OULU UNIVERSITY OF
APPLIED SCIENCES

Roope Kettunen

IMPLEMENTING A SEARCH ENGINE SOLUTION

IMPLEMENTING A SEARCH ENGINE SOLUTION

Roope Kettunen
Bachelor's Thesis
Spring 2022
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Option of Software Development

Author: Roope Kettunen
Title of the bachelor's thesis: Implementing a Search Engine solution
Supervisor: Kari Jyrkkä
Term and year of completion: 2022 Number of pages: 31

The aim of this thesis was to create an efficient search solution for a pre-existing internal website so that users of the site could find test reports in a more flexible and easy-to-use manner. The thesis work was commissioned by Nokia Solutions and Networks Oy. To be more precise, the project was done for the RF BB (radio frequency baseband) team in a Test Automation trainee position.

This thesis documents the complete design and implementation process of a search engine solution. Starting off, the prerequisite information to understand the gist of the underlying project is presented, followed by a quick overview of the problem. Designing and implementing the solution is then discussed at length. The thesis is wrapped up with presenting the final product along with brief descriptions about future upkeep and possible updates.

The end-result of the thesis work was a fully working search system up and running on the company's internal website. Internal documentation was prepared for other developers to maintain and update the system if needed.

Keywords: Software Development, Web Development, Search Engine

PREFACE

This thesis was commissioned by Nokia Solutions and Networks Oy. The work was done when working full time for the company as a test automation software trainee under the course of roughly three months. The instructing teacher for this thesis was Kari Jyrkkä from Oulu University of Applied Sciences and the client's representative was Miika Tuuli from Nokia.

Oulu, 17.2.2022
Roope Kettunen

CONTENTS

VOCABULARY

Docker Image

Docker images are files used to execute code in Docker containers – basically acting as templates for building docker containers. These images can be created based on, for example, a developer's own created software application, or they can be downloaded from online sources. The latter option is the case for the software used in this thesis. (1.)

Docker Container

A docker container is a piece of software that packages up code and its required dependencies so that an application can be set-up quickly, run independently in its own environment and be moved easily to different environments. (2.)

DUT

DUT stands for a "device under test". In this environment, these are radio circuit boards used for testing in a laboratory environment, where test engineers perform various measurements on them.

ELK Stack

The ELK stack is a collection of open-source software products by Elastic. They are intended to be used together for efficiently creating and maintaining search engines that use the Elasticsearch engine. These software products may be

used independent of each other as well, but in this thesis they are used together for a common purpose of building a search engine solution. (3.)

Instrument

Instruments are lab equipment used in performing test scenarios. These include equipment such as oscilloscopes and temperature chambers. Many of these instruments may be controlled using PC software.

1 INTRODUCTION

This thesis was commissioned by Nokia Solutions and Networks Oy. The thesis work was performed under the RF BB (radio frequency baseband) team. Nokia is one of the biggest tech companies in Finland as well as around the entire world, employing close to 100,000 people globally, with revenue numbers in the tens of billions. Currently, its primary area of specialization is telecoms, notably 5G networks. (4.)

The objective of this thesis was to create a working solution for efficiently searching and displaying data based on automatically generated test reports. The thesis starts with introducing the reader to the background of the pre-existing Test Automation project and thus the problem at hand. Afterwards the text focuses on planning and implementation of a solution in two parts, firstly detailing the back-end features and after that moving onto the front-end features.

2 INTRODUCTION TO TEST AUTOMATION

2.1 Test Automation Project

Testing is an integral part of any hardware-related project. Before a product can be delivered to a customer, it is essential that its features have been confirmed to work as expected. Naturally, any improvements in speeding the testing process up, reducing its costs or improving the quality of the results are invaluable in delivering the product in a timely manner as well as keeping the company ahead of its competitors.

