UNIVERSITY OF CRETE COMPUTER SCIENCE DEPARTMENT

Diploma Thesis

Enriching the Demo of SPARQL-QBE over Movies with Images

Author: Maria Foteini Troupi

AM: CSD 4055

Supervisor: Prof. Yannis Tzitzikas

Member of the Examination Committee: Prof. Dimitris Plexousakis

1. Abstract

One of the most significant challenges in modern technology is ensuring that systems are accessible and user-friendly for non-expert users. This thesis focuses on enhancing the front-end demo of a SPARQL-QBE system [1], designed for querying movie databases, with the goal of making the user interface (UI) more visually appealing and easier to use. By integrating additional user-friendly information and visual elements, the enhanced demo aims to improve user engagement and efficiency.

The project leverages modern front-end technologies, including JavaScript, HTML, and CSS, to redesign the UI, providing a clearer and more intuitive layout. These changes enable users to navigate the interface more effectively and highlight key information with greater efficiency. Additionally, the demo integrates an external API source (The Movie Database - TMDb) to fetch complementary information for each movie, such as posters, overviews, ratings, genres, release dates, and links to official trailers. This enriched data helps users better understand and refine their search results, especially when they are unsure of what they are looking for or lack confidence in their query choices.

By combining visual enhancements with enriched data, this project demonstrates how user-friendly interfaces can significantly improve the usability of complex systems like SPARQL-QBE, making them more accessible to non-expert users. The results highlight the importance of visual elements and additional information in creating intuitive and engaging user experiences.

2. Acknowledgments

I would like to express my gratitude to my supervisor, Prof. Yannis Tzitzikas, for his guidance and support throughout this project. I also thank the University of Crete and the Computer Science Department for providing the resources and environment necessary to complete this work.

3. Introduction

3.1. Background

Structured query formulation over Knowledge Graphs (KGs) is a complex task, particularly for users unfamiliar with the underlying data structure or query language syntax. SPARQL, the standard query language for RDF data, is powerful but often inaccessible to non-expert users due to its steep learning curve. To address this challenge, Query By Example (QBE) has emerged as a user-friendly paradigm, enabling users to construct queries by providing examples and feedback rather than writing complex code. This approach is inspired by QBE systems in relational databases and relevance feedback mechanisms in information retrieval.

The SPARQL-QBE method, as proposed by [Akritas Akritidis, 2021], introduces an interactive, example-based approach for SPARQL query formulation. Unlike traditional methods, SPARQL-QBE allows users to provide positive and negative examples and offer feedback on generated constraints, making it accessible even to users without prior knowledge of SPARQL or the KG's structure.

3.2. Problem Statement

The existing SPARQL-QBE demo for querying movie databases effectively demonstrates the functionality of SPARQL Query by Example (QBE) but suffers from several limitations that hinder its usability and user engagement. The key issues identified are as follows:

- **Lack of Visual Elements:** The current demo primarily presents query results in a text-based format, which lacks visual appeal and fails to engage users effectively. In today's digital age, users expect rich, multimedia experiences that combine textual information with visual elements such as images, icons, and interactive components.

The absence of movie posters or other visual aids makes the interface less intuitive and less appealing, particularly for users unfamiliar with SPARQL or semantic web technologies.

- **Incomplete Movie Search Results:** The existing demo provides basic information about movies, such as titles and actors, but lacks additional details that would make the search results more comprehensive and informative. For example, users might want to see movie posters, release dates, genres, ratings, and trailers to better understand and interact with the results. This lack of enriched data limits the demo's utility as a tool for exploring and analyzing movie databases.
- **Unrelevant and no user-friendly information:** While the information relevant to the position of the information in the database for expert users might be very informatory, non-expert users may find this overwhelming and detracting for their goals.

3.3. Objective

The goal of this project is to enhance the existing SPARQL-QBE demo by addressing its key limitations and improving its usability and user engagement. Specifically, the project aims to:

1. **Integrate Visual Elements:** Enrich the demo by incorporating movie posters and other visual aids to create a more engaging and intuitive user interface.
2. **Provide Comprehensive Search Results:** Enhance the search results by fetching and displaying additional movie details, such as release dates, genres, ratings, and trailers, from The Movie Database (TMDb) API.
3. **Resolve Data Integration Challenges:** Develop a reliable mechanism to map internal movie identifiers to TMDb's unique identifiers, ensuring accurate and seamless integration of external data.
4. **Improve the Front-End Experience:** Leverage modern front-end technologies, including JavaScript, jQuery, CSS3, HTML5, and Bootstrap, to create a visually appealing, responsive, and interactive interface.

By achieving these objectives, the enhanced SPARQL-QBE demo will become a more comprehensive and user-friendly tool for exploring and querying movie databases, catering to both expert and non-expert users. This project not only addresses the current limitations but also demonstrates the importance of visual elements and enriched data in improving the usability of semantic web applications.

3.4. Scope

This project focuses on enhancing the front-end of the existing SPARQL-QBE demo for the **movie database** querying by integrating visual elements and improving the overall user experience. The scope of the project includes the following key areas:

1. **Integration of The Movie Database (TMDb) API:** The project leverages the TMDb API to fetch movie posters and additional metadata, such as the overviews, release dates, genres, ratings, alternative titles and trailers, if are available. This integration enriches the search results and provides users with a more comprehensive and visually appealing interface.
2. **Front-End Enhancements:** The project utilizes modern web technologies, including JavaScript, jQuery, CSS3, HTML5, and Bootstrap 5, to redesign and improve the user interface. These enhancements ensure a responsive, interactive, and visually engaging experience for users.
3. **Image Lazy Loading:** To optimize performance and reduce page load times, the project implements lazy loading for movie posters. This technique ensures that images are loaded only when they are needed, improving the overall efficiency of the application.
4. **Data Integration and Mapping:** The project addresses the challenge of integrating external data from TMDb by developing a mechanism to map internal movie identifiers (used in the SPARQL-QBE demo) to TMDb's unique identifiers. This ensures accurate and reliable data retrieval and display.
5. **User Experience Improvements:** The project aims to create a more intuitive and user-friendly interface by incorporating visual elements, interactive components, and enriched data. These improvements are designed to enhance usability, particularly for non-expert users who may not be familiar with SPARQL or semantic web technologies.
6. **Browser Developer Tools for Testing and Debugging:** Throughout the development process, browser developer tools are used extensively to test, debug, and optimize the front-end implementation. This ensures compatibility across different browsers and devices.

3.5. Out of Scope

While this project focuses on front-end enhancements and data integration, it does not cover the following:

1. **Back-End Modifications:** The project does not involve changes to the underlying SPARQL-QBE query engine or the back-end infrastructure.
2. **Advanced Query Features:** The project does not extend the functionality of the SPARQL-QBE system itself but rather enhances its presentation and usability.
3. **Comprehensive API Integration:** While TMDb API is used for fetching movie posters and metadata, the project does not explore integration with other external APIs or data sources.
4. **Mobile responsive UI:** While the initial demo project could be modified to have seamless responsive design through different devices and screen sizes, the TMDB API registration process is not optimized for mobile devices so you should access these pages on a desktop computer and browser.

4. Literature Review

4.1. SPARQL and Query By Example (QBE)

Structured query formulation over Knowledge Graphs (KGs) is a complex task, particularly for users unfamiliar with the underlying data structure or query language syntax. SPARQL, the standard query language for RDF data, is powerful but often inaccessible to non-expert users due to its steep learning curve. To address this challenge, Query By Example (QBE) has emerged as a user-friendly paradigm, enabling users to construct queries by providing examples and feedback rather than writing complex code. This approach is inspired by QBE systems in relational databases and relevance feedback mechanisms in information retrieval.

The SPARQL-QBE method, as proposed by [Akritas Akritidis, 2021], introduces an interactive, example-based approach for SPARQL query formulation. Unlike traditional methods, SPARQL-QBE allows users to provide positive and negative examples and offer feedback on generated constraints, making it accessible even to users without prior knowledge of SPARQL or the KG's structure. For instance, a user can start by selecting examples (e.g., movies like The Prestige and The Dark Knight) through keyword search. The system then generates a list of common constraints and returns entities that satisfy them. Users can refine the results by providing additional feedback, such as rejecting unwanted constraints (e.g., excluding movies starring Christian Bale) or adding new examples. This iterative feedback loop enables users to gradually formulate precise queries without needing to understand SPARQL syntax.

A key distinction of SPARQL-QBE is its support for negative examples and constraint feedback, features not commonly found in other QBE-like systems for KGs. For example, systems like Qbees [7] lack support for negative examples, while others like Query from Examples [8] require users to answer multiple questions, adding complexity to the interaction. SPARQL-QBE, on the other hand, simplifies the process by dynamically generating SPARQL queries based on user input and feedback, making it both intuitive and effective.

The feasibility and effectiveness of SPARQL-QBE have been demonstrated through its application to real-world datasets, such as DBpedia (Movies, Actors) and scientific papers. A task-based evaluation involving users unfamiliar with SPARQL confirmed that the interaction is easy to grasp and enables most users to formulate their desired queries successfully. This highlights the potential of SPARQL-QBE to bridge the gap between complex query languages and non-expert users, making semantic web technologies more accessible.

However, despite its advantages, existing SPARQL-QBE implementations often lack visual elements such as images, which could further enhance usability and user engagement. This gap presents an opportunity to enrich SPARQL-QBE demos by integrating multimedia content, such as movie posters, to create a more intuitive and visually appealing interface.

4.2. The Movie Database (TMDb) API

Movie databases, such as The Movie Database (TMDb), provide extensive metadata and multimedia content, including posters, trailers, and cast information. APIs like TMDb enable developers to programmatically access this data, making it a valuable resource for enhancing user interfaces. According to [Nektarios Makrydakis, 2023][2], The integration of multimedia elements into web pages provides a visually appealing and interactive experience that captures users' attention and encourages them to spend more time on a website. Multimedia-rich content is crucial in enhancing user engagement. Studies from recent years highlight those websites incorporating relevant images and videos experience significantly lower bounce rates compared to text-heavy pages lacking visual content. However, challenges such as API rate limits and data consistency must be addressed to ensure seamless integration.

4.3. Front-End Technologies

Modern web development relies heavily on technologies like JavaScript, jQuery, CSS3, HTML5, and Bootstrap to create responsive and interactive user interfaces.

