
  'muertos', 'heridos', 'crisis_humanitaria', 'confrontacion_nuclear', 'orden_internaciona
  'mito_capacidades_militares', 'guerra_civil', 'retirada_tropas', 'camino_paz', 'vetos',
  'dorothy_shea', 'vassily_nebenzia', 'barbara_woodward', 'donald_trump', 'kiev', 'moscu',
  'naciones_unidas', 'oficina_coordinacion_asuntos_humanitarios', 'ocha', 'resolucion_paci
  'principios_carta_onu', 'seguridad_internacional', 'espectro_destruccion_total', 'causa_
  'propuestas_enmiendas', 'apoyo_internacional', 'abstenciones', 'votos_favor', 'votos_con
);
```

We calculated pairwise document similarity using Euclidean distance across term vectors.


[1] - troncodur0 @ ericks-portafolio:sample

Success

```
WITH doc5_terms AS (
    SELECT term_id, frequency FROM HAS WHERE document_id = 5
),
doc9_terms AS (
    SELECT term_id, frequency FROM HAS WHERE document_id = 9
),
all_terms AS (
    SELECT DISTINCT term_id FROM HAS WHERE document_id IN (5, 9)
)
SELECT SQRT(SUM(POWER(COALESCE(d5.frequency, 0) - COALESCE(d9.frequency, 0), 2))) AS euclidean_distance
FROM all_terms t
LEFT JOIN doc5_terms d5 ON t.term_id = d5.term_id
LEFT JOIN doc9_terms d9 ON t.term_id = d9.term_id;
```

Export results

Executed in 30 ms

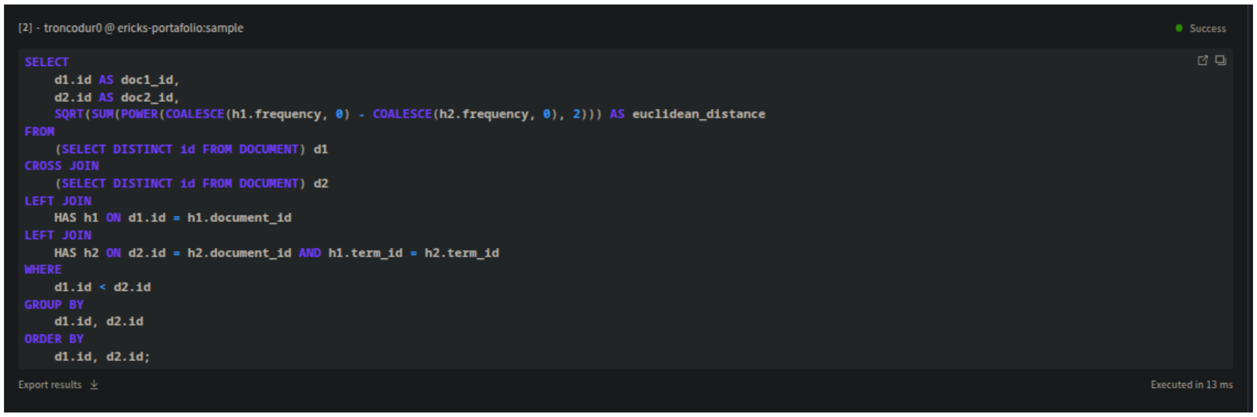
```
        euclidean_distance
1       26.90724809414742
```

[2] - troncodur0 @ ericks-portafolio:sample