This is where test automation comes in. Nokia's RF BB team's Test Automation project aims to provide the tools and support to enable automating as much of the RF BB team's workload as feasibly possible. In practice, this means providing software solutions which can reduce manual testing phases, improve the quality of the results and speed-up testing in general. This includes not only programmatically controlling various devices in the laboratory, such as oscilloscopes, but also providing features that can automatically generate graphic results from performed measurements for users to review and confirm the validity of the results. (5.)

2.2 Test Report Website

For the purposes of this thesis, it is essential for the reader to understand the basis of the underlying project in which the solution is being implemented to. The various changes presented in this thesis are planned and implemented into an already existing website created by Nokia's RF BB Test Automation team.

In the current state of the project, tests performed by test engineers are automatically compiled into Test Reports which are displayed in an in-house Test Report Website solution. This allows the users to easily view and share the results of their testing.

The main technologies used in the pre-existing solution include React JS for the front end, Flask for the back end and MySQL for the database. The website is privately hosted and is not publicly available for viewing.

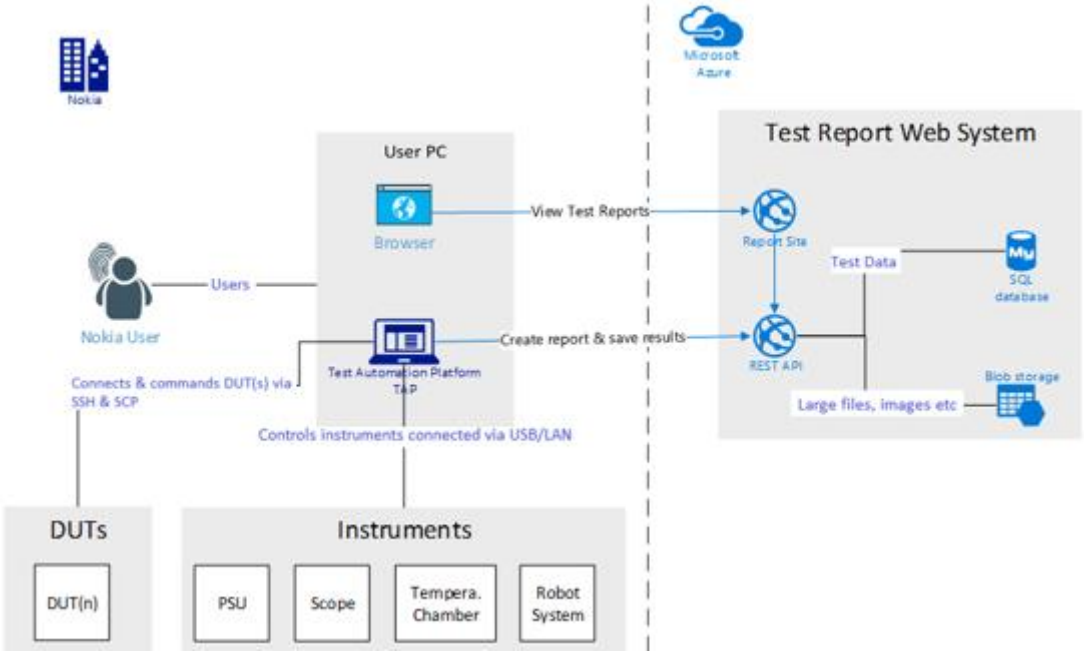

FIGURE 1. RF BB Test Automation Solution

The basics of the pre-existing Test Automation solution is shown in figure 1 above. Test engineers operate a PC in a lab environment which in turn controls various instruments to perform the required measurements on DUTs. When using the specialized TAP program, the test results may be automatically saved and test reports are created for later viewing and review.

The changes presented in this thesis are done strictly on the Test Report Web System's side of things, with the possible exception of slight changes in the TAP system if any new types of data needs to be saved from future Test Runs to facilitate the Search Engine usage.

2.3 The Problem

In the current state of the project, it is easy for the users to find the results of their most recently performed tests on the website. However, problems arise when trying to find older, more specific test reports. The only way to search for different test reports is to search for them by user-defined name from a specific time interval. In addition, since some test reports can be exceedingly long, finding a specific measurement result can be difficult still.