- **JavaScript and jQuery:** These technologies enable dynamic content updates and interactivity, making it possible to fetch and display data from APIs like TMDb without reloading the page. jQuery simplifies DOM manipulation and event handling, reducing development time and improving cross-browser compatibility.
- **CSS3 and HTML5:** These standards provide the foundation for structuring and styling web content. CSS3 allows for advanced styling, animations, and responsive design, while HTML5 introduces semantic elements and multimedia support, enhancing the overall user experience.
- **Bootstrap**: This front-end framework simplifies the development of visually appealing and mobile-friendly designs. Its grid system, pre-designed components, and utility classes ensure consistency and responsiveness across devices.

Studies have shown that Bootstrap significantly enhances web development efficiency and usability [9], particularly for mobile devices. By providing pre-designed components and responsive design features, it reduces development time and ensures adaptability across screen sizes. Case studies show improved loading times, navigation speed, and user satisfaction, proving Bootstrap's effectiveness in creating modern, responsive web interfaces.

4.4. Image Lazy Loading

Lazy loading is a performance optimization technique that delays the loading of non-critical resources, such as images or videos, until they are needed (e.g., when they enter the viewport). This approach reduces initial page load time, decreases bandwidth usage, and improves user experience, particularly for content-heavy websites. According to web.dev[20], lazy loading is especially effective for improving Largest Contentful Paint (LCP), a Core Web Vital metric that measures the loading performance of the largest visible element on a webpage.

The implementation of lazy loading can significantly enhance LCP by prioritizing the loading of above-the-fold content while deferring off-screen resources. However, improper use of lazy loading, such as applying it to critical elements like hero images or LCP candidates, can negatively impact performance. For instance, lazy loading an image that is part of the LCP can delay its rendering, leading to poor user experiences. Therefore, it is crucial to identify and exclude LCP elements from lazy loading to ensure optimal performance.

Modern browsers support native lazy loading through the *“loading="lazy"”* attribute for images and *“iframes”*, which simplifies implementation. Additionally, JavaScript-based solutions and libraries like Intersection Observer API provide more granular control over lazy loading behavior. These tools enable developers to dynamically load resources as users scroll, further improving page speed and responsiveness.

In conclusion, lazy loading is a powerful technique for optimizing web performance, but its effectiveness depends on proper implementation. By prioritizing critical content and deferring non-essential resources, developers can achieve faster LCP scores, reduce load times, and enhance overall user satisfaction. Future research could explore advanced lazy loading strategies, such as adaptive loading based on network conditions, to further optimize performance across diverse user environments.

4.5. User-Centered Design Principles

Previous research in web design and user-centered design emphasizes that usability alone is no longer sufficient for modern systems. Designers must also focus on aesthetic value and creating enjoyable user experiences. As noted by [3], systems that inject a sense of fun and pleasure into users' lives are more likely to retain their attention and encourage repeat visits. This is particularly important because users form their first impression of a website within 50 milliseconds [3][5], and this impression rarely changes over time. Key design factors that influence user perception include color schemes, layout, white space, and the size of visual representations. These elements have been shown to significantly affect how users perceive and interact with a website [5].

4.6. Color Schemes and Layout

One of the most critical factors influencing user perception is the color scheme of a website. While there is no universal "optimal" color scheme, research shows that colors are deeply tied to human emotions. For example, [3] found that users generally prefer blue-based color schemes, which are perceived as calming and professional, while gray-based schemes are often seen as unappealing. Although orange has been shown to increase user engagement slightly, a study by [4] revealed that it is strongly disliked by many users, particularly women. For these reasons, the updated version of the SPARQL-QBE demo features blue as the primary color.
In addition to color, the layout and white space between visual elements play a crucial role in user engagement. According to [6], there is no single layout design that guarantees increased engagement. However, several key elements are consistently highlighted in research: "*(1) organization—is the website logically structured? (2) content utility—is the information useful and interesting? (3) navigation—is the website easy to navigate? (4) graphical representation—does the website use icons, contrasting colors, and multimedia effectively? (5) purpose—is the website's goal clear? (6) valid links—are all links*

functional? (7) simplicity—is the design clean and uncluttered? (8) efficiency—can users find information quickly? (9) scannability—can users quickly identify relevant information? and (10) learnability—how easy is it for users to learn how to use the website?" [6].

5. Methodology

5.1. Approach

The project was implemented using a combination of front-end technologies and third-party APIs. The Movie Database (TMDb) API was used to fetch movie posters based on query results. JavaScript and jQuery were employed to dynamically update the front-end with images, while Bootstrap ensured a responsive and visually appealing layout. To optimize performance, lazy loading was implemented using the loading attribute. The development process involved iterative testing and debugging using browser developer tools to ensure compatibility and performance.

5.2. Technologies Used

1. JavaScript: Used for dynamic content updates and interactivity.
2. jQuery: Simplified DOM manipulation and event handling.
3. CSS3 and HTML5: Provided the foundation for structuring and styling web content.
4. Bootstrap 5: Ensured a responsive and mobile-friendly design.
5. TMDb API: Fetched movie posters and metadata.
6. Lazy Loading: Optimized performance by delaying the loading of non-critical images.

5.3. Implementation Steps

1. Remove database selection:
    a. This project focuses on movie database querying. The selection of the database for deferent search options has been withdrawn for the user.
    b. The homepage has changed from the database selection to the movie search page.
2. Fetching Movie Data and Images from TMDb API:
    a. The first API fetch request using the movie name returns the TMDb ID of the movie.
    b. Subsequent API calls fetch additional information, such as posters, overviews, ratings, genres, release dates, alternative titles and trailer links.
3. Integrating Images into the Front-End:
    a. Movie posters and metadata were dynamically added to the search results using JavaScript and jQuery.

    b. Bootstrap components were used to create a visually appealing and responsive layout.
4. Using Lazy Loading to Optimize Performance:
    a. The loading="lazy" attribute was added to <img> tags to ensure that images below the viewport are loaded only when needed.
    b. This technique reduced initial page load times and improved overall performance.
5. Enhancing the UI with Bootstrap:
    a. Bootstrap's grid system and pre-designed components were used to create a clean and intuitive layout.
    b. Interactive elements, such as dropdowns and modals, were added to improve usability.
6. Data pagination:
    a. Through the TMBd API rate limitations, only 50 request calls can occur per second. To minimize delays to the eye of the user, a 12-movie results per page limit has been established.

5.4. API Integration and Data Fetching

To address the limitations of the existing SPARQL-QBE demo, such as the lack of enriched movie data, the project integrates The Movie Database (TMDb) API. This API provides access to a wealth of movie-related information, including posters, overviews, ratings, genres, release dates, and links to official trailers. However, integrating this data posed a challenge due to the absence of a universal unique identifier for movies across different databases. To resolve this, the project uses the movie title as the primary key for fetching data from TMDb. The following code snippet demonstrates the initial API call to retrieve the TMDb ID for a given movie:

```
fetch('https://api.themoviedb.org/3/search/movie?query={movie_name}&include_adult
=false&language=en-US&page=1', options)
        .then(res => res.json())
        .then(res => console.log(res))
        .catch(err => console.error(err));
```

Once the TMDb ID is retrieved, subsequent API calls fetch additional details, such as the movie poster, overview, rating, genres, release date, and trailer link. This enriched data is then dynamically integrated into the SPARQL-QBE demo, providing users with a more comprehensive and visually appealing interface.

5.5. Performance Optimization (Lazy Loading)

One of the challenges of integrating multimedia content, such as movie posters, is the potential impact on page load times. To address this, the project implements lazy loading, a technique that delays the loading of non-critical resources (e.g., images) until they are needed. This is particularly useful for images located below the viewport, as they are loaded with a lower priority. The following HTML snippet demonstrates how lazy loading is implemented:

```
<img src="image.png" loading="lazy" alt="Movie Poster">
```

By using lazy loading, the project minimizes unnecessary delays in page rendering, ensuring a smooth user experience even when dealing with large datasets. Performance tests using browser developer tools confirmed that lazy loading reduced initial page load, significantly improving the demo's efficiency.

5.6. Challenges and Solutions

1. **API Rate Limits:** The TMDb API has rate limits that restrict the number of requests per second. To address this, the project implemented a caching mechanism to store frequently accessed data locally.
2. **Responsive Design Issues:** Ensuring compatibility across different devices and screen sizes was challenging. Bootstrap’s responsive utilities were used to create a consistent experience across devices. Although because the API registration process is not optimized for mobile devices, unexpected errors may occur. For this deficiency no resources have been given to mobile optimize mobile responsiveness but only for the desktop devices.

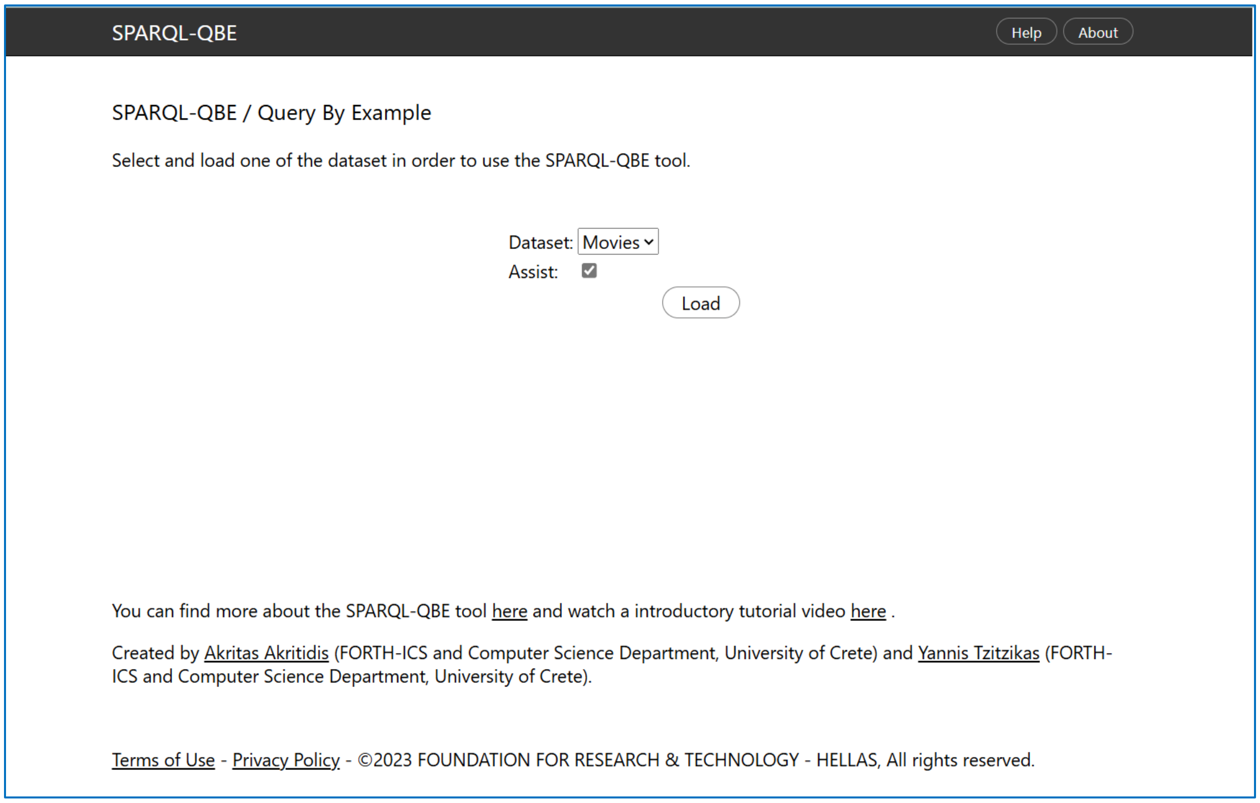
6. Results

This section presents the results of the enhancements made to the demo, focusing on the integration of multimedia content and improvements to the user interface. Screenshots of the updated demo are included to visually demonstrate the changes, and a comparison between the old and new versions is provided to highlight the key improvements.