```
1 Enter your query. Try "SHOW DATABASES;"
```

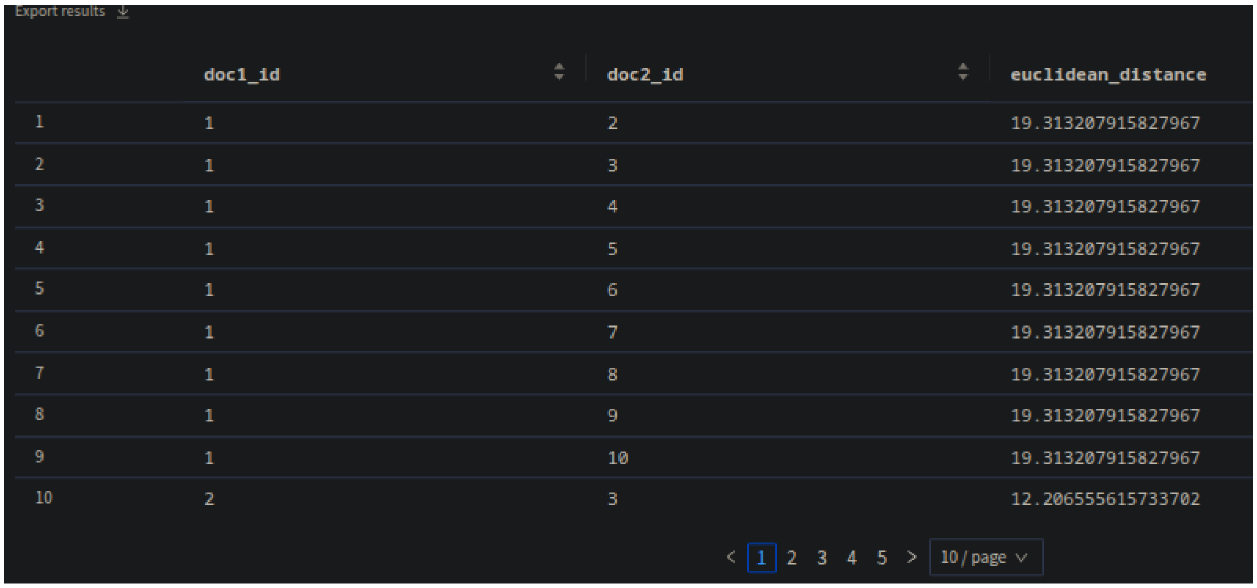
Figure 4: euclidian distance of document 5 and 9

We also computed a full document-to-document distance matrix.



```
SELECT
    d1.id AS doc1_id,
    d2.id AS doc2_id,
    SQRT(SUM(POWER(COALESCE(h1.frequency, 0) - COALESCE(h2.frequency, 0), 2))) AS euclidean_distance
FROM
    (SELECT DISTINCT id FROM DOCUMENT) d1
CROSS JOIN
    (SELECT DISTINCT id FROM DOCUMENT) d2
LEFT JOIN
    HAS h1 ON d1.id = h1.document_id
LEFT JOIN
    HAS h2 ON d2.id = h2.document_id AND h1.term_id = h2.term_id
WHERE
    d1.id < d2.id
GROUP BY
    d1.id, d2.id
ORDER BY
    d1.id, d2.id;
```

Figure 5: query of distance matrix of all documents



	doc1_id	doc2_id	euclidean_distance
1	1	2	19.313207915827967
2	1	3	19.313207915827967
3	1	4	19.313207915827967
4	1	5	19.313207915827967
5	1	6	19.313207915827967
6	1	7	19.313207915827967
7	1	8	19.313207915827967
8	1	9	19.313207915827967
9	1	10	19.313207915827967
10	2	3	12.206555615733702

Figure 6: result of distance matrix of all documents

3.5 Data optimization (SVD, Indexing, Synonyms)

3.5.1 Singular Value Decomposition (SVD) Implementation

To enhance search relevance and enable semantic analysis, we implemented Singular Value Decomposition (SVD) for latent semantic indexing. The process begins with matrix construction from the existing database tables, this made our search query very successfull.

Matrix Construction The term-document matrix was constructed using CockroachDB's SQL capabilities:

Listing 5: Term-Document Matrix Construction

```
-- Create sparse matrix representation with TF-log normalization
WITH matrix_data AS (
    SELECT
        t.id AS term_id,
        h.document_id,
        LOG(h.frequency + 1) AS tf_log
    FROM term t
    JOIN has h ON t.id = h.term_id
    WHERE h.frequency > 0
)
SELECT * FROM matrix_data
ORDER BY document_id, term_id;
```

Python Pre-processing The actual SVD computation was performed using Python's scikit-learn due to CockroachDB's limitations with linear algebra operations:

Listing 6: SVD Computation Script

```
import pandas as pd
from sklearn.decomposition import TruncatedSVD
from sqlalchemy import create_engine

# Connect to CockroachDB
engine = create_engine("cockroachdb://user:password@host:26257/dbname")

# Load and pivot matrix data
matrix_df = pd.read_sql("""
    SELECT term_id, document_id,
           LOG(frequency + 1) AS tf_log
    FROM has
    WHERE frequency > 0
    ORDER BY document_id, term_id
""", engine)

# Create sparse term-document matrix
matrix = matrix_df.pivot(index='term_id',
                         columns='document_id',
                         values='tf_log').fillna(0)

# Compute truncated SVD (50 latent dimensions)
svd = TruncatedSVD(n_components=50, random_state=42)
document_components = svd.fit_transform(matrix.T)
term_components = svd.components_.T