This presents a clear opportunity for improvement. The Test Report Website needs an efficient solution for finding specific data. To be more precise, there needs to be a way to search for test reports and results using broader terms or a combination of factors, such as the tested device, measurement type or the test result (pass or fail). The optimal way to offer this solution would be to implement a free-form search field backed up by optional filters.

3 PLANNING A SOLUTION

3.1 Dedicated Search Engine vs Database Queries

The need for searching data presents an important question; why not simply use the query functions in the storage database? True enough, MySQL offers great functionality for querying data. However, these queries are primarily intended to respond to queries where the user is looking for a specific value in a specific table. Unlike search engines, database queries struggle heavily when trying to find specific words in a string field.

In addition to generally much faster queries, a dedicated search engine provides various useful features not feasibly producible with the MySQL query syntax. One example of this would be a fuzzy search, where typos in the search query are accepted and easily handled. Another example would be predictive text, where the search field could intelligently provide the user with autofill options. Some search engines also come bundled with handy visualization features. (6, 7.)

3.2 Choosing the Right Search Engine

It is important to keep in mind that the use cases for a search engine in this project are relatively narrow. Therefore, the best options for choosing a search engine were the ones that were both open-source software to minimize development and upkeep costs as well as popular enough to have various tutorials available to ease getting started with the development process. The feature-sets of the three most popular open-source search engines are compared in table 1 below.

TABLE 2. Comparison of Feature Sets of the Most Popular Open-source Search Engines (8).

	Elasticsearch	**Solr**	**Sphinx**
Search Features	Good	Best	Good
Performance	High	High	High
Scalability	High	High	High
Data Visualization	Supported	Supported	Not Supported
Machine Learning	Yes	Yes	No

In the end, Elasticsearch was chosen for the project. While all reviewed candidates offered a good selection of features, Elasticsearch's easy-to-setup data visualization system as well as its widespread popularity and associated supporting software helped it edge out the competition.

3.3 Self-hosted vs Cloud Service

Generally, when looking at open-source options, their primary business model is to offer the open-source software in a ready-to-use Cloud Service. Such is also the case with Elasticsearch. Of course, self-hosting the service is not free either, as the developers need to configure their own environment to run the service.

Using a paid Cloud Service comes with two major benefits. First, the burden of upkeeping the service is shifted to the service-provider instead of the developer. Usually, this is translated to increased reliability and scalability. Secondly, generally speaking, less work needs to be done by the developers to get the services running as a dedicated server environment does not need to be configured to run the services.

All things considered, it was decided that the service would be initially self-hosted as many of the project team members already have experience of hosting web services. Furthermore, if scalability or reliability ever became an issue, it would be relatively easy to transfer the services to a dedicated Cloud provider as their contents would already be mapped out.

4 IMPLEMENTING PROJECT BACK END

4.1 Tech Stack for the Project Back End

Before heading onto the actual description of the solution implementation process, the main technologies used in creating the back end solution are briefly presented.

4.1.1 Elasticsearch

Elasticsearch is the most integral component of the open-source ELK stack. It is a search and analytics engine for various types of data, including textual, numerical, structured or unstructured. The primary use of the engine in this project is to use its provided APIs to create functionality that can provide a website requested data using powerful search-optimized tools. Furthermore, the included various logging and analytics tools could be used to expand the project in the future, if deemed useful. (9.)

4.1.2 Kibana

Kibana is another part of the open-source ELK stack. It is a frontend application that provides search and data visualization capabilities for data indexed in Elasticsearch. While not an absolute necessity for implementing the solution, it can still be an extremely useful tool for monitoring, managing and securing data. (10.)

4.1.3 Logstash

Logstash is also a part of the open-source ELK stack. It is a data processing pipeline that can ingest data from various sources, transform it, and send it to a

“stash”. In this project it is used to pipeline test-related data from a MySQL database to an Elasticsearch data index. (11.)

