Albert Jané Lardiés - U215114
Marc de los Aires Tello - U198732
Jordi Esteve Claramunt - U215108
https://github.com/jordiestevee/irwa-search-engine-g_019
TAG: IRWA-2025-part-4

Information Retrieval & Web Analytics - Part 4: RAG, UI, and Analytics

1. Introduction

This report describes the implementation of Part 4 of the Information Retrieval & Web Analytics project. In this last phase, we integrated the previously developed search algorithms into a fully functional Flask web application. A custom Web Analytics module to monitor user interactions and a Retrieval-Augmented Generation (RAG) component to generate AI summaries were added to the system. The goal was to create a robust, user-friendly search engine that not only retrieves documents but also assists users in decision-making and provide insights into system usage.

2. Search Engine Optimization & Architecture

2.1 Code Structure

The project is organized into a Flask application structured as follows:

- **web_app.py**: It runs the Flask server and handles HTTP routes for the search page, results page, and document details.
- **myapp/search/**: Directory which contains the core Information Retrieval logic.
    - search_engine.py: Contains the main SearchEngine class and the public search() function.
    - algorithms.py: Helper functions for BM25 and tokenization.
    - objects.py: Defines data models to ensure strict type validation for prices and ratings.
- **myapp/generation/rag.py**: Handles the interaction with the Large Language Model (LLM) for the RAG features.
- **myapp/analytics/**: Tracks user events and logging data to JSON.

2.2 Optimization for Web

To ensure the search engine performs efficiently in a real-time web environment, some optimizations were implemented in the search() function:

1. **Lazy Indexing:** The SearchEngine class checks if not self._indexed before searching. This ensures that the inverted index is built only once when the

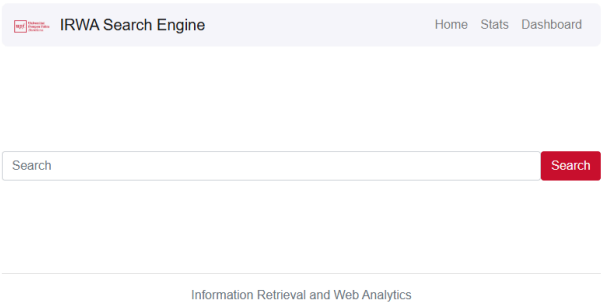
application starts (or the first search is run), rather than rebuilding it for every single user request. This significantly reduces latency.

2. **Soft Matching (Candidate Pre-filtering):** Before calculating complex BM25 scores for the whole corpus, the algorithm computes a "soft match." It identifies a set of candidates(documents that contain at least one term from the query). The expensive scoring math is only performed on this smaller subset, not the entire database.
3. **Data Validation via Pydantic:** The Document class in objects.py includes validators (e.g., parse_price, parse_rating). During the loading stage, this cleans the raw dataset by converting strings like "1,200" to floats. This optimization prevents runtime errors during the search ranking process and makes sure the UI always receives valid numbers.
4. **Search Function Design:** The search function definitionwas standardized to: def search(self, search_query, search_id, corpus, num_results=20)
    - **search_id**: Added to track the specific query instance for analytics purposes (linking the results to subsequent clicks).
    - **num_results**: A parameter to limit the output size (default 20), reducing the payload sent to the frontend.

3. User Interface Design

3.1 The Search Page

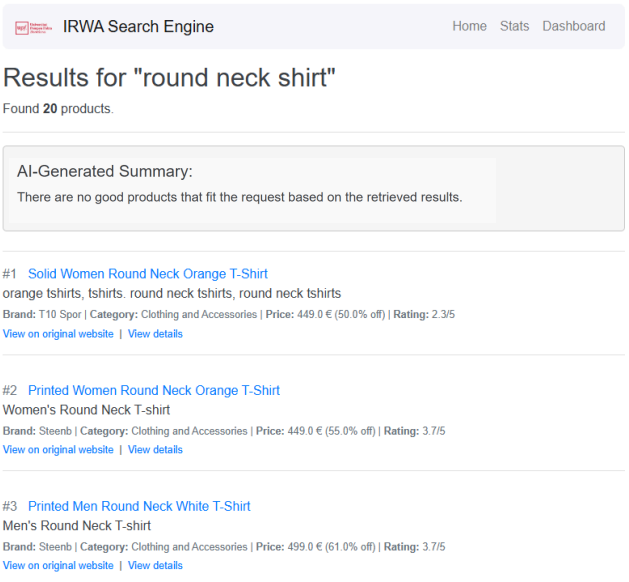
We implemented a central search bar and a "Search" button. A top navigation bar allows users to switch between "Home", "Stats" (Analytics), and "Dashboard".



