Towards Semantic Search in Building Metadata: A System Exploration

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Aishwarya Gavili

Spring, 2022

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Hongning Wang, Department of Computer Science

Towards Semantic Search in Building Metadata: A System Exploration

Introduction

The below report outlines the changes I have implemented and the research I have done on the UVAMonitor system as well as semantic search in building metadata as a whole. It covers changes in event name handling, search bar functionality, and testing/system exploration in relation to minimizing API calls and more complex queries.

Event Name Handling

Upon joining this project, my goal and main tasks were to figure out how to make existing implementations more efficient and resistant to bugs. The first of these tasks was to modify the way events were handled when they were created and saved. More specifically, one bug the system faced was with how certain events were saved and were accessible to view later and how certain events weren't. This boiled down to how events were characterized as events, which previously was set by the system if the event name had the keyword 'event' in it. However, the way the event name was extracted to search through the event table for a specific event caused event names to be erased completely as the original implementation replaced all occurrences of the keyword 'event'. More specifically, if the event name was 'event2', the system would modify the name to be '2' and the event would be unsaved and be inaccessible. With my implementation, all events regardless of their names could be saved and accessed.

```
var lastIndex = query.lastIndexOf('event');
query = query.substring(0, lastIndex);
var eventName = query.trim();
```

Figure 1. New Implementation

```
var eventName = query.replace('event', '').trim();
```

Figure 2. Previous Implementation

Search Bar Functionality

To reduce complexity of design from a useability and code readability standpoint, I had implemented functionality changes with regards to the search bar. Previously, the search bar could search events as well as queries. Over the past two semesters, I worked on reducing the search bar's functionality to only be able to search queries, by splitting up the search() function into two separate functions in /monitor-api/routes/api.js: searchQuery() and search(). This involved removing code in the search() function that checked if a query was an event or not and moving that code into the new searchEvent() function. Additionally, a lot of redundant and unnecessary code within the original search() function was removed with the assistance of alum Max Zheng. The below image displays the new searchEvent() function. The changes made in the search() function are too large to be displayed in a singular image, but these changes are accessible through the ag5yy and ag5yy_f21 branches on Github.

```
router.get("/searchEvent/:query", async function (req, res, next) {

  let globalLog = "";

  tStart = now();
  res.header("Access-Control-Allow-Origin", "*");

  var query = req.params.query;

  var data = await queryer.getEventData(req, query);
  tEnd = now();

  console.log(globalLog);
  res.json({ data: data, log: globalLog });

});
```

Figure 3. searchEvent() function

Additionally, to reflect the changes made in the backend in the front end, I modified /monitor-front/src/app/chart/chart.component.html. More specifically, I changed the script so that searching queries from the search bar only invoked the search() function and clicking on events from the side bar with existing events only invoked the searchEvent() function. In other

words, the search() function was split into two functions of which one was used to search events and one was used to search queries. The image below demonstrates the changes.

```
<a (click)="searchEvent(event); snav.close()" *ngFor="let event of events"
  mat-list-item>{{event + ' event'}}</a>
```

Figure 4. Front-end searchEvent() invocation

Here, it is evident that displaying an event's charts from the side navigation bar calls the searchEvent() function only on the event name without the extra 'event' keyword, which the previous implementation had not done. Additionally, the searchEvent() endpoint was added to two typescript files: chart.service.ts and chart.component.ts.

Testing and System Exploration

Due to the size of the system and its multiple dependencies, I had spent most of my second semester testing out certain functionalities and understanding how they worked and interacted with each other. The two tasks that I focused on were converting the searching of queries from 2 API calls to 1 API call and allowing queries to handle multiple keywords.

1. *Reducing searching queries from 2 API calls to 1 API call*

   When a query is searched through the search bar, it goes through two endpoints: translateQuery() and search(). And the translateQuery() function returns a parsed version of the query using the parser.js module. This new parsed query is then passed into the search() function where the queries are issued and the results are pulled from the database and displayed. The issue existed in the search() function where the query was parsed again after being fed in the parsed query from translateQuery(), resulting in a lot of redundancy in code between the two functions. This task is still a work in progress but

below I outline some of the steps I took to and what I learned in terms of what worked and what didn't.

