Information Retrieval Final Report

Edoardo Ababei, Joaquin Perdomo Roget

December 7, 2024

Repository

The source code for this project is available on GitHub: https://github.com/edoardo-aba/travels

Introduction

This report outlines the design, implementation, and functionality of a web-based search application that integrates web scraping, indexing, and various search-related features. The primary objectives were to construct an end-to-end system that: (1) scrapes content from multiple external websites, (2) stores the extracted data in a database for persistence, (3) indexes the data using a search engine for efficient retrieval, and (4) provides a rich frontend interface to present, filter, and refine the search results. In addition, the application includes features such as result snippets, user relevance feedback, and a pseudo recommendation system to enhance the user's search experience.

Throughout this document, the key technologies and workflows are explained, including how the system fetches data, indexes it into Apache Solr, and displays it interactively via a React-based frontend. The chosen features from the project specification are also described in depth.

Technologies Used

Backend Technologies

Node.js and Express.js: The backend server is implemented using Node.js and the Express.js framework. Node.js provides an asynchronous, event-driven runtime suitable for I/O-intensive tasks like web scraping and making multiple network requests. Express.js offers a clean, minimal framework for defining APIs and handling requests and responses.

Puppeteer and Puppeteer-extra (with Stealth Plugin): Puppeteer is a Node.js library that provides a high-level API to control headless instances of the Chrome browser. This project uses Puppeteer for scraping data from various web pages. The Puppeteer Stealth Plugin helps mask the automated browser to reduce detection, ensuring stable scraping sessions on the target websites.

Different scraper functions are used for different sources because each site may have unique DOM structures. After navigating to the page and waiting for content to load, Puppeteer extracts the title, description, and image metadata from HTML elements.

MongoDB and Mongoose: MongoDB is a NoSQL database chosen for its flexibility in handling unstructured or semi-structured data, which is common in scraped content. Mongoose, an ODM (Object Data Modeling) library, allows for schema-based interactions with MongoDB, simplifying document creation, updates, and queries. The backend stores the scraped data into a MongoDB collection, making it easy to re-scrape and refresh the data.

Apache Solr: Solr is a highly reliable, scalable, and fault-tolerant search platform built on Apache Lucene. After the scraped data is saved in MongoDB, it is indexed into Solr for full-text search capabilities. Solr supports fast keyword queries, filtering, scoring, and sorting results by relevance. Integrating Solr ensures that when a user searches for a term on the frontend, the query is routed through Solr, returning structured JSON responses containing the most relevant documents.

Cron (node-cron): To keep the dataset fresh, node-cron schedules periodic scraping and re-indexing. This ensures that if source websites update their content, the local database and Solr index remain up-to-date.

NLP Tools (@nlpjs/lang-en-min): For better result relevance and snippet generation, a simple English tokenizer and stopword remover are used. These tools help in refining search ranking by tokenizing user queries and documents, removing common stopwords, and improving the scoring mechanism used by the custom logic in the backend.

Frontend Technologies

React.js: The frontend is developed in React, enabling a modular, component-based architecture. This allows for rapid UI development and easy maintenance. Key components such as search bars, result cards, recommendation sections, and snippet highlighting are implemented as separate, reusable pieces.

Axios: Axios is used in the frontend to handle HTTP requests to the Express.js backend. It simplifies making API calls, handling responses, and catching errors.

CSS and DOMPurify: Basic CSS is used for styling. DOMPurify is utilized to safely set HTML (for highlighted snippets) to prevent XSS vulnerabilities.

System Workflow

Data Flow: From Scraping to Search Results

The lifecycle of data in this application involves several key steps:

1. **Scraping:** The backend defines a list of target URLs along with a dedicated scraper function for each website. Puppeteer is launched in headless mode and navigates to each URL. Once the page's DOM is ready, Puppeteer extracts fields like title, description, and image URLs. Any textual descriptions are cleaned and trimmed, and the data is stored temporarily before insertion into the database.