6.1. Demo Enhancements:

- Home page:

  Initial:



SPARQL-QBE

SPARQL-QBE / Query By Example

Select and load one of the dataset in order to use the SPARQL-QBE tool.

Dataset: Movies

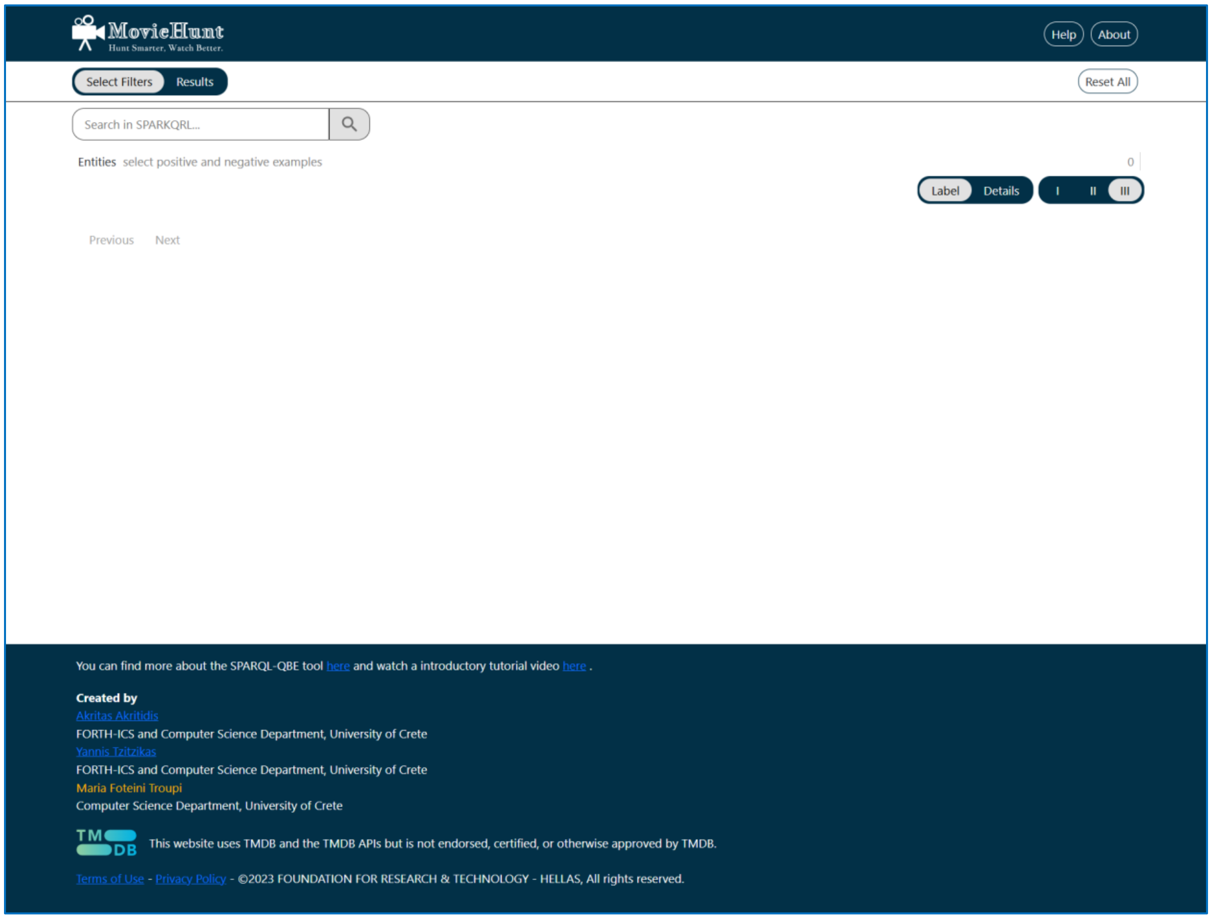
Assist: ✓

You can find more about the SPARQL-QBE tool here and watch a introductory tutorial video here .

Created by Akritas Akritidis (FORTH-ICS and Computer Science Department, University of Crete) and Yannis Tzitzikas (FORTH-ICS and Computer Science Department, University of Crete).

Terms of Use - Privacy Policy - ©2023 FOUNDATION FOR RESEARCH & TECHNOLOGY - HELLAS, All rights reserved.

Updated:



You can find more about the SPARQL-QBE tool here and watch a introductory tutorial video here .

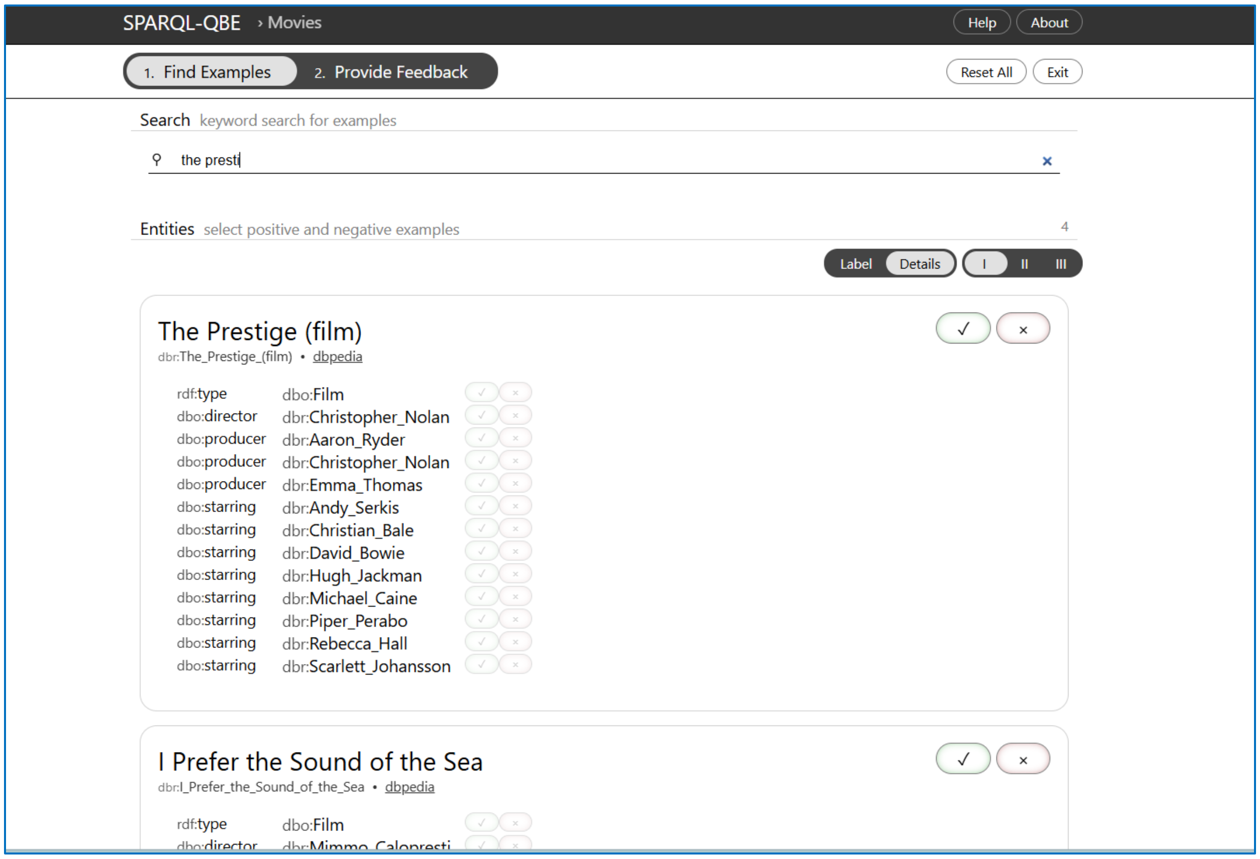
Created by
Akritas Akritidis
FORTH-ICS and Computer Science Department, University of Crete
Yannis Tzitzikas
FORTH-ICS and Computer Science Department, University of Crete
Maria Foteini Troupi
Computer Science Department, University of Crete

This website uses TMDB and the TMDB APIs but is not endorsed, certified, or otherwise approved by TMDB.

Terms of Use - Privacy Policy - ©2023 FOUNDATION FOR RESEARCH & TECHNOLOGY - HELLAS, All rights reserved.