4.1.4 Elastic App Search

The final open-source component provided by Elastic used is Elastic’s App Search software. This software provides the most modern and best supported way for indexing data for searching purposes as well as providing the tools to handle API requests.

4.1.5 Docker

Docker is a set of software that can deliver software in packages called containers. Containers are isolated from each other and utilize their own software and libraries, although communications can be set to pass through them. In this project the ELK stack is divided into these containers with each container taking care of a single service. Using docker-compose multiple apps (containers) can be brought to communicate and depend on each other. (2.)

4.1.6 Debian/Linux/NESC

The back-end is hosted on a virtual machine running Debian. This virtual machine is run on Nokia’s own privately hosted servers and thus grants in-house software developers free access to practically any feature that a developer may need.

4.1.7 MySQL

MySQL is a widely-used relational database management system. Unlike the ELK stack, the project already has a running MySQL solution implemented. This database contains testing related data gathered by the pre-existing Test Automation solution. (12.)

4.2 Implementing the Solution

The primary object of the back end solution is to setup the Elasticsearch engine in such a manner that its functions can be accessed via API calls from the Test Report website. This requires not only setting up Elasticsearch and related, required services, but also indexing the pre-existing MySQL data and also making sure that future MySQL data additions are automatically indexed as well.

4.2.1 Indexing Data

Indexing means optimizing the contents of a database so that they can be rapidly searched for. The downside and reason that all database contents of every database are not indexed is that the indexed contents take up more memory. Thus, indexing should only be done for data that needs to be searched for often.

Pre-existing test data contained in the Test Automation MySQL database would be set to be indexed fully only once. Afterwards data would be indexed automatically every few minutes – this would include only new data to keep the queries short. Of course, this would mean that old data, which might get deleted from the database, would remain in the index – which would result in search results containing dead links – although this case would not really have a chance to happen in practice, as its unlikely data would be deleted. This could

also simply be fixed by checking if the content has been deleted from a database row.

Furthermore, not every single piece of data needs to be indexed, only data that could actually be used in a user's search. For example, the user would never search for a web address link to an image used in a test report page, even if such would be used in the document.

First, the used software is going to need a reliable host environment to be set-up in. Using the paid version of each of the ELK stack members would be the easiest way of taking care of this problem. However, Nokia has provided the Test Automation project with access to its privately hosted virtual machines which allows setting up the back end in a relatively risk-free and widely configurable environment.

Having set up a Virtual Machine running Linux Debian, the first step was to install the required Docker software. The process was well enough documented to make this a fairly simple task (13). This allowed the use of OS-level virtualization to create separate distinct containers for each of the tech stack members.

To download the services and get them running, docker-compose was used. This is a service that can "bundle" the setup of multiple containers into a single file and thus make them easy to setup as interconnected. After configuring the docker-compose file, running it would launch a chain of events that would download "images" of the services, configure data volumes for storage purposes and finally start running the services, ready to use. (14.)

Although the necessary services' images could be freely downloaded for use from docker's provided resources, deeper configuration was necessary. One example of this was setting security measures for the software. This was a step that was not clear-cut in the slightest and required an in-depth review of the Elastic Stack's documentation.

Another sizable part of the configuration process was setting up an automated indexing pipeline. In practice, this was performed in a manner in which queries need to be set to return all data that needs to be indexed. These queries would then be run every so often as per a pre-defined time interval. In this case, as searches do not need to be done immediately after finishing a test, it was deemed fine to only index new data every 5 minutes or so.

For creating this functionality, Elastic's Logstash service was used. It could be set to retrieve data from a MySQL service (through a JDBC-connector plug-in), process it and send it to Elastic's App Search's index.

Setting up the App Search service itself was a simpler task – using the Kibana visualization service provided in the Tech Stack, the App Search service could easily be deployed and monitored through a web interface. With App Search up and running, the next task would be setting up querying the data from an actual web service.