2. **Data Storage in MongoDB:** After scraping, all previously existing documents in MongoDB are deleted to maintain a clean dataset. The scraped documents are inserted into the `website` collection. Each document includes fields for *title*, *description*, *image*, *source*, and a *relevance* score initialized at 100.

3. **Indexing in Solr:** After updating MongoDB, the system synchronizes data with Solr. It first deletes any existing documents in Solr to avoid duplicates or stale entries. The current documents from MongoDB are then fetched and added to Solr's `websites` core. A commit operation ensures that Solr's index is up-to-date and ready for queries.

4. **Searching and Ranking:** When a user searches for a term on the frontend, a request is sent to the Express server. The backend queries Solr using a combination of fields (title and description) to find the best matches. The raw results from Solr are then re-ranked by the backend. This ranking leverages tokenization, stopword removal, and counting query term occurrences to produce a final score that combines the document's original relevance score and its textual match quality.

5. **Rendering Results on the Frontend:** The frontend receives a list of documents with titles, descriptions, images, and sources. These results are displayed in a structured layout. Query terms are highlighted in the displayed snippets, providing users with a quick preview of relevance.

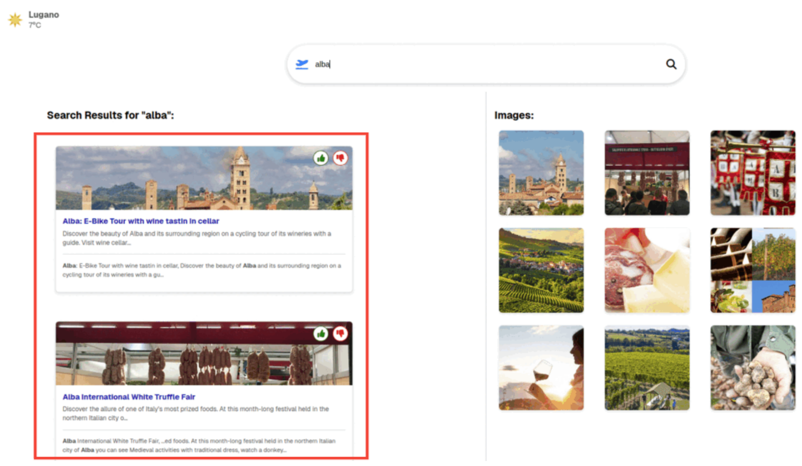
Implemented Functionalities

According to the project's requirement, the application needed at least one feature from the "Simple Features" group and one from the "Complex Features" group. The chosen features are:

Simple Features Implemented

Features and Implementation

Results Presentation: Results are displayed in a card-based layout. Each card shows the title, a short description, and a snippet containing highlighted query terms. The arrangement makes it easy for users to scan and compare multiple results simultaneously.


Lugano
7°C

alba

Search Results for "alba":

Alba: E-Bike Tour with wine tastin in cellar
Discover the beauty of Alba and its surrounding region on a cycling tour of its wineries with a guide. Visit wine cellar...