- Examples Page: Details (view 3 results per row)

  Initial:



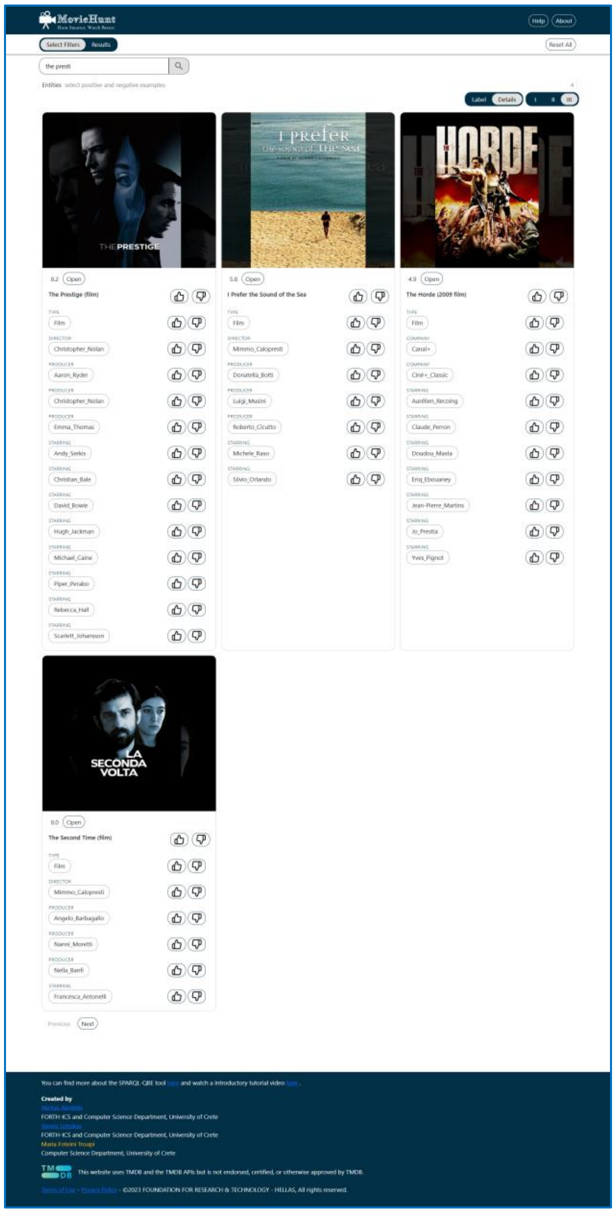
The Prestige (film)
dbr:The_Prestige_(film) • dbpedia

rdf:type	dbo:Film
dbo:director	dbr:Christopher_Nolan
dbo:producer	dbr:Aaron_Ryder
dbo:producer	dbr:Christopher_Nolan
dbo:producer	dbr:Emma_Thomas
dbo:starring	dbr:Andy_Serkis
dbo:starring	dbr:Christian_Bale
dbo:starring	dbr:David_Bowie
dbo:starring	dbr:Hugh_Jackman
dbo:starring	dbr:Michael_Caine
dbo:starring	dbr:Piper_Perabo
dbo:starring	dbr:Rebecca_Hall
dbo:starring	dbr:Scarlett_Johansson

I Prefer the Sound of the Sea
dbr:I_Prefer_the_Sound_of_the_Sea • dbpedia

rdf:type	dbo:Film
dbo:director	dbr:Mimmo_Calopresti

Updated:



the presti

Entities select positive and negative examples 4

8.2 The Prestige (film)

TYPE	Film
DIRECTOR	Christopher_Nolan
PRODUCER	Aaron_Ryder
PRODUCER	Christopher_Nolan
PRODUCER	Emma_Thomas
STARRING	Andy_Serkis
STARRING	Christian_Bale
STARRING	David_Bowie
STARRING	Hugh_Jackman
STARRING	Michael_Caine
STARRING	Piper_Perabo
STARRING	Rebecca_Hall
STARRING	Scarlett_Johansson

5.8 I Prefer the Sound of the Sea

TYPE	Film
DIRECTOR	Mimmo_Calopresti
PRODUCER	Donatella_Botti
PRODUCER	Luigi_Musini
PRODUCER	Roberto_Cicutto
STARRING	Michele_Raso
STARRING	Silvio_Orlando

4.9 The Horde (2009 film)

TYPE	Film
COMPANY	Canal+
COMPANY	Ciné+_Classic
STARRING	Aurélien_Recoing
STARRING	Claude_Perron
STARRING	Doudou_Masta
STARRING	Eriq_Ebouaney
STARRING	Jean-Pierre_Martins
STARRING	Jo_Prestia
STARRING	Yves_Pignot

8.0 The Second Time (film)

TYPE	Film
DIRECTOR	Mimmo_Calopresti
PRODUCER	Angelo_Barbagallo
PRODUCER	Nanni_Moretti
PRODUCER	Nella_Banfi
STARRING	Francesca_Antonelli

You can find more about the SPARQL-QBE tool here and watch a introductory tutorial video here .

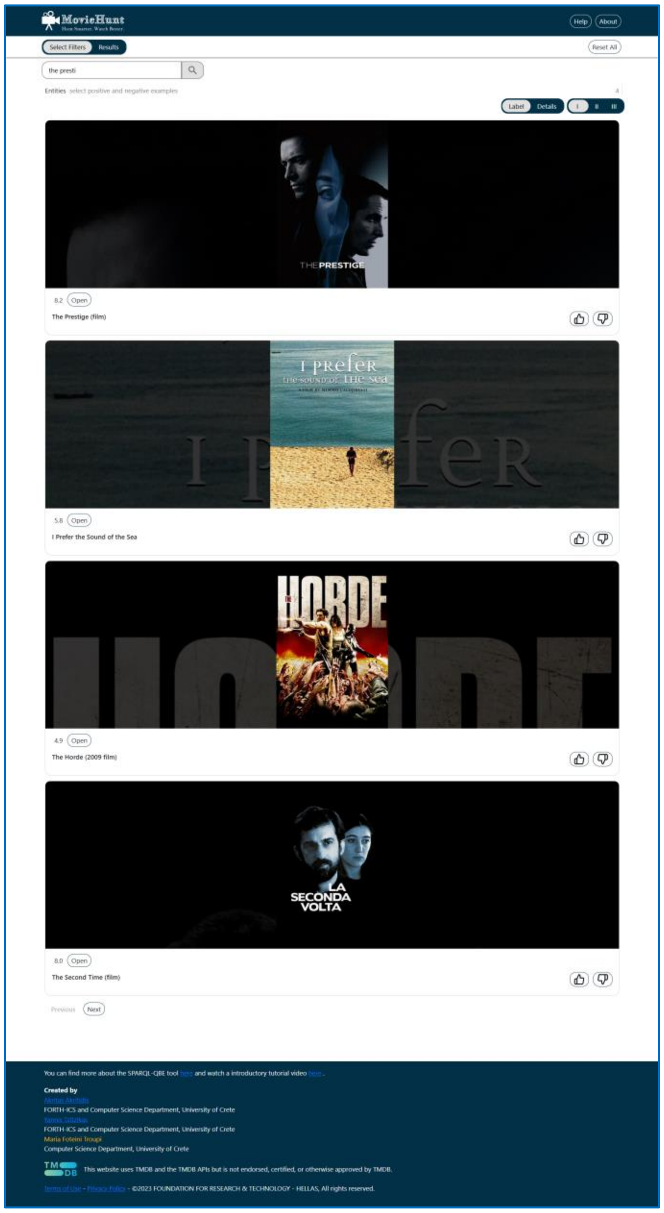
Created by
Akritas Akritidis
FORTH-ICS and Computer Science Department, University of Crete
Yannis Tzitzikas
FORTH-ICS and Computer Science Department, University of Crete
Maria Foteini Trougi
Computer Science Department, University of Crete

This website uses TMDB and the TMDB APIs but is not endorsed, certified, or otherwise approved by TMDB.

Terms of Use - Privacy Policy - ©2023 FOUNDATION FOR RESEARCH & TECHNOLOGY - HELLAS, All rights reserved.

- Find Examples Page: Label (view 1 result per row)
  Initial:

Updated:



the presti

Entities select positive and negative examples 4

8.2 The Prestige (film)

5.8 I Prefer the Sound of the Sea

4.9 The Horde (2009 film)

8.0 The Second Time (film)

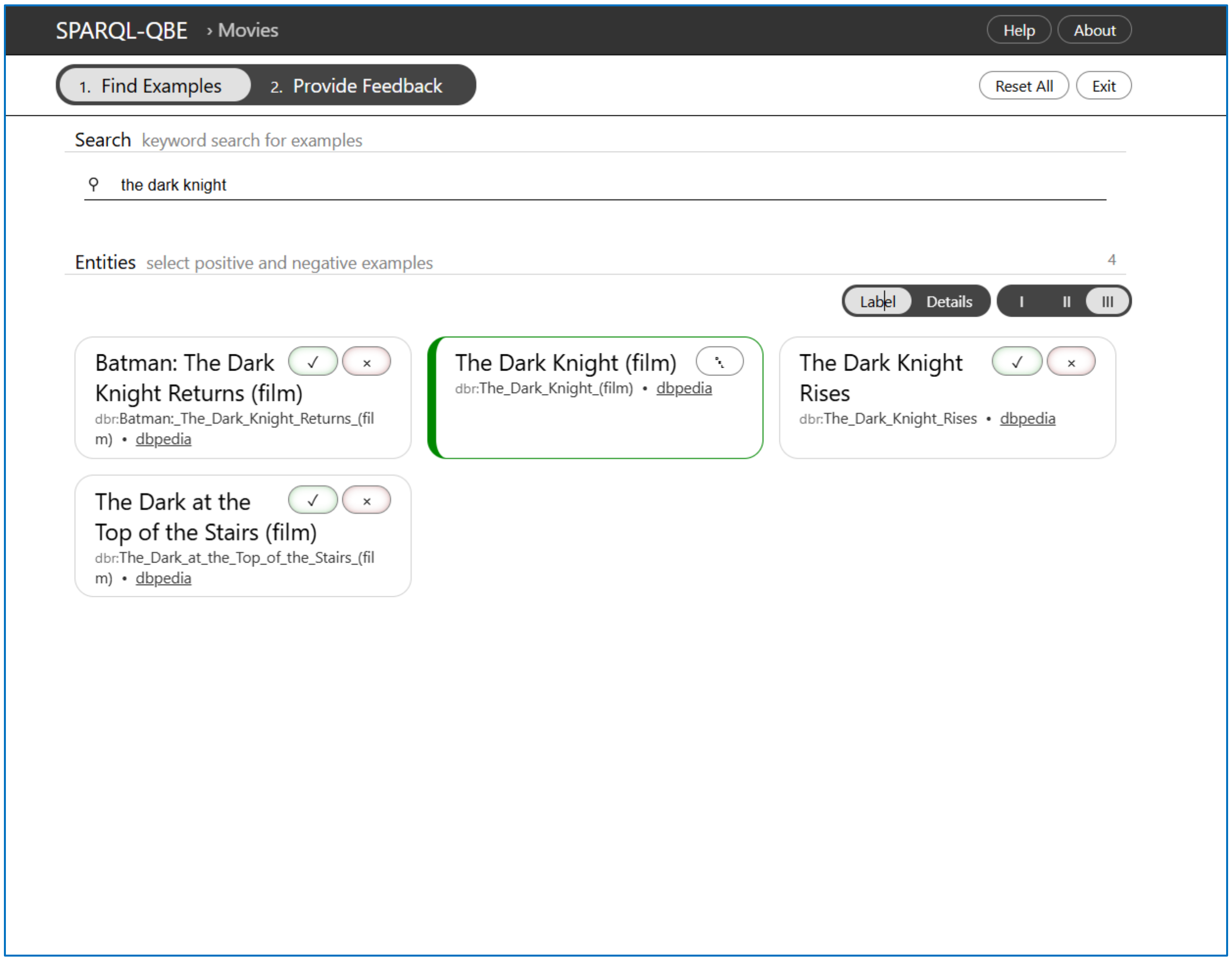
You can find more about the SPARQL-QBE tool here and watch a introductory tutorial video here .