4.3 The Completed Back End Solution

With all the services installed and their configurations set, the back end side of the project was finished. This means that a pipeline had been fully configured to take care of automatically processing data from a database to a search engine index. The search engine's functionalities were ready to be utilized through Elastic's App Search. A simplified explanation of the final back end solution is presented in the figure 2 below.

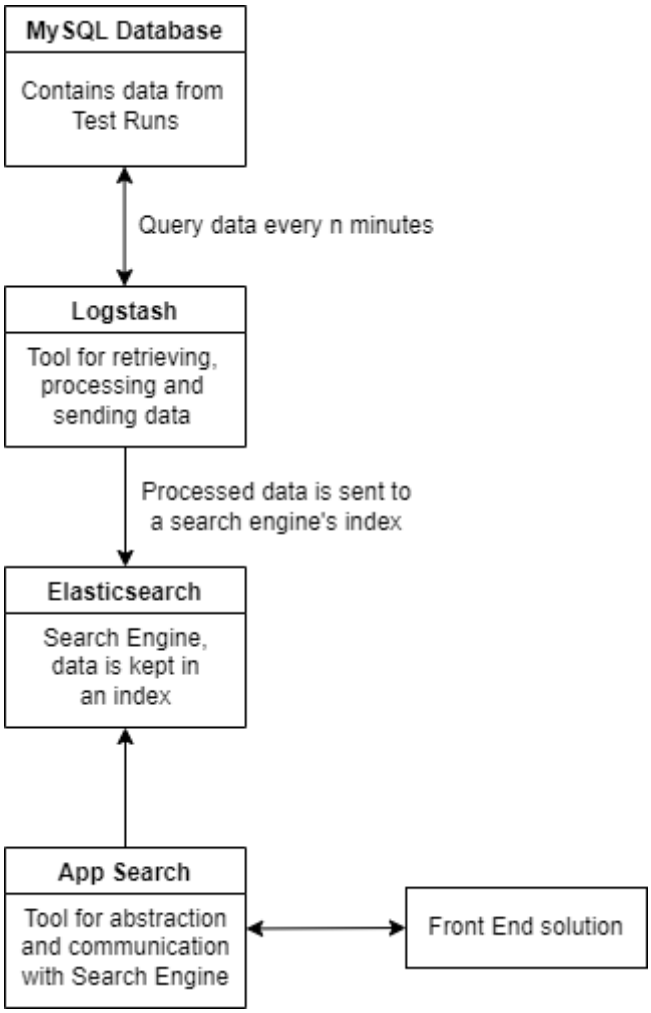

FIGURE 2. Back End Solution Simplified

5 IMPLEMENTING PROJECT FRONT END

5.1 Interaction Between Back End and Front End

As stated in the earlier chapter, the primary purpose of the back-end solution created for the purposes of this thesis is to provide the “backbone” of features implemented in the front-end. In practice, this means that appropriate mechanisms need to be in place for the back end to answer to queries performed from the front-end user interface.

In this case, this cross-communication consists of the user’s search string as well as optional search settings. The back end would then reply with the best matching results, or to be more specific, the ID of the best matching test report. This can then be used to create and open up a web-view of the appropriate results page.

5.2 Introduction to Web Stack

This sub-chapter describes the technologies used in the pre-existing web solution for displaying test reports.

5.2.1 React JS

React is one of the currently most used JavaScript libraries for building user interfaces for web applications. It uses an intuitive component-based system in which these components manage their own state and are composed together into a more complex User Interface. These component views are highly efficient in updating and rendering based on their state and changes. (5.)

5.2.2 Flask

Flask is the main framework used for the web site's backend. It is a library for the Python programming language designed for a simple creation process for web backend solutions. The main advantage of it is indeed its simplicity, provided by the comparatively easy-to-use, yet powerful tools it provides. In the Test Automation project, the primary use of it was to provide database interaction functionalities. This part of the front end solution would need to be reconfigured if the underlying MySQL database would need changes to facilitate certain Search Engine related functionalities.