3.2 The Results Page

The results page displays the retrieved documents in a vertical list format.

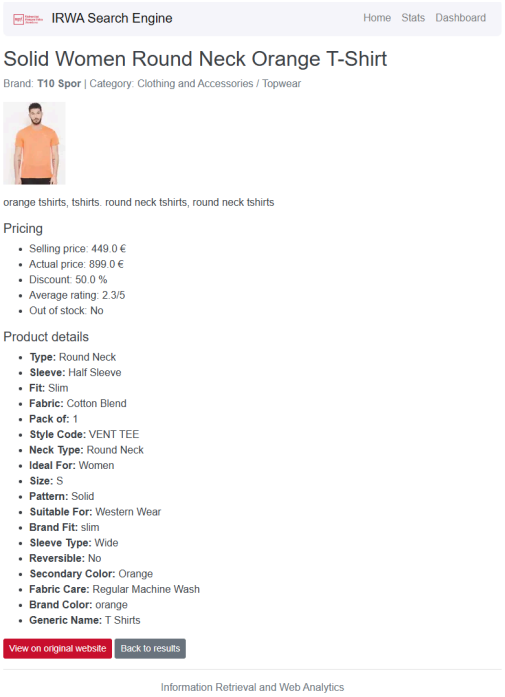
- **RAG Integration:** A "AI-Generated Summary" box is placed at the top of the results. This allows users to read a synthesized answer before scrolling through individual results.
- **List Layout:** Each result item shows the Title (hyperlinked) and the Description.
- **Ranking:** Results are ordered by their BM25 relevance score.



3.3 The Document Details Page

When a user clicks on a result, they are directed to a detailed view of the product. This page xpands on the metadata available in the corpus:

- **Pricing & Rating:** We display "Selling Price," "Actual Price," "Discount %," , "Average Rating" and “Out of stock”.
- **Product Specifications:** A list of attributes (Type, Sleeve, Fit, Fabric, Pattern) is rendered using bullet points.
- **Navigation:** Two buttons are shown: one to "View on original website" and another to "Back to results."



4. Retrieval-Augmented Generation (RAG)

4.1 Baseline Analysis

The baseline RAG pipeline provided a functional but naive implementation. It typically retrieved the top-k documents and fed their titles and descriptions to the LLM. We identified the following weaknesses:

- **Lack of Metadata:** The baseline did not include price, brand, or rating in the context. If a user asked for "cheap t-shirts," the LLM could not answer accurately because it didn't know the prices.
- **Hallucinations on Poor Results:** If the search engine returned irrelevant documents, the baseline LLM would often try to "make up" an answer based on the irrelevant context, misleading the user.

4.2 Proposed Improvements

To address these weaknesses, we implemented a RAGGenerator class in rag.py with two specific improvements:

Improvement 1: "No Good Products" Logic Gate

- **Description:** We implemented a logic gate before calling the LLM API. The system checks the BM25 score of the top-ranked result. If top_score < 0.5 (indicating low relevance) or if no results are found, the system bypasses the LLM and returns a

hardcoded message.It increases trust by admitting when the system cannot find a relevant answer, rather than hallucinating.

Improvement 2: Structured Metadata Injection & Prompt Engineering

- **Description:** We engineered a strict prompt (IMPROVED_PROMPT_TEMPLATE) that forces the LLM to act as a "product advisor." Crucially, we enrich the context passed to the LLM. Instead of just text, we inject the Price, Discount, Brand, and Rating for every retrieved item.
- **Prompt Strategy:** The prompt explicitly commands the model to: *"Pick the single best product... Explain why using concrete attributes."*
- **Comparison:**
    - *Baseline Context:* "Title: Blue Shirt..."
    - *Improved Context:* "PID: 123 | Title: Blue Shirt | Price: 500 | Discount: 10% | Rating: 4.5"

4.3 Final RAG Implementation

- **API Used:** We utilized the **Groq API** running the llama-3.1-8b-instant model.
- **Integration:** The generated summary is returned as a string and displayed in a dedicated alert box at the top of the results page.

5. Web Analytics Implementation

5.1 Data Collection Strategy

We implemented a custom analytics module (analytics_data.py) that captures user behavior:

- **Data Storage:** Usage data is stored in a JSON file (data/analytics.json) which acts as a persistent log.
- **Tracked Events:**
    1. **Queries:** We record the search string and the exact timestamp (datetime.now().isoformat()) every time a user hits search.
    2. **Clicks:** When a user clicks a product title, we record the Product ID (PID) and the timestamp.
- **Metrics:** We calculate the Click-Through Rate (CTR) proxy by measuring Average Clicks per Query (Total Clicks / Total Queries).