Created by
Akritas Akritidis
FORTH-ICS and Computer Science Department, University of Crete
Yannis Tzitzikas
FORTH-ICS and Computer Science Department, University of Crete
Maria Foteini Trougi
Computer Science Department, University of Crete

This website uses TMDB and the TMDB APIs but is not endorsed, certified, or otherwise approved by TMDB.

Terms of Use - Privacy Policy - ©2023 FOUNDATION FOR RESEARCH & TECHNOLOGY - HELLAS, All rights reserved.

- Find Examples Pages: Label (view 3 results per row)
  Initial:



Search keyword search for examples

the dark knight

Entities select positive and negative examples 4

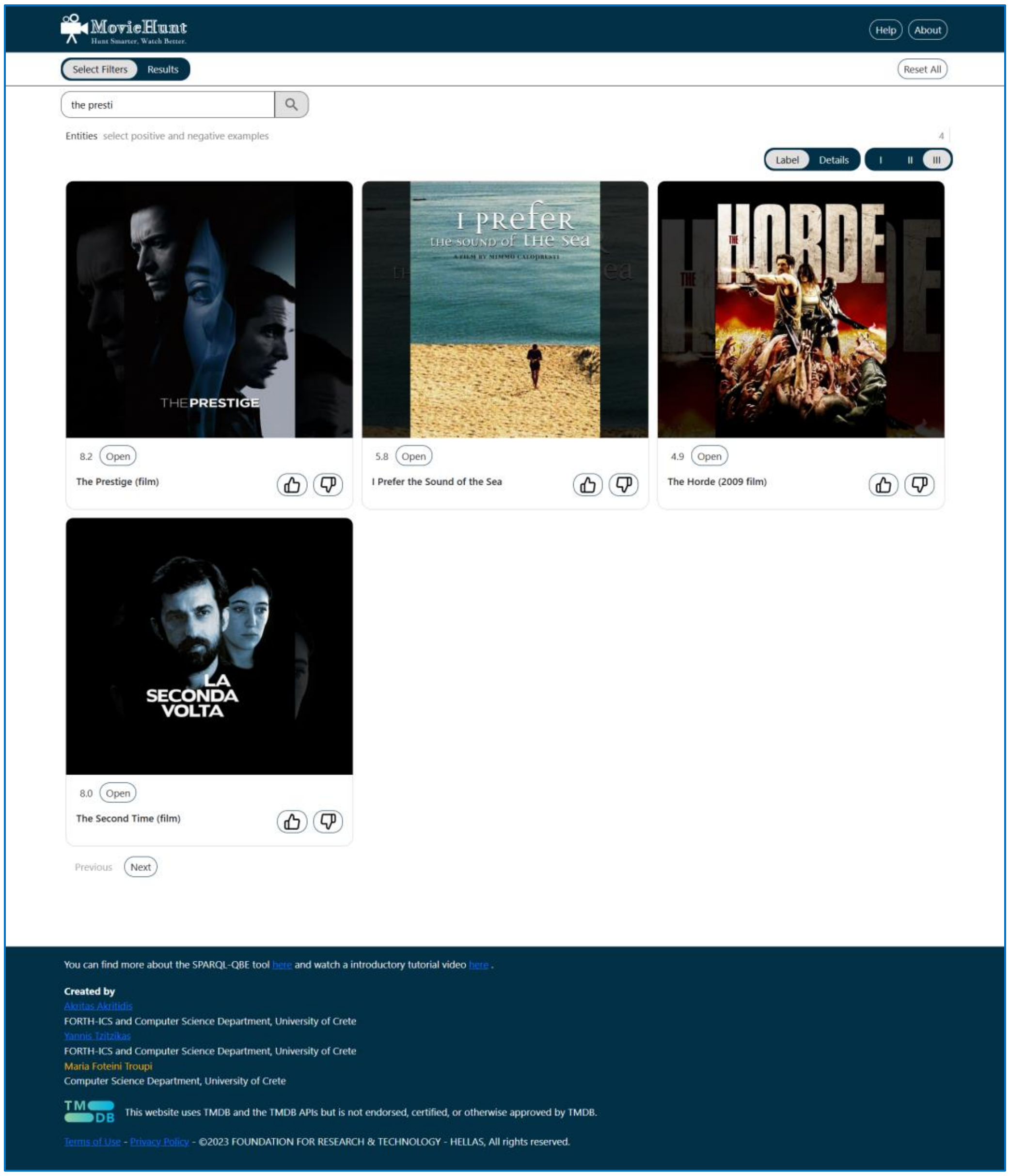
Batman: The Dark Knight Returns (film)
dbr:Batman:_The_Dark_Knight_Returns_(film) • dbpedia

The Dark Knight (film)
dbr:The_Dark_Knight_(film) • dbpedia

The Dark Knight Rises
dbr:The_Dark_Knight_Rises • dbpedia

The Dark at the Top of the Stairs (film)
dbr:The_Dark_at_the_Top_of_the_Stairs_(film) • dbpedia

Updated:



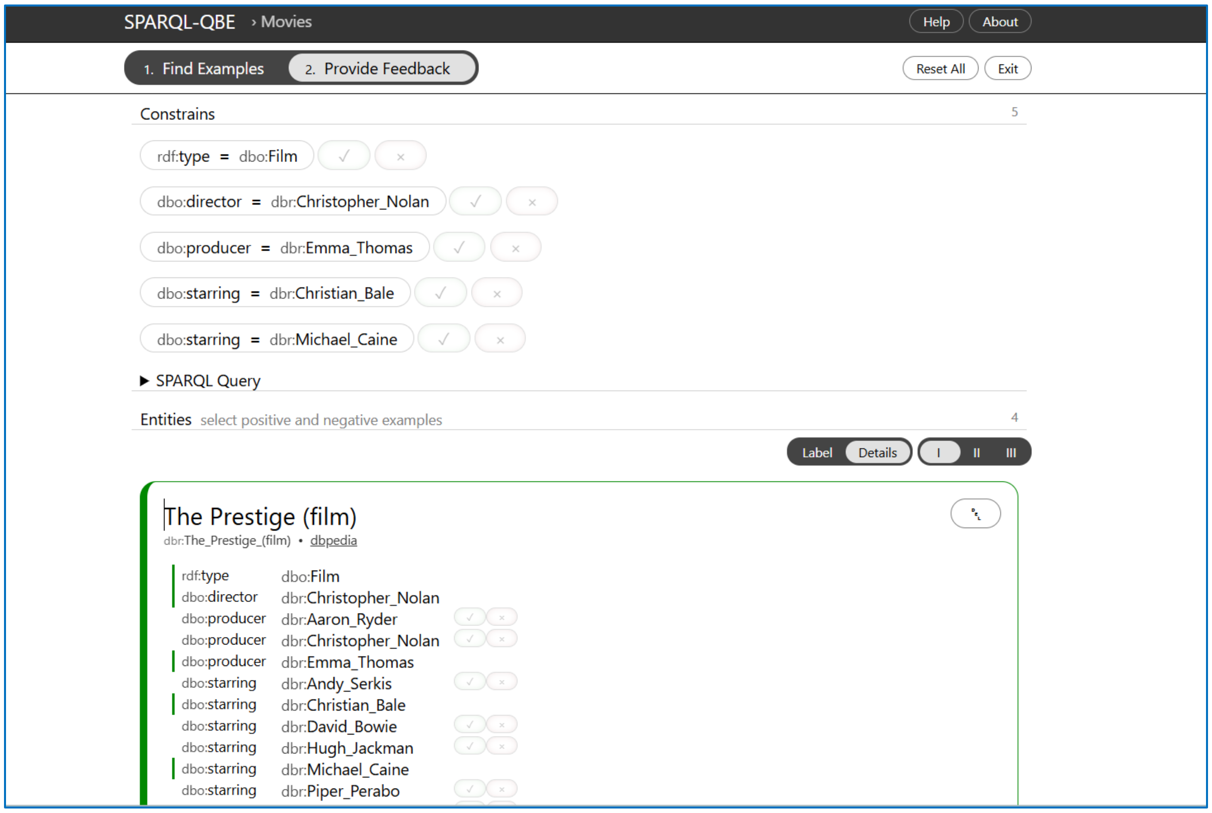
You can find more about the SPARQL-QBE tool here and watch a introductory tutorial video here .

Created by
Akritas Akritidis
FORTH-ICS and Computer Science Department, University of Crete
Yannis Tzitzikas
FORTH-ICS and Computer Science Department, University of Crete
Maria Foteini Troupi
Computer Science Department, University of Crete

This website uses TMDB and the TMDB APIs but is not endorsed, certified, or otherwise approved by TMDB.

Terms of Use - Privacy Policy - ©2023 FOUNDATION FOR RESEARCH & TECHNOLOGY - HELLAS, All rights reserved.