5.2.3 Search UI

Search UI is an open-source React library that allows the user to relatively easily implement search experiences. It comes out of the box with support for the Elastic App Search used in this project making it the perfect candidate for implementing the front end user interface. Developers using it can also freely customize the User Interfaces provided by the library, providing a high degree of flexibility. (16.)

5.3 User Experience

As the Test Results Web page is unsurprisingly an interface made for the end-users (mainly test engineers), user experience becomes an extremely important subject to discuss. User experience, or UX for short, is a vital part of any graphical piece of software, such as a web site, video game or a mobile application. It dictates how it feels to use the system and how intuitive it is. As such, the most important graphical elements, search bar included, should be readily available and usable without an external guide whenever possible. Furthermore, the system should work with as little delay as possible and avoid hick-ups in possible animations used.

5.4 Creating the User Interface

The initial plan was to create a fully tailor-made React component for displaying the data. However, upon further research it was discovered that Elastic already provides a framework for creating a User Interface as well as communicating with the Elastic back-end. This, however, required the addition of the Elastic App Search / Enterprise Search to the back end, which shifted the amount of workload backwards a fair amount as the entire Docker Compose process had to be redone.

The used solution was the open-source Search UI React library. This provided access to flexible and customizable UI components as well as the possibility to communicate with the back end via a more abstracted interface.

Firstly, creating the UI components themselves ended up being a trivially simple task. A button was dedicated in the sidebar of the website to reach the actual search window. There the user would be greeted simply by a Search bar, which included auto-fill options, similar to Google's search, for example. Upon entering a search term, the user could then filter and re-order the results from a sidebar.

One of the problem areas worth mentioning in creating the UI was linking the user to the related Test Report based on the search results. The preferred way provided by the UI library was to include result links in the underlying database, but as this was not the case in this project, a new solution needed to be implemented. In the end, a link-generating clause of code was added to the project back-end to solve the problem.

The final flow of using the system is presented in a simplified form in figure 3 below. Simply put, users can search for terms which result in a list of search results appearing in the web app. The inner workings of the system are abstracted outside of the web application.