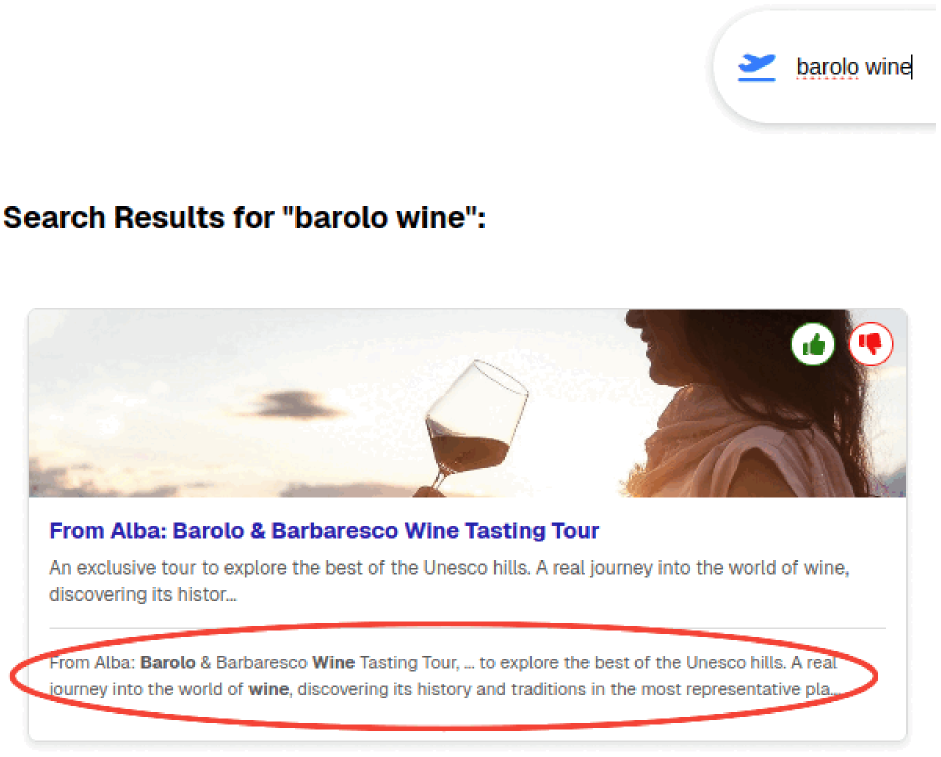
Alba: E-Bike Tour with wine tastin in cellar, Discover the beauty of **Alba** and its surrounding region on a cycling tour of its wineries with a gu...

Alba International White Truffle Fair
Discover the allure of one of Italy's most prized foods. At this month-long festival held in the northern Italian city o...

Alba International White Truffle Fair, ...ed foods. At this month-long festival held in the northern Italian city of **Alba** you can see Medieval activities with traditional dress, watch a donkey...

Images:

Results Snippets: For each returned result, the system generates a snippet similar to a Google search result. Query terms in the snippet are highlighted. The backend uses substring extraction and HTML markup to emphasize terms. On the frontend, DOMPurify ensures safe insertion of highlighted snippets. This helps users quickly assess the relevance of a result.


barolo wine

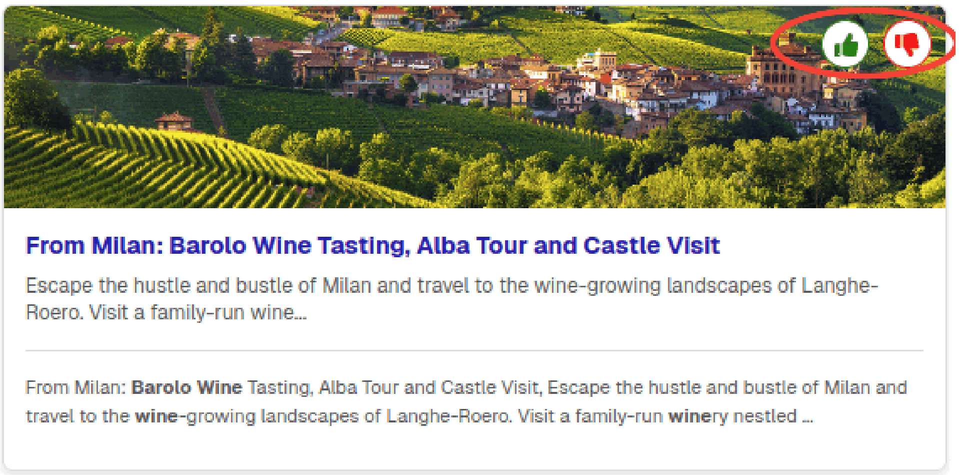
Search Results for "barolo wine":

From Alba: Barolo & Barbaresco Wine Tasting Tour
An exclusive tour to explore the best of the Unesco hills. A real journey into the world of wine, discovering its histor...

