Search Optimization: Data Refactoring to Enhance User Search Experience

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

Rehan Javaid

Spring, 2023

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

Brianna Morrison, Department of Computer Science

Search Optimization: Data Refactoring to Enhance User Search Experience

CS4991 Capstone Report, 2022

Rehan Javaid
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
rj3dxu@virginia.edu

Abstract
Event-organization technology provider, Cvent, noticed missing booking-restriction data from their Venue Search product, a venue search engine that displays booking rates for rooms in various venues. The missing data caused Venue Search to return unbookable rates to users. The solution to this problem involved performing a back-end data refactor in order to add missing fields to the current data, and designing logic to handle search functionality after the data modification. To implement our solution, I used Java and the Elasticsearch search-engine framework. As a result of our work, venues have enhanced flexibility in specifying their criteria-to-book; and users are shown more accurate booking rates while using Venue Search. Next steps involve front-end work to handle the display of the new data fields within the user interface (UI).

1. Introduction

Consider the following scenario: After the best-performing quarter since launch, I decide to treat my employees to a getaway to plan for next quarter. I take advantage of a third-party service to help plan my event in order to relieve some of the stress. The process for making my selections works perfectly: I am able to search for different hotels in the destination of choice, view meeting-room spaces available in these hotels, and instantly make bookings through the service. However, just as I am checking out, I receive a notice that I am unable to book. All the time I invested browsing through my options is wasted, and I am forced to rethink my options. This kind of customer experience within an application can be frustrating and is likely to result in their decision not to become a returning-user.

In the summer of 2022, Cvent experienced a very similar issue with one of its products currently in development. Similar to the service described, Cvent's *Instant Book* product allows customers to search for hotels and instantly book meeting and guest rooms through their interface. As part of *Instant Book*, *Venue-Search* is the search engine that powers the ability to search for hotels in the process. When a search is performed, hotels and booking rates that match the selected search criteria are shown. After testing this feature, however, Cvent noticed an unusual occurrence. Sometimes searches would disguise unbookable rates as bookable for users—an issue which could confuse and frustrate users trying to complete their booking.

A deep dive into the issue revealed that invalid search results were an effect of missing data fields. In the third-party venue database used by Cvent, a set of venue-specific restrictions on the ability to book were available but not being utilized. Thus,

searches did not take these restrictions into account and would display rates that did not pass the restrictions. As an intern, I was tasked with spearheading the effort to fix this issue.

2. Related Works

Missing data in an online platform is an issue that affects many fields of science and technology, including medical research, marketing, and search algorithms. Golebiewski and Boyd (2018) discuss data voids, which they define as data that is "limited, non-existent, or deeply problematic." Data-voids are dangerous due to the fact that they can lead to severe misinterpretation and bias within the results returned from a search. Querying, the action that filters the search results, is a major part of the problem since the way a query is structured can completely change the results shown to users. It is the responsibility of the search engine manager/creator to ensure that data voids are minimized at all costs within the data sets being utilized.

Cvent's *Venue Search* product was built upon the technology offered by Elasticsearch. Elasticsearch is an open-source search engine that provides an efficient solution to searching by allowing you to "store, search, analyze big volume of data quickly and in near real time." (Dritto, 2019). It is often integrated into applications that have "complex search features and requirements" in order to use Elasticsearch's solution to efficient, rapid searching while still offering customizability to meet the needs of your specific application. Some of the most well-known companies using Elasticsearch in their applications include Netflix, Uber, and LinkedIn (Scott, 2022).

3. Process Design

Implementing the solution required the use of a specific technology stack and a number of steps that built off one another. In this section, I describe those specific technologies that were used and provide an outline of the steps that I took to build my solution.

3.1 Technology Stack

Designing the correct solution to the issue required working with a number of different technologies and targeting a number of touchpoints within the code base for *Venue Search*. Notable technologies used in the design of my solution include Java, Junit, Painless, Postman, Kibana, Elasticsearch, Karate, and Docker. Notably, nearly all of these technologies are used for back-end development.

3.2 Process Timeline

Cvent follows the Agile software development cycle. This style of group development involves the division of larger projects into chunks of smaller work called tickets which are assigned to individual developers. Assigned tickets are to be completed at the end of two weeks, after which new tickets are assigned. My project was similarly divided up into bi-weekly tickets for me to complete over the duration of my ten-week internship. There were five tickets for me to complete in total.

3.3 Explanation of New Fields

Cvent obtains all of its venue data from a third-party service called DerbySoft. The availability restrictions for each venue were accessible in DerbySoft, but were not being integrated in *Venue Search.* The availability restrictions placed a variety of constraints on a customer's ability to book a particular rate. Some specific restrictions included firstValidReserveTime and lastValidReserveTime restrictions which provided a time window of days before a booking for which you are able to make a booking and also minBookingLimit and maxBookingLimit which placed restrictions

on the number of nights for which a rate could be booked. In order to be effectively handled within searches, these restrictions would need to be added to the data model for rates. Next, logic would need to be added in order to use these new fields to filter results returned from performing a search within *Venue Search*.