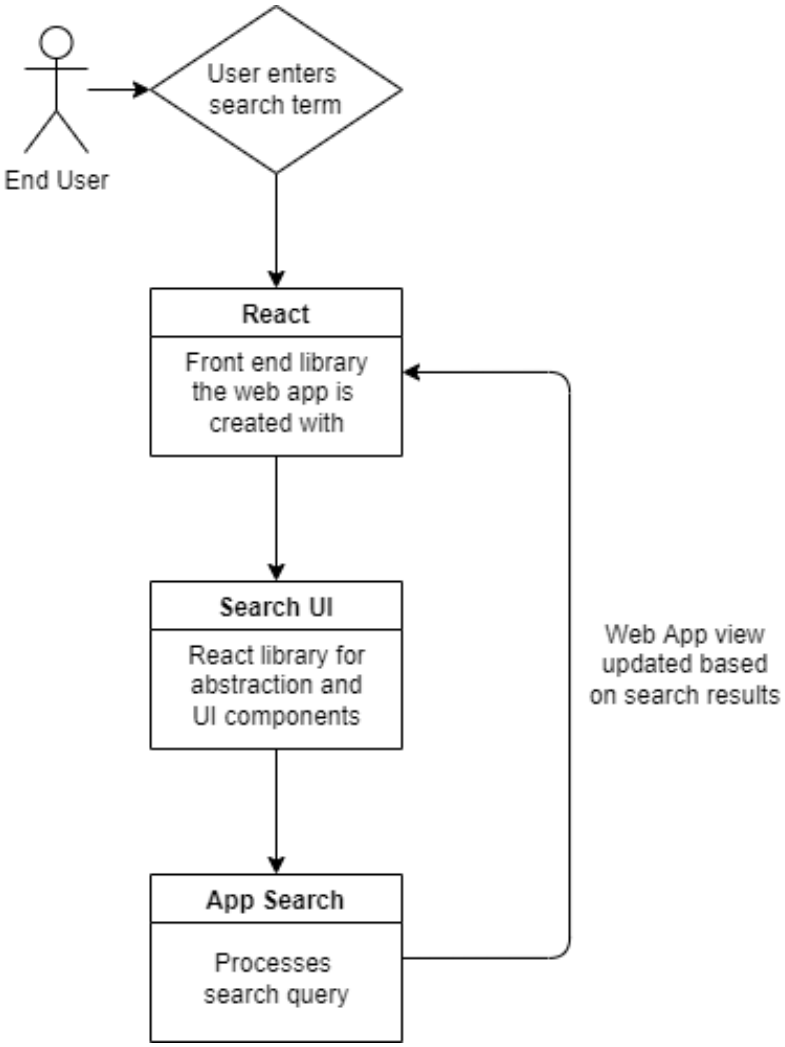

FIGURE 3. Front End Solution Communications Simplified

5.5 Search Features

Upon arriving on the search page, the user would be greeted by nothing but a simple Search bar. The search bar comes with auto-fill options for most common search cases, similar to Google's search, for example. Upon entering a search term, the user could then, for example, filter the results based on the test's performer or re-order them based on the test's date, all handily from the controls in the sidebar. A fuzzy search is an another interesting feature provided by the system. With a fuzzy search the user can mistype words and the search

engine can still find accurate results. Finally, the user can click on the search results to open the Test Report, as expected.

6 CONCLUSION

6.1 Putting It All Together

Upon demonstrating the fully working solution to members of the RF BB team, it was deemed acceptable and ready to put into use, after a few minor changes to usability based on received feedback. With the back end already up and running, the final step was to push the front end changes to the corresponding Git repository and to update the live website to include the changes. With these things in order, the solution was finally ready and in use. Below is a sample of the final product in use (figure 4).

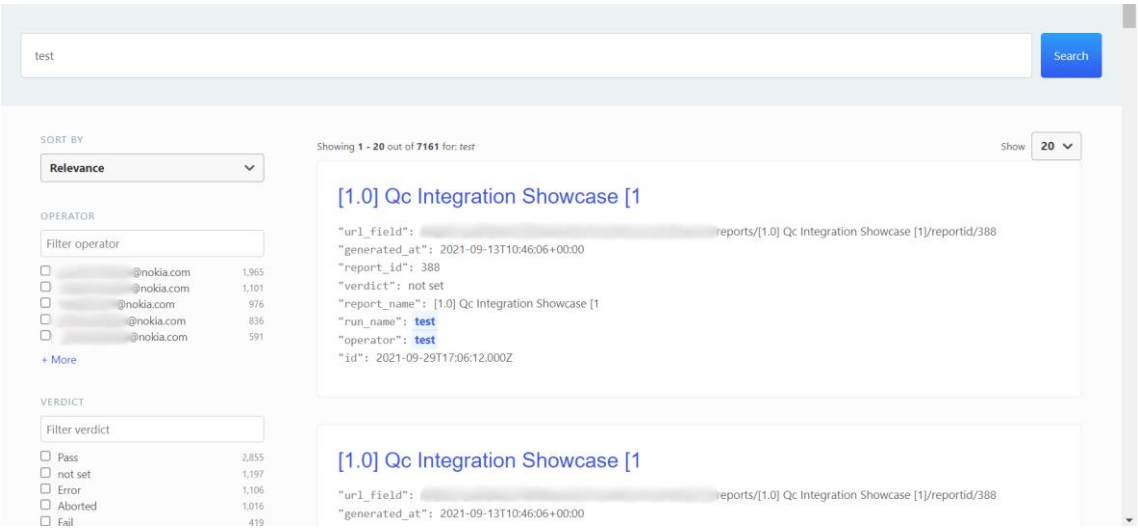

FIGURE 4. Search System in Use

6.2 Matching Initial Needs

As mentioned at the beginning of the thesis, there was a need for an easy-to-use system to search for Test Reports. There needed to be a way to streamline the search process to include things, such as sorting and searching by the test operator's name.