My first step was moving all of the code in translateQuery() inside the search() function in api.js. It should be noted this change is on a local branch and has not been pushed to Github yet. By doing so the original query could be parsed, undergo stopword removal and wordnet expansion, and finally be issued all in one function. However, doing so was not enough as I found that a call to the search endpoint triggered a call to translateQuery() because of how they were nested, which is evident in the search(query) function in /monitor-front/src/app/chart/chart.component.ts. To overcome this, I had removed the call to the translateQuery() and directly passed the original query into the search() function.

```
//const translated = this.chartService.translateQuery(query, this).then(translated => {
  // const query = translated['newQuery'];
   const query2 = query;
     this.isEvent = false;
     this.event = null;
     this.queryTimestamp = timestamp;

     this.swiperMode = false;

    this.chartService.search(query2, this);
```

Figure 5. Removal of translateQuery intermediary API call

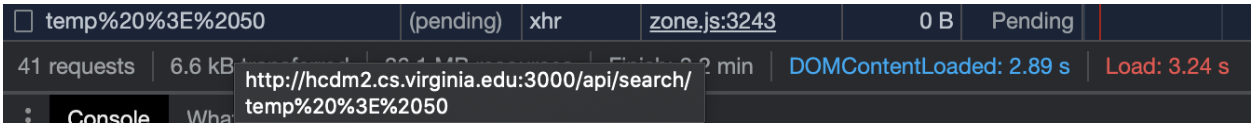

Figure 6. Browser network trace of searching a query (translateQuery API call does not exist)

After bypassing the translateQuery() API call, the next bug that I had to address was actually retrieving the charts and data associated with the query. This led me to do some research with what Promises and asynchronous functions were within Javascript; something I was not too familiar with before. More specifically, I needed to understand why the

translateQuery API call consisted of a Promise and why both the search() function and translateQuery() function were asynchronous functions. I began with first making the search function a synchronous function and experimenting with the removal of the Promise within the translateQuery functions within, however doing so triggered other errors related to a 404 response, prompting me to revert the changes. Currently, this error is still being debugged.

2. *Handling multiple keywords for more complex queries*

    The second task I worked on was exploring how the search bar ingested queries and if there was a way to allow it to handle multiple keywords as opposed to just one. For this task, I mainly delved into what the parseQuery() function, within the parser.js module, was outputting. More specifically, I wanted to see how 'parsed' or parseQuery()'s return value was being split up amongst 'operatorSearchTerms', 'attributes', and 'attributeValue'. To do this, I essentially outputted these different components in the JSON object's global log inside translateQuery() for different queries I had searched. I had tested queries with one key word and multiple keywords to help me understand specifically how a query with multiple keywords, including the keyword 'and', would be split up amongst 'operatorSearchTerms', 'attributes', and 'attributeValue'.

Conclusion and Future Work

With regards to next steps, the system exploration that I have done with minimizing the number of API calls in searching queries and multiple keywords can be extended to actual changes in terms of functionality. As of now, I defintely have a stronger base understanding of how the system is working with regards to these two tasks. Overall, through this project I gained a lot of experience with contributing to and understanding how a search engine API would

function in a production environment. Initially, I came in with zero Docker experience and faced a lot of difficulties with debugging Docker errors with system setup during the first month I started working on the project. By doing so, however, I learned a lot about the importance of using Docker and containerization in systems as large as UVAMonitor and also how to navigate those different errors outputted by Docker. Similarly, by working on the event handling and search bar functionality tasks, I learned more about how the backend and frontend of a system interact through API calls *across* multiple containers; something I had never worked with previously. Lastly, by exploring minimizing API calls and keywords in queries, I gained great insight into how a query is parsed and split up before being used to extract information from a database. All in all, I am confident that the technical skills that I gained working on this system will guide me as I start my career as a data engineer this summer.