5.2 Data Model

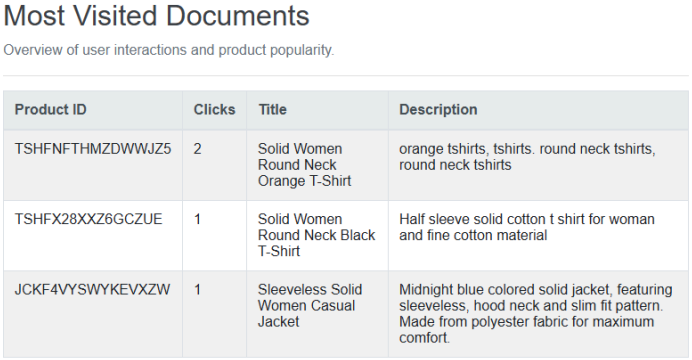
- **Fact Tables:**
    - queries: Acts as a transaction log recording every search intent.
    - clicks: Acts as an interaction log recording user engagement.
- **Dimensions:**
    - **Time:** All events are timestamped.

- **Document:** The clicks table references documents by PID, which links back to the main Corpus.

5.3 The Analytics Dashboard and Stats page

We created a stats page (/stats) and a dashboard page (/dashboard) to visualize these metrics:

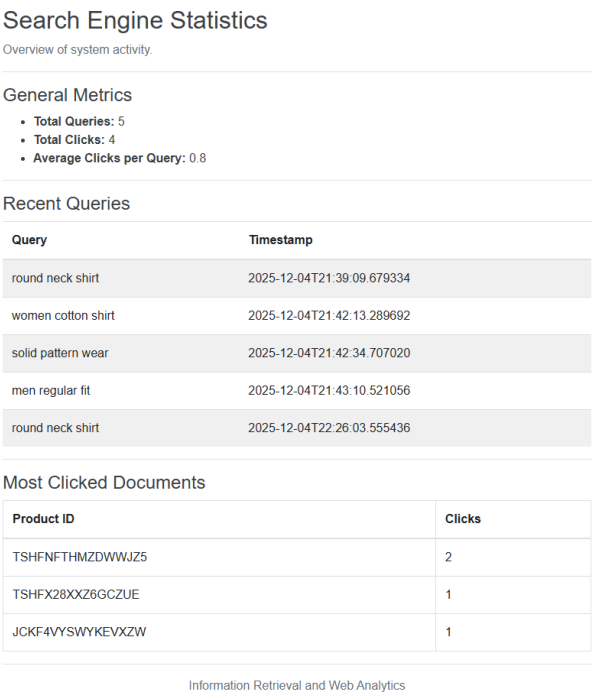
- Most Visited Documents
- Click Distribution Chart
- Average clicks per query
- Recent Queries



Most Visited Documents

Overview of user interactions and product popularity.

Product ID	Clicks	Title	Description
TSHFNFTHMZDWWJZ5	2	Solid Women Round Neck Orange T-Shirt	orange tshirts, tshirts. round neck tshirts, round neck tshirts
TSHFX28XXZ6GCZUE	1	Solid Women Round Neck Black T-Shirt	Half sleeve solid cotton t shirt for woman and fine cotton material
JCKF4VYSWYKEVXZW	1	Sleeveless Solid Women Casual Jacket	Midnight blue colored solid jacket, featuring sleeveless, hood neck and slim fit pattern. Made from polyester fabric for maximum comfort.



Search Engine Statistics

Overview of system activity.

General Metrics

- **Total Queries:** 5
- **Total Clicks:** 4
- **Average Clicks per Query:** 0.8

Recent Queries

Query	Timestamp
round neck shirt	2025-12-04T21:39:09.679334
women cotton shirt	2025-12-04T21:42:13.289692
solid pattern wear	2025-12-04T21:42:34.707020
men regular fit	2025-12-04T21:43:10.521056
round neck shirt	2025-12-04T22:26:03.555436

Most Clicked Documents

Product ID	Clicks
TSHFNFTHMZDWWJZ5	2
TSHFX28XXZ6GCZUE	1
JCKF4VYSWYKEVXZW	1

Information Retrieval and Web Analytics

AI Usage Statement

ChatGPT (GPT-5) was used to structure and format this report. All code and results were implemented and verified by the authors.