The end-result ended up being perhaps even better and more efficient than was initially planned, thanks to the various modern features provided by the Search Engine that were fully utilized in the final product. It is also worth noting that the final product was lightning-fast to use, offering a seamless and pleasant searching experience. That being said, sometimes search results would not quite match up with the users' search queries. These could be fixed in post by introducing search keyword weightings and additional filters.

6.3 Things Learned

Search engines can be an extremely powerful addition to any web application, be it a Test Report system or perhaps some kind of an e-commerce or social media platform. When properly implemented, the User Experience can be greatly boosted in quality. They come with a hefty drawback though; they are rarely easy to implement and require knowledge in various areas of software development. It is important to realize when one could benefit from a search engine and when a simple database query would suffice.

6.4 Future of the Search System

Considering that other developers may need to configure and upkeep the search system, internal documentation was prepared to explain the basics and flow of the system, as well as detailing possible problem cases and solutions.

A few ideas that were considered outside the scope of the thesis work were presented during development. One example of these would be auto-scrolling to a point in a Test Report indicated by the search results.

REFERENCES

1. Gillis, A. Docker image. SearchITOperations TechTarget. Date of retrieval 21.1.2022 https://searchitoperations.techtarget.com/definition/Docker-image

2. Docker. Use containers to Build, Share and Run your applications. Date of retrieval 21.1.2022 https://www.docker.com/resources/what-container

3. Elastic. What is the ELK Stack? Date of retrieval 21.1.2022 https://www.elastic.co/what-is/elk-stack

4. Alsop, T. 2022. Nokia – statistics & facts. Date of retrieval 3.2.2022 https://www.statista.com/topics/1183/nokia

5. Hautamäki, A. 2020. Designing Software Arcitecture for a Test Automation System. Date of retrieval 21.12.2021 https://www.theseus.fi/handle/10024/347132

6. Lucidworks 2019. Full Text Search Engines vs. DBMS. Date of retrieval 28.10.2021 https://lucidworks.com/post/full-text-search-engines-vs-dbms/

7. Thunderstone 2019. What is the Difference Between a Database and a Search Engine? Date of retrieval 28.10.2021 https://www.thunderstone.com/blog/archive/what-is-the-difference-between-a-database-and-a-search-engine/

8. Klimenko, A. SOLR VS. ELASTICSEARCH VS. SPHINX: BEST OPEN-SOURCE SEARCH PLATFORM COMPARISON. Date of retrieval 29.10.2021 https://greenice.net/elasticsearch-vs-solr-vs-sphinx-best-open-source-search-platform-comparison/

9. Elastic. What is Elasticsearch? Date of retrieval 11.11.2021 https://www.elastic.co/what-is/elasticsearch

10. Elastic. What is Kibana? Date of retrieval 11.11.2021 https://www.elastic.co/what-is/kibana

11. Elastic. Logstash – Centralize, transform & stash your data. Date of retrieval 11.11.2021 https://www.elastic.co/logstash

12. MySQL. What is MySQL? Date of retrieval 12.11.2021 https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

13. Docker docs. Install Docker Engine on Ubuntu. Date of retrieval 15.12.2021 https://docs.docker.com/engine/install/ubuntu/

14. Docker docs. Overview of Docker Compose. Date of retrieval 15.12.2021 https://docs.docker.com/compose/

15. React. React. Date of retrieval 17.12.2021 https://reactjs.org/

16. Chow, Kellen, Stoltzfus, Yakhin 2022. Search UI. Date of retrieval 14.1.2022 https://github.com/elastic/search-ui