- Provide Feedback Page: Details (view 1 result per row)
  Initial:


SPARQL-QBE › Movies

Help About

1. Find Examples 2. Provide Feedback

Reset All Exit

Constrains 5

rdf:type = dbo:Film ✓ ×

dbo:director = dbr:Christopher_Nolan ✓ ×

dbo:producer = dbr:Emma_Thomas ✓ ×

dbo:starring = dbr:Christian_Bale ✓ ×

dbo:starring = dbr:Michael_Caine ✓ ×

▶ SPARQL Query

Entities select positive and negative examples 4

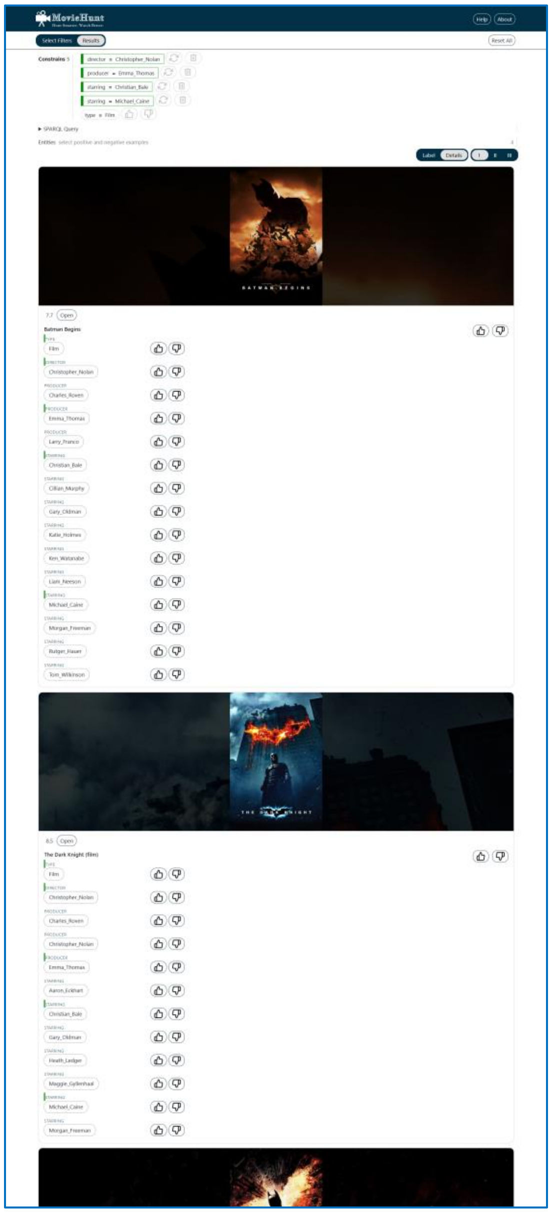
Label Details I II III

The Prestige (film)

dbr:The_Prestige_(film) • dbpedia

rdf:type	dbo:Film
dbo:director	dbr:Christopher_Nolan
dbo:producer	dbr:Aaron_Ryder
dbo:producer	dbr:Christopher_Nolan
dbo:producer	dbr:Emma_Thomas
dbo:starring	dbr:Andy_Serkis
dbo:starring	dbr:Christian_Bale
dbo:starring	dbr:David_Bowie
dbo:starring	dbr:Hugh_Jackman
dbo:starring	dbr:Michael_Caine
dbo:starring	dbr:Piper_Perabo

Updated:



Constrains 5
director = Christopher_Nolan
producer = Emma_Thomas
starring = Christian_Bale
starring = Michael_Caine
type = Film

Entities select positive and negative examples

7.7
Batman Begins
TYPE Film
DIRECTOR Christopher_Nolan
PRODUCER Charles_Roven
PRODUCER Emma_Thomas
PRODUCER Larry_Franco
STARRING Christian_Bale
STARRING Cillian_Murphy
STARRING Gary_Oldman
STARRING Katie_Holmes
STARRING Ken_Watanabe
STARRING Liam_Neeson
STARRING Michael_Caine
STARRING Morgan_Freeman
STARRING Rutger_Hauer
STARRING Tom_Wilkinson

8.5
The Dark Knight (film)
TYPE Film
DIRECTOR Christopher_Nolan
PRODUCER Charles_Roven
PRODUCER Christopher_Nolan
PRODUCER Emma_Thomas
STARRING Aaron_Eckhart
STARRING Christian_Bale
STARRING Gary_Oldman
STARRING Heath_Ledger
STARRING Maggie_Gyllenhaal
STARRING Michael_Caine
STARRING Morgan_Freeman

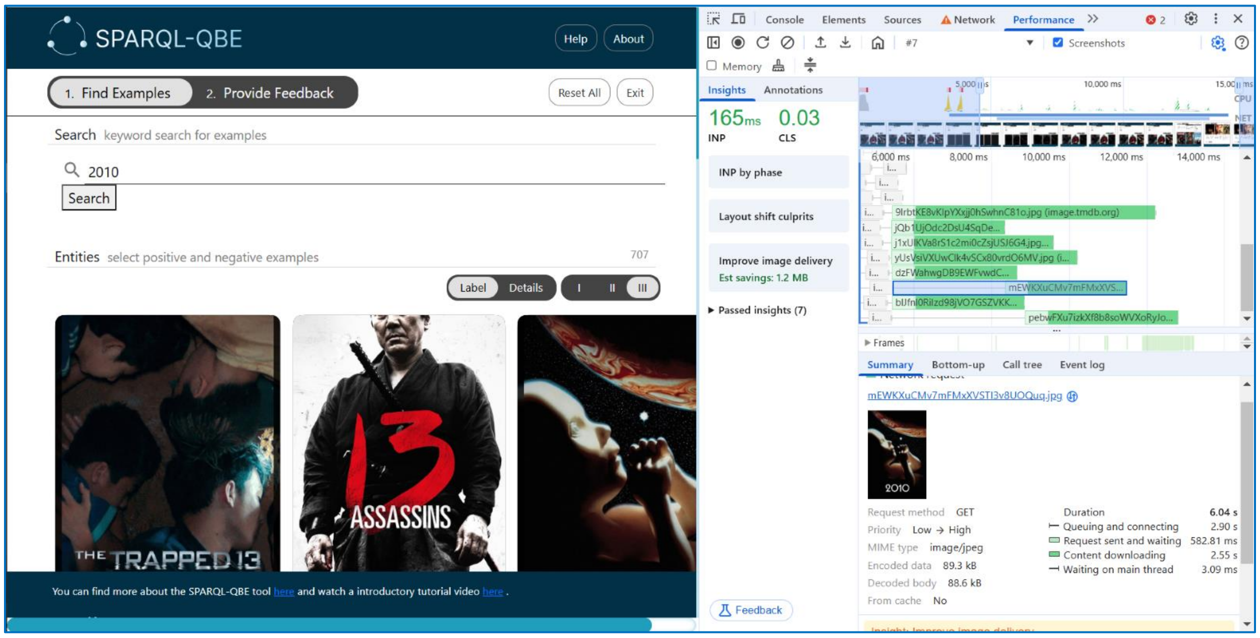
6.2. Performance

To prove the effectiveness of the lazy loading of the image the developer tools of google chrome [18] are used. Images before and after the implementation are compared and the results are explained in two sections.

6.2.1. Default image loading

The 'screenshot 1' shows that an image of a movie poster that is at the very first search results is prioritized as 'low to high' and the request to render the image it starts after

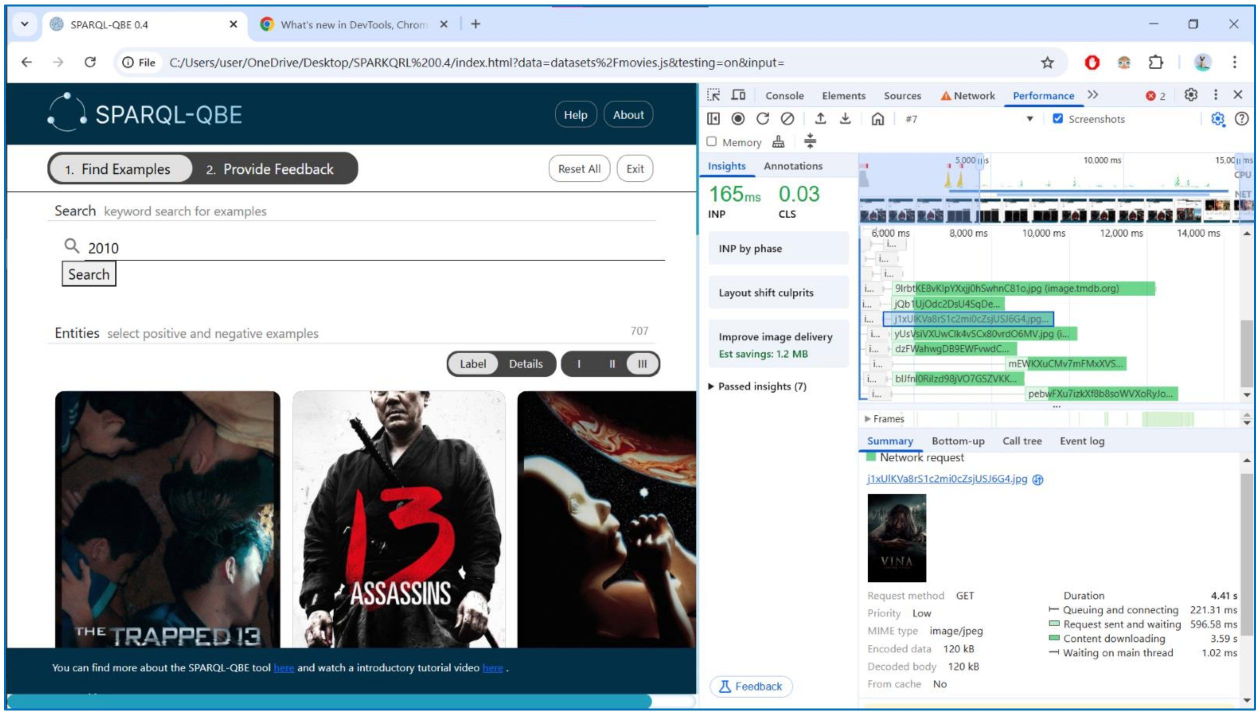
8,000ms approximately. The “screenshot 2” shows that an image of a movie poster that is below the fold and not yet visible by the user and labeled with a low priority renders after 6,000ms approximately. that proves the unorder rendering of images despite their priority.



mEWKXuCMv7mFMxXVSTl3v8UOQuq.jpg

Request method	GET	Duration	6.04 s
Priority	Low → High	Queuing and connecting	2.90 s
MIME type	image/jpeg	Request sent and waiting	582.81 ms
Encoded data	89.3 kB	Content downloading	2.55 s
Decoded body	88.6 kB	Waiting on main thread	3.09 ms
From cache	No		