From Alba: **Barolo** & Barbaresco **Wine** Tasting Tour, ... to explore the best of the Unesco hills. A real journey into the world of **wine**, discovering its history and traditions in the most representative pla...

Complex Features Implemented

User Relevance Feedback: Users can provide feedback on the results by marking them as relevant (thumbs up) or irrelevant (thumbs down). When a user clicks the feedback buttons on a result, the backend updates the relevance score in MongoDB and Solr. Positive feedback increments the relevance by 1, and negative feedback decrements it by 1. After feedback is recorded, users can re-run the query to see updated rankings, introducing an element of personalized and iterative search result refinement.

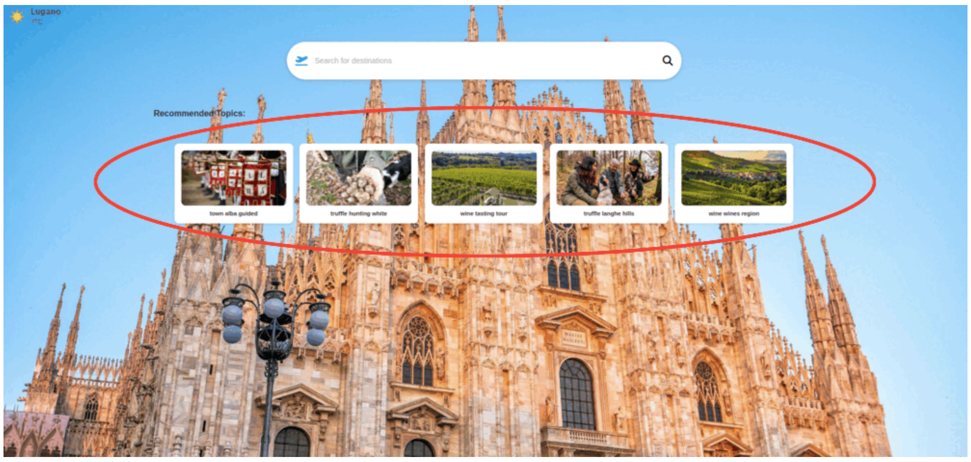


From Milan: Barolo Wine Tasting, Alba Tour and Castle Visit

Escape the hustle and bustle of Milan and travel to the wine-growing landscapes of Langhe-Roero. Visit a family-run wine...

From Milan: **Barolo Wine** Tasting, Alba Tour and Castle Visit, Escape the hustle and bustle of Milan and travel to the **wine**-growing landscapes of Langhe-Roero. Visit a family-run **wine**ry nestled ...

Pseudo Automatic Recommendation: The application also includes a pseudo recommendation feature displayed at the start (before the user performs a search). A set of recommended items is shown based on their relevance scores. While this recommendation is not dynamically tailored to the user's query, it offers a starting point for exploration. By clicking on these recommended items, the application extracts keywords from their titles and descriptions, triggering a search that guides the user into related content.



Conclusion

This project demonstrates a comprehensive approach to building a search-focused web application. It integrates multiple technologies—web scraping with Puppeteer, data storage in MongoDB, indexing and search via Solr, and a React frontend. The chosen features—Results Presentation, Results Snippets, User Relevance Feedback, and Pseudo Automatic Recommendation—collectively enhance the user experience. Users see query-matching snippets at a glance, refine results by providing relevance feedback, and start their exploration with recommended items immediately.

The architecture is modular and extensible, allowing future additions like filtering and clustering. The reliance on standard tools and clear data flow ensures maintainability and scalability. Overall, this system meets the project requirements and lays a foundation for further enhancements in search functionality and user interactivity.

References

- Puppeteer Documentation: `https://github.com/puppeteer/puppeteer`
- Solr Reference Guide: `https://solr.apache.org/`
- Mongoose Documentation: `https://mongoosejs.com/`
- React Documentation: `https://reactjs.org/`
- NLP.js: `https://github.com/axa-group/nlp.js/`