3.4 Key Steps

The first step of my solution was to change the data model for rates to add the availability restrictions. To accomplish this, I modified a JSON file containing the structure for venue rates; placed the new fields within the JSON rate object; and —tested that the new restrictions had been added using a service called Kibana. Kibana allows you to simulate performing searches with Elasticsearch. I used the service to ensure that when I searched for various rates I was able to see the new ingested restrictions within the returned data.

Next, I needed to create a function to bulk update all of the current rate data shapes to the new shape with the added restrictions. Doing this involved the use of a scripting language: Painless. A scripting language allows you to write programs to be executed by another program. In this case, the function that I wrote would be executed by Elasticsearch. I assigned every previous field being stored within the rate data structure to the field of a newly created rate containing the availability restrictions. Testing this function could similarly be performed with Kibana.

The longest step in the process involved working in Java to add the filtration logic necessary for the availability restrictions to be effectively handled by *Venue Search*. With the rate data structure updated to contain the new fields, I needed to design the logic that would specify the rate return criteria using these restrictions. As an example, using the window set by the lastValidReserveTime/firstValidReserveTime availability restrictions, if someone is performing a search from *Venue Search* at a time that does not fall within this window, the rate should not show up in the search results since it could not be booked at the time.

Logic like this was designed for each of the availability restrictions. I used Junit extensively as a means of testing the correctness of the logic implemented. Junit is a testing framework for Java used for automated unit testing: a means of testing individual units of a program. It was important that these tests handled edge cases of each of the restrictions appropriately. I also used Docker extensively in order to be able to run simulated versions of *Venue Search* within the testing environment for testing. After all of the Java logic had been written and tested I was confident that I had produced a working solution to the issue.

3.5 Challenges

One of the biggest challenges to overcome while coding my solution was learning about the new technologies that I was working with. I sometimes found my success being hindered by a lack of knowledge about a certain technology and a lack of online resources specifically related to the issue I was having. However, as I grew more comfortable with the technologies that I was using, over time this became less of a challenge.

Another challenge I faced while implementing my solution was vagueness causing misinterpretations. The availability restriction fields being defined by DerbySoft rather than by Cvent sometimes led to misinterpretations of exactly what a specific availability restriction meant. One availability restriction in particular—the

min/maxAdvanceDay fields—was a prime example. I designed the logic to handle this availability restriction within a search; however due to the vagueness of what the fields actually meant I ended up with logic that incorrectly handled the restriction. Meeting with DerbySoft representatives for clarification steered me in the right direction, though the process of doing so delayed my development schedule.

4. Results

Through my work, the issue in which *Venue Search* would return unbookable rates for customers is no longer present. This has significant ramifications for the overall customer experience using *Venue Search*. Only displaying rates that are bookable eliminates the frustration of searching for extensive periods of time for a desired rate only to find that it is not available.

The impact of adding availability restrictions to rates in *Venue Search* has beneficial effects for venues, as well. Since availability restrictions are venue-specific, adding these criteria into the conditions to book gives venues more specificity over the conditions to book their spaces.

5. Conclusion

In conclusion, the solution to *Venue Search*'s issue of unbookable rates resulted in an improved customer booking experience and offers venues new specificity in determining the requirements that must be met in order for a customer to make a booking. By creating a more robust, error free, product, Cvent increases the likelihood of getting customers to continuously use their service. Furthermore, offering venues more specific requirements that must be met in order to make a booking means that venues are able to better target the exact customers that they are able to oblige. This results in more clarity for customers looking to make a booking.

6. Future Work

While the solution has been successfully built in the backend, the frontend piece of the solution is still needed. The frontend work of this solution involves additions to the user interface such as a message that displays which rates are bookable and which rates are not bookable when *Venue Search* returns. Additionally, user testing of the new additions needs to take place in order to ensure that the additions work well from their perspective. I estimate roughly 2-weeks' worth of work is needed to complete these additions.

References

Dritto, G. P. (2019, March 27). An overview on Elasticsearch and its usage. Retrieved October 30, 2022, from https://towardsdatascience.com/an-overview-on-elasticsearch-and-its-usage-e26df1d1d24a

Golebiewski, M., & Boyd, D. (2018, May). *Data Voids: Where Missing Data Can be Easily Exploited* [Scholarly project]. In *Data Society.* Retrieved September 20, 2022, from https://datasociety.net/wp-content/uploads/2018/05/Data_Society_Data_Void_Final_3.pdf.

Scott, S. (2022, October 25). These 15 tech companies chose the elk stack over proprietary logging software. Retrieved October 30, 2022, from https://logz.io/blog/15-tech-companies-chose-elk-stack/