SCREENSHOT1 ”ABOVE THE FOLD” WITH RELATIVELY HIGH PRIORITY



Network request

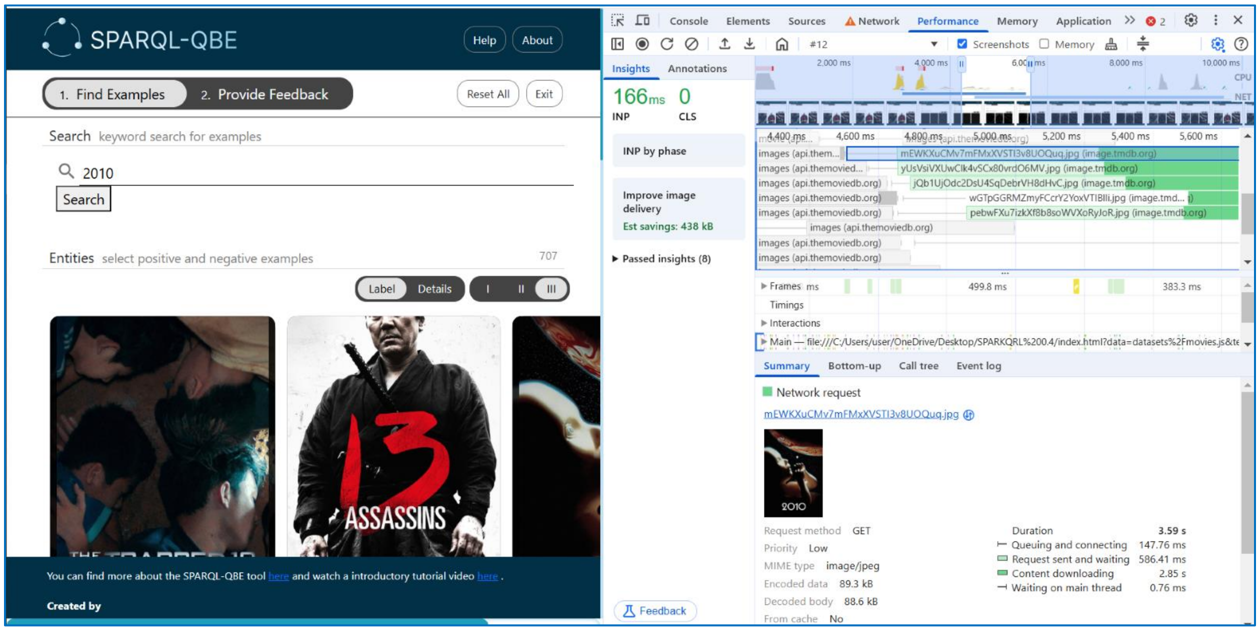
j1xUlKVa8rS1c2mi0cZsjUSJ6G4.jpg

Request method	GET	Duration	4.41 s
Priority	Low	Queuing and connecting	221.31 ms
MIME type	image/jpeg	Request sent and waiting	596.58 ms
Encoded data	120 kB	Content downloading	3.59 s
Decoded body	120 kB	Waiting on main thread	1.02 ms
From cache	No		

SCREENSHOT 2 ”BELOW THE FOLD” WITH RELATIVELY LOW PRIORITY

6.2.2. Lazy Loading

Below the screenshots “3”, “4”and “5” have been taken with the lazy loading implementation using the DevTools. It’s visible that the very first images that the user sees at the search results are the images that are the browser fetches and renders first.



You can find more about the SPARQL-QBE tool here and watch a introductory tutorial video here .

Created by

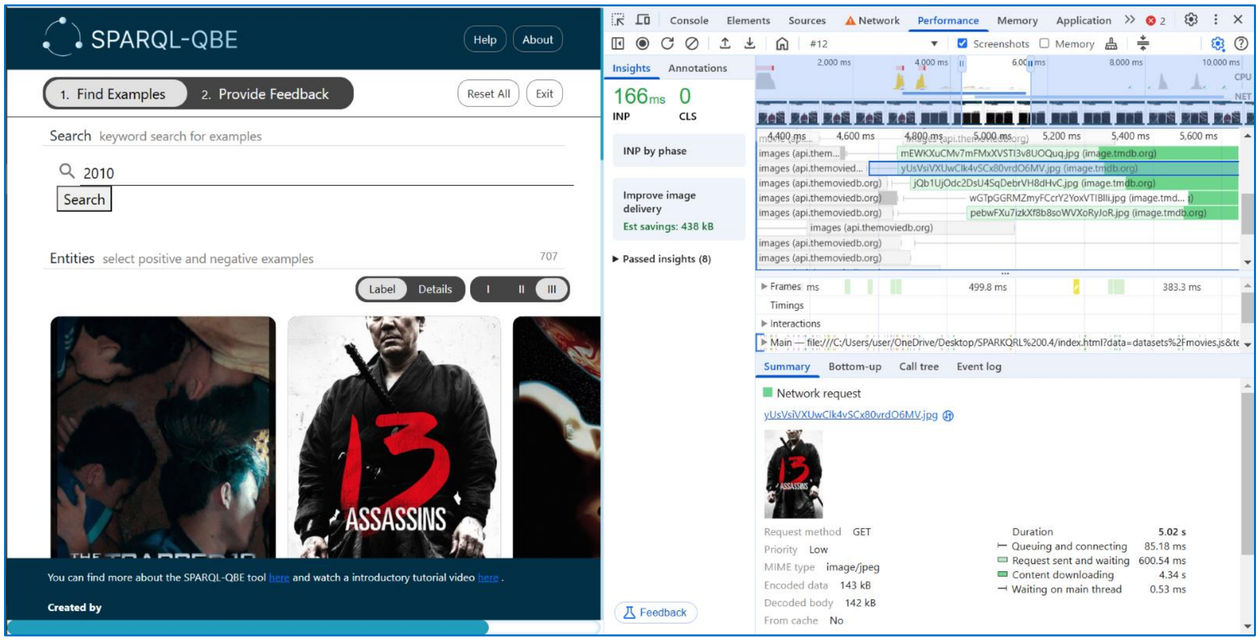
```
images (api.them...      mEWKXuCMv7mFMxXVSTI3v8UOQuq.jpg (image.tmdb.org)
images (api.themovied... yUsVsiVXUwClk4vSCx80vrdO6MV.jpg (image.tmdb.org)
images (api.themoviedb.org) jQb1UjOdc2DsU4SqDebrVH8dHvC.jpg (image.tmdb.org)
images (api.themoviedb.org) wGTpGGRMZmyFCcrY2YoxVTIBlli.jpg (image.tmd...)
images (api.themoviedb.org) pebwFXu7izkXf8b8soWVXoRyJoR.jpg (image.tmdb.org)
images (api.themoviedb.org)
images (api.themoviedb.org)
images (api.themoviedb.org)

Main — file:///C:/Users/user/OneDrive/Desktop/SPARKQRL%200.4/index.html?data=datasets%2Fmovies.js&te

Network request
mEWKXuCMv7mFMxXVSTI3v8UOQuq.jpg

Request method  GET          Duration                    3.59 s
Priority  Low                Queuing and connecting      147.76 ms
MIME type  image/jpeg        Request sent and waiting    586.41 ms
Encoded data  89.3 kB        Content downloading         2.85 s
Decoded body  88.6 kB        Waiting on main thread      0.76 ms
From cache  No
```

SCREENSHOT 3



Search keyword search for examples

2010

Entities select positive and negative examples 707

You can find more about the SPARQL-QBE tool here and watch a introductory tutorial video here .

Created by

166ms INP 0 CLS

INP by phase

Improve image delivery
Est savings: 438 kB

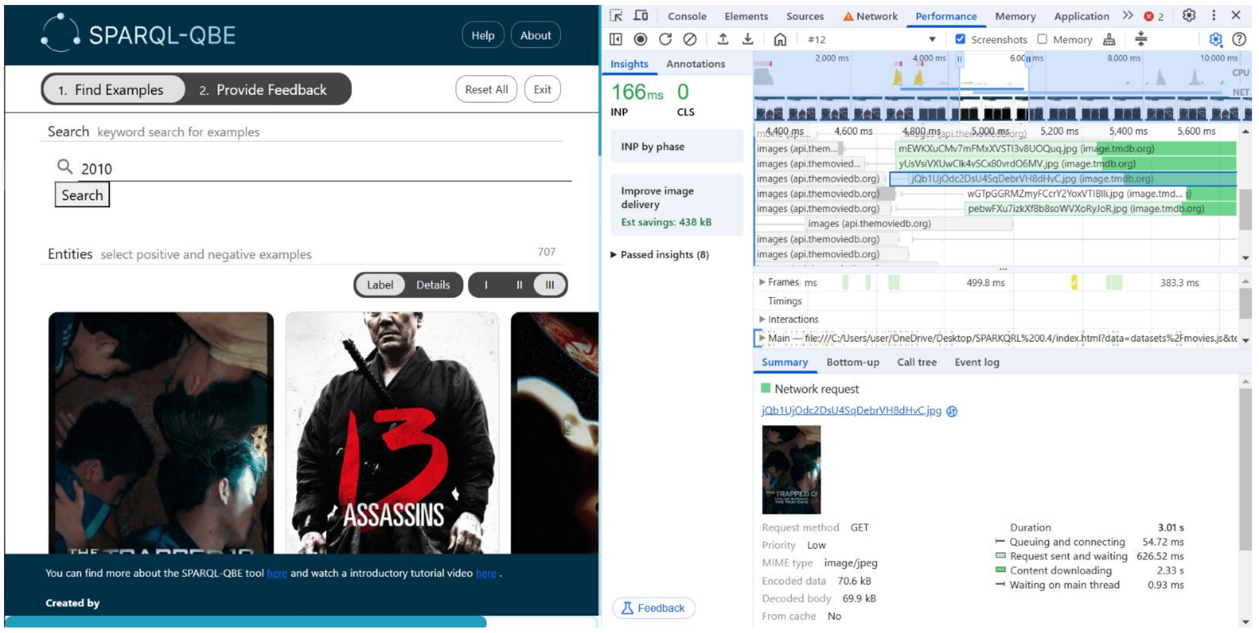
Passed insights (8)

Main — file:///C:/Users/user/OneDrive/Desktop/SPARKQRL%200.4/index.html?data=datasets%2Fmovies.js&te

Network request
yUsVsiVXUwClk4vSCx80vrdO6MV.jpg

Request method	GET	Duration	5.02 s
Priority	Low	Queuing and connecting	85.18 ms
MIME type	image/jpeg	Request sent and waiting	600.54 ms
Encoded data	143 kB	Content downloading	4.34 s
Decoded body	142 kB	Waiting on main thread	0.53 ms
From cache	No		

SCREENSHOT 4



Search keyword search for examples

2010

Entities select positive and negative examples 707

You can find more about the SPARQL-QBE tool here and watch a introductory tutorial video here .