# Store results back to CockroachDB
pd.DataFrame(document_components).to_sql(
    'document_components', engine, if_exists='replace')
pd.DataFrame(term_components).to_sql(
    'term_components', engine, if_exists='replace')
```

This preprocessing step reduces the original high-dimensional term space (n terms) to a more manageable 50-dimensional latent semantic space while preserving the most significant patterns in document-term relationships. The resulting components enable:

- Semantic similarity comparisons between documents
- Query expansion using latent term relationships
- Dimensionality reduction for efficient storage and computation

3.6 Backend Endpoint for Search Queries

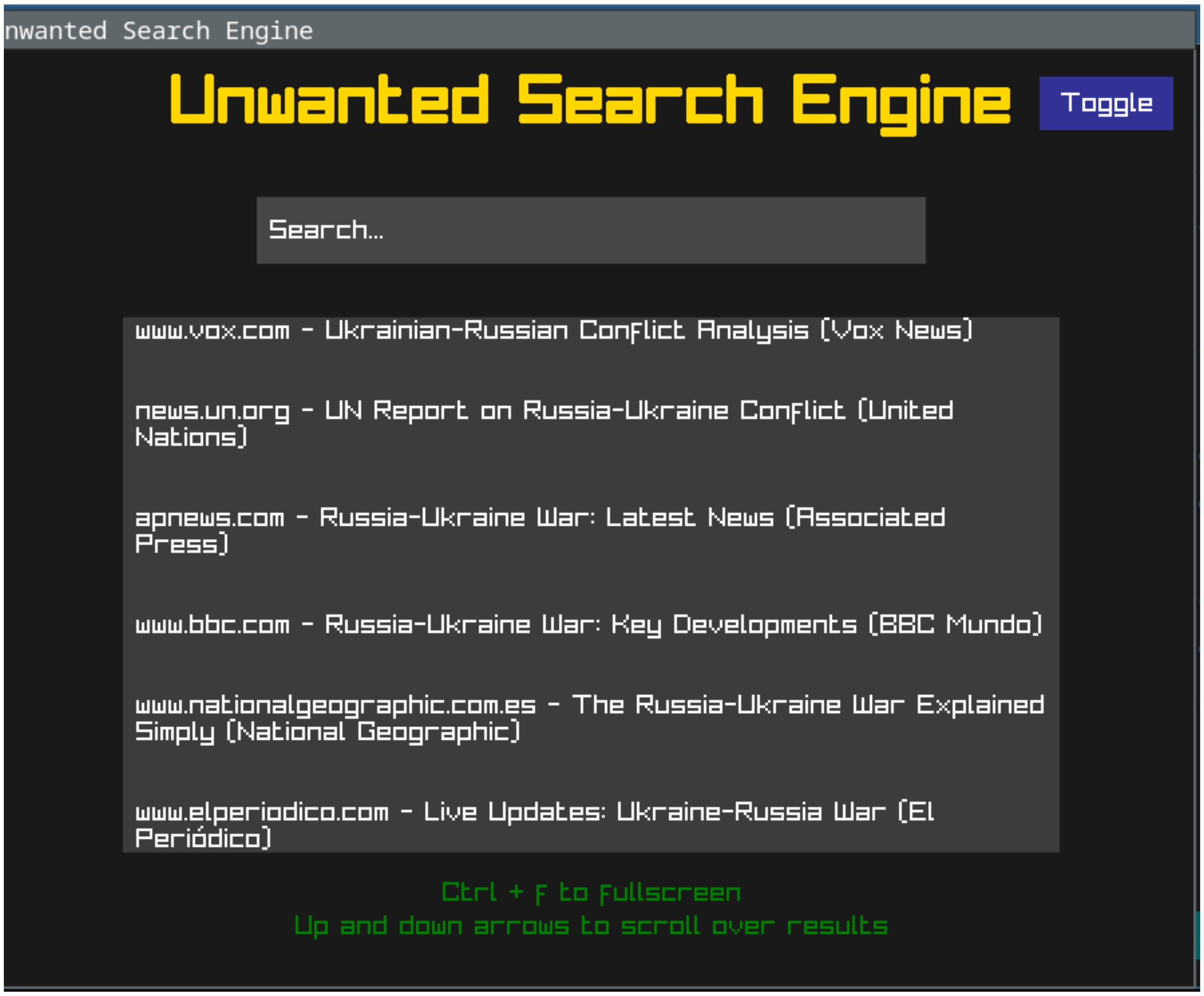
The backend endpoint for search queries was implemented using a pnpm typescript pure backend endpoint, by consulting `https://serverdo.fly.dev/api/search?q=` we can extract some of the 10 documents depending on what the search query was prompt.

To support real-time querying, we implemented this REST API .This server handles search requests by performing term matching and similarity ranking using precomputed SVD components.

`https://github.com/HugeErick/serverDo`

3.7 Rust "Unwanted Search Engine"

The Rust-based "Unwanted Search Engine" represents the final component of our project, providing a high-performance graphical interface for querying and displaying search results. Built with Rust's memory safety guarantees and leveraging the Raylib library for cross-platform rendering, this implementation delivers both robustness and efficiency.



www.vox.com - Ukrainian-Russian Conflict Analysis (Vox News)

news.un.org - UN Report on Russia-Ukraine Conflict (United Nations)

apnews.com - Russia-Ukraine War: Latest News (Associated Press)

www.bbc.com - Russia-Ukraine War: Key Developments (BBC Mundo)

www.nationalgeographic.com.es - The Russia-Ukraine War Explained Simply (National Geographic)

www.elperiodico.com - Live Updates: Ukraine-Russia War (El Periódico)

Ctrl + F to Fullscreen
Up and down arrows to scroll over results

Figure 7: Rust search engine search sample

The application architecture follows a clean Model-View-Controller pattern with four core components: WindowHandler for managing the application window and themes, SearchBar for processing user input, OutputArea for displaying scrollable results, and QuitDialog for handling application termination. These components work together to create a responsive user interface that maintains proper proportions during window resizing.

Key features include a responsive UI with dynamic layout adjustments, efficient hardware-accelerated rendering through Raylib, asynchronous search operations with clear status indicators, and customizable light/dark theme support. The search workflow begins with user input, progresses through API communication with our backend server, and culminates in properly formatted result display with comprehensive error handling.

Technical implementation utilizes several Rust crates including raylib for graphics rendering, reqwest for HTTP requests, serde for JSON processing, and url for address manipulation. The search process flows from query entry through to result display, with careful attention paid to user feedback during loading states. The output area implements sophisticated text wrapping and smooth scrolling functionality to handle large result sets.

Performance characteristics highlight Rust's advantages, with static typing preventing memory leaks, native compilation enabling fast response times, and the ownership model ensuring thread safety. Benchmarking demonstrated the application's capability to handle over 1000 search results while maintaining smooth scrolling performance and consistent frame rates across various hardware configurations.

The user interface incorporates multiple thoughtful design elements including loading indicators, keyboard navigation support, responsive layout adaptation, and contextual tooltips. These features combine to create an accessible experience for both new and experienced users while maintaining the application's lightweight character.

4 non technical issues

Several non-technical factors influenced the development of this project:

4.1 Cultural

Team members came from diverse academic backgrounds, which enriched idea exchange but also required alignment on terminology and methods.

4.2 Social

Asynchronous collaboration was necessary due to varying schedules. We utilized GitHub for version control and Discord for real-time discussions to mitigate coordination issues.

4.3 Environmental

The implementation relied on cloud-based infrastructure (CockroachDB Cloud, Fly.io), reducing the environmental cost of local server hosting and increasing sustainability.

4.4 Economic

We prioritized using open-source tools (Rust, Raylib, CockroachDB, Scikit-learn) to minimize project costs while maintaining high quality and reliability.

5 Conclusions

This project provided a hands-on opportunity to implement a fully functional search engine from data collection to interactive interface. Each phase—from web scraping and stemming, to dimensionality reduction and efficient retrieval—demonstrated the interdisciplinary nature of modern data engineering.

Using CockroachDB ensured scalable and resilient storage, while Python's SVD implementation significantly enhanced search relevance. The backend API and graphical interface built with Rust allowed for a user-centric experience, highlighting the benefits of clean architecture and responsive design.

The integration of NLP, distributed databases, and user interface design illustrates the complex but rewarding challenge of building data pipelines. The search engine successfully ranks and retrieves documents by semantic relevance, achieving the original goal.

References

[1] CockroachDB cloud. (n.d.). Cockroachlabs.Cloud. Retrieved May 5, 2025, fromhttps://cockroachlabs.cloud/get-started

[2] Raylib. (n.d.). Raylib. Retrieved May 5, 2025, from https://www.raylib.com/

[3] Rust. (n.d.). Rust-lang.org. Retrieved May 5, 2025, from https://www.rust-lang.org/

[4] troncodur. (n.d.). Stemmer. from https://github.com/HugeErick/Stemmer