Created by

166ms INP 0 CLS

INP by phase

Improve image delivery
Est savings: 438 kB

Passed insights (8)

Main — file:///C:/Users/user/OneDrive/Desktop/SPARKQRL%200.4/index.html?data=datasets%2Fmovies.js&te

Network request
jQb1UjOdc2DsU4SqDebrVH8dHvC.jpg

Request method	GET	Duration	3.01 s
Priority	Low	Queuing and connecting	54.72 ms
MIME type	image/jpeg	Request sent and waiting	626.52 ms
Encoded data	70.6 kB	Content downloading	2.33 s
Decoded body	69.9 kB	Waiting on main thread	0.93 ms
From cache	No		

SCREENSHOT 5

6.3. Code Statistics

The development of this project involved significant changes to the codebase. Below is a summary of the code changes between the **first commit** and the **last commit**:

File	Lines Added	Lines Deleted	Net Change
index.html	316	200	+116
main.js	191	100	+91
moviesImgs.js	122	0	+122
style.css	401	181	+220
Total	**749**	**281**	**+468**

- **index.html**:
  - Lines Added: 316
  - Lines Deleted: 200
  - Net Change: +116
  - Changes: Enhanced the structure and functionality of the HTML file, including the integration of new UI components and the TMDb API.
- **main.js**:
  - Lines Added: 191
  - Lines Deleted: 100
  - Net Change: +91
  - Changes: Refactored and expanded the core functionality, implementing features such as dynamic data fetching, lazy loading, and improved user interactions.
- **moviesImgs.js**:
  - Lines Added: 122
  - Lines Deleted: 0
  - Net Change: +122
  - Changes: Introduced functionality to handle movie posters and images fetched from the TMDb API.
- **style.css**:
  - Lines Added: 401
  - Lines Deleted: 181
  - Net Change: +220
  - Changes: Updated the layout and design to improve responsiveness, visual appeal, and user experience.

7. Conclusion

This thesis has successfully demonstrated the importance of integrating visual elements and enriched data into the SPARQL-QBE demo, transforming it into a more intuitive and engaging tool for querying movie databases. By leveraging modern front-end technologies such as JavaScript, jQuery, CSS3, HTML5, and Bootstrap, the project enhanced the user interface, making it more visually appealing and user-friendly. The integration of The Movie Database (TMDb) API allowed for the inclusion of movie posters, ratings, genres, release dates, and trailers, significantly improving the completeness and utility of search results.

One of the key achievements of this project is the implementation of lazy loading, which optimized performance by reducing page load times. This ensures a smooth user experience, even when dealing with large datasets. Additionally, the project addressed challenges such as API rate limits and identifier mismatches by developing a robust mechanism for data integration and mapping.

The enhanced SPARQL-QBE demo not only improves usability for non-expert users but also serves as a case study for the importance of visual design in data-driven applications. By combining aesthetic value with functional utility, the project highlights how user-friendly interfaces can make complex systems more accessible and engaging.

7.1. Key Contributions

1. **Visual Enhancements:** The integration of movie posters and enriched metadata provides users with a more engaging and informative experience.
2. **Performance Optimization:** The use of lazy loading significantly improved page load times, ensuring a smooth and efficient user experience.
3. **User-Centered Design:** The project applied principles of user-centered design, such as clear organization, intuitive navigation, and visually appealing layouts, to create a more accessible interface.

7.2. Limitations

While the project achieved its objectives, it is important to acknowledge its limitations:

1. **Dependency on TMDb API:** The demo relies heavily on the TMDb API for fetching movie data. Any changes to the API’s structure or availability could impact the demo’s functionality.
2. **Scalability:** Although lazy loading improves performance, the demo may face scalability issues with very large datasets.

3. **Poster Mismatches:** Since the use of the movie name has the only identifier available for the initial search in the TMDb API to retrieve movie's ids some mismatches are inevitable. This occurs in movies that share the same title. The system can't identify the requested one and retrieves the first in the results leading to incomplete search result enchantment or to misleading results.
4. **Unpredictable Image Fetching Order:** While lazy loading ensures that images are loaded in a prioritized manner based on their position in the viewport, there is no way to guarantee the order in which images are fetched from the TMDb API. This is because the fetching order depends on the response time of TMDb's database, which can vary based on server load, network conditions, and other external factors. As a result, users may occasionally experience delays or inconsistencies in the display of movie posters.

7.3. Future Work

Like any other software system, this project will require constant improvements and maintenance to ensure its continued functionality, performance, and relevance. Software development is an iterative process, and there will always be areas for enhancement, bugs to fix, and new features to implement as user needs and technologies evolve. With this in mind, some key improvements for future work include:

1. **Integration of Additional APIs:** Incorporating data from other sources, such as IMDb or Rotten Tomatoes, could provide even richer and more comprehensive search results.
2. **Support for More Entity Types:** Extending the demo to include images and metadata for other entities, such as actors, directors, and genres, could further enhance its utility.
3. **Accessibility Improvements:** Future work could focus on improving accessibility for users with disabilities, such as by adding alt text for images and ensuring compatibility with screen readers.
4. **Machine Learning for Data Matching:** Exploring machine learning techniques to improve the accuracy of data matching and integration could address the issue of identifier mismatches.
5. **Caching Mechanisms:** Implementing a caching system to store frequently accessed movie data locally could reduce dependency on the TMDb API and improve the consistency of image fetching.

7.4. Final Thoughts

This project underscores the value of combining visual design with functional utility in creating user-friendly interfaces for complex systems. By enriching the SPARQL-QBE demo with visual elements and additional information, the project has made significant strides in improving the usability and accessibility of semantic web technologies. The results highlight the importance of considering both aesthetic and functional aspects in the design of data-driven applications, paving the way for future innovations in this field.

8. Appendix 1 - Project Timeline

1. **Understanding the System (1 Week)**
   The initial phase involved experimenting with the existing SPARQL-QBE demo and performing basic reverse engineering to gain a comprehensive understanding of its structure and functionality. This phase included analyzing the codebase, identifying key components, and familiarizing myself with the system's architecture. The goal was to establish a solid foundation for subsequent development efforts.
2. **Feature Experimentation (2 Weeks)**
   Over the next two weeks, I focused on experimenting with the addition of new features to evaluate the system's flexibility and scalability. This phase involved testing various enhancements, such as integrating external APIs and incorporating visual elements, to determine whether the system could support these changes without compromising its core functionality. The experimentation process was critical in identifying the system's limitations and opportunities for improvement.
3. **Rebuilding the Project (1.5 Weeks)**
   To mitigate the risk of introducing bugs or disrupting existing functionality, I decided to rebuild the project from the version provided to me. This approach ensured that the system remained stable and that any issues could be easily identified and resolved. By starting fresh, I was able to implement enhancements in a controlled and systematic manner, minimizing potential risks.
4. **Implementing Fixes and Suggestions (0.5 Weeks)**
   After presenting the initial version of the project to my supervisor, I dedicated half a week to addressing feedback and implementing suggested improvements. This phase involved refining the user interface, optimizing the user experience, and ensuring that the system met the project's objectives. The feedback-driven improvements were essential in aligning the project with academic and functional expectations.
5. **Writing the Initial Thesis Report (1 Week)**The following week was devoted to drafting the initial version of the thesis report. This phase involved documenting the

project's goals, methodology, results, and conclusions, as well as reflecting on the challenges and lessons learned throughout the development process. The report served as a comprehensive record of the project's progress and outcomes.

6. **Finalizing the Thesis (0.5 Weeks)**The final half-week was spent revising the thesis based on feedback from my supervisor. This phase included improving the clarity and structure of the report, incorporating additional details, and ensuring that the thesis met all academic requirements. The revisions were critical in producing a polished and professional document.
7. **Publishing the Project (2 Days)**The last phase involved publishing the project, which included uploading the code and datasets to remote servers, deploying the project, and resolving any complications that arose during the data transfer process. This phase also addressed technical challenges, such as configuring new IP addresses and ensuring the system's accessibility. The successful deployment marked the completion of the project.

9. References

9.1. Literature

1. Querying Knowledge Graphs through Positive and Negative Examples and Feedback https://users.ics.forth.gr/~tzitzik/publications/Tzitzikas_2024-JIIS.pdf
2. The phenomenon of the ever-increasing impact of visual content on search engines' websites ranking and the indicated SEO adjustments The phenomenon of the ever-increasing impact of visual content on search engines' websites ranking and the indicated SEO adjustments
3. The impact of colour on Website appeal and users' cognitive processes https://doi.org/10.1016/j.displa.2010.12.002
4. Web user Experience and Consumer behaviour: The Influence of Colour, Usability and Aesthetics on the Consumer Buying behaviour. ISSN: 0193 - 4120 Page No. 16592 - 16600
5. Impact of website visual design on user experience and website evaluation: The sequential mediating roles of usability and pleasure https://doi.org/10.1080/0267257X.2022.2085315
6. A Literature Review: Website Design and User Engagement Online J Commun Media Technol. 2016 July ; 6(3): 1–14
7. Metzger, S., Schenkel, R., Sydow, M.: Qbees: query by entity exam ples. In: Proceedings of the 22nd ACM International Conference on Information & Knowledge Management (2013). https://doi.org/10.1007/s10844-017-0443-x

8. Li, H., Chan, C.-Y., Maier, D.: Query from examples: An iterative, data-driven approach to query construction. Proceedings of the VLDB Endowment 8(13) (2015). https://doi.org/10.14778/2831360.2831369
9. Bootstrap as a tool for web development and graphic optimization on mobile devices https://doi.org/10.1007/978-3-030-68080-0_22

9.2. Tools and Technologies

10. EC Demonstration https://www.youtube.com/watch?v=k9OZ3vSaUmQ
11. Figma: The Collaborative Interface Design Tool https://www.figma.com/
12. JavaScript Programming Language https://developer.mozilla.org/en-US/docs/Web/JavaScript
13. JQuery API https://api.jquery.com/
14. Bootstrap toolkit https://getbootstrap.com/
15. The Movie Database https://developer.themoviedb.org/docs/getting-started
16. W3schools Learning Platform https://www.w3schools.com/
17. Coolors - Color Palette Generator https://coolors.co/
18. HTML YouTube Videos https://www.w3schools.com/html/html_youtube.asp
19. PageSpeed Insights https://developers.google.com/speed/docs/insights/v5/about
20. Browser-level image lazy loading https://web.dev/articles/browser-level-image-lazy-loading
21. Chrome DevTools https://developer.chrome.com/docs/devtools