Republic of Tunisia
الجمهورية التونسية
Ministry of Higher Education and Scientific Research
وزارة التعليم العالي و البحث العلمي

End-of-studies Internship Report

submitted in partial fulfillment of the requirements for the degree of

Software Engineering

Design and Implementation of a Name Indexing and Search Tool

Achieved by :
M. Sleimi Khalil

Supervised by :
M. Ben Romdhane Ahmed (Vneuron)
Mme. Bouzouita Ines (ENIT)

Defended on July ___, 2021, In front of the jury composed of :

M. _____ _____ : ENIT - President
M. _____ _____ : ENIT - Examiner
M. _____ _____ : ENIT - Rapporteur

Promotion : 2020/2021

Supervisor Signature

Prof. Ines Bouzouita

Dr. Ahmed Ben Romdhane

Dedication

“

To my parents, before all
The amount of love and appreciation that you two showed me cannot ever be repayed. I'll forever be indebted to you two. I can only be thankful and grateful for the efforts that you've done for my success as well as my well being day and night throughout the years. This work is the fruit of the sacrifices that you've done for my education and success, I hope you're proud.
« I love you very much »

To my dear brother and sister.
You were always by my side helping me, motivating me, and pushing me forward. I am proud to have you. You have contributed to my success through your unwavering support.
« A big thank you from all my heart. »

Thank you.

”

- Khalil

Acknowledgments

First of all, I would like to especially thank my company supervisor *Mr. BEN ROMDHANE Ahmed*, manager of the R&D department, for his immense help, the quality of his follow-up as well as for all the advice and information he lavished on me with an unequaled degree of patience and professionalism, and also for offering me the opportunity to join his team and for his support.

I would also like to thank my university supervisor *Mrs. BOUZOUITA Ines*, for the competent help she gave me, for her patience and her encouragement. Her critical eye was invaluable to me in structuring the work and in improving the quality of the different sections, for her availability and guidance.

I would also like to thank all of the employees, engineers and intern at Vneuron for the experiences and time spent together and for the invaluable information they provided me as well as for their encouragements.

May the jury members find here the expression of my sincere thanks for the honor they do me by taking the time to read and evaluate this work.

Finally, I would like to thank everyone who contributed directly or indirectly to the achievement of this work.

Résumé

Dans le cadre d'un stage de fin d'études pour l'obtention du diplôme national d'ingénieur en génie informatique de l'École Nationale d'Ingénieurs de Tunis, ce travail est réalisé au sein de la societé Vneuron pendant la période entre Février 2021 et Juin 2021.

Ce projet a pour objectif principal le développement d'un outil qui permet d'indexer une liste de noms de personnes et d'entités et de faire des recherches floues et/ou phonétiques sur cette liste. La méthodologie du travail adoptée est une méthodologie agile à s'avoir la méthode « Scrum ». On a utilisé le C4 Model pour la conception générale de la solution tout au long des différentes étapes du projet. Le développement de cette application est effectué à l'aide des Framework Spring et Angular ainsi que le moteur de recherche et d'indexation Elasticsearch et la base de données Postgresql, et le serveur d'authentication et d'autorisation Keycloak.

Mots clés : Framework Spring, Spring Boot, Spring Data, Spring Security, Angular, Elasticsearch, Postgresql, analysis-icu Plugin, analysis-phonetic Plugin, LTR Plugin, Indexation, Recherche, Correspondance des Noms, Sanctions Economiques, Docker, Prometheus, Grafana, Keycloak, Angular AuthGuard, Dom4J, OpenCSV, Jackson, Logstash, Kibana...

Abstract

As part of a final dissertation for the National Engineering Diploma in Software Engineering of the National School of Engineers of Tunis, this work is carried out within the host company Vneuron during the period From February 2021 to June 2021.

The main objective of this project is to develop a tool that indexes a list of names of people and entities and allows for fuzzy and/or phonetic searches on this list. The methodology of the work is an agile philosophy, in particular we followed the Scrum method. The C4 Model was used for the general design of the solution throughout the various stages of the project. The development of this application is done using the Spring and Angular Frameworks as well as the Elasticsearch search and indexing engine, a Postgresql DBMS and Keycloak's authentication and authorisation server.

Keywords : Framework Spring, Spring Boot, Spring Data, Spring Security, Angular, Elasticsearch, Postgresql, analysis-icu, analysis-phonetic plugin, LTR Plugin, Index, Search, Name Matching, Economic Sanctions, Docker, Prometheus, Grafana, Keycloak, Angular AuthGuard, Dom4J, OpenCSV, Jackson, Logstash, Kibana...

ملخص

وقع تقديم هذا العمل في إطار مشروع تخرج لنيل الشهادة الوطنية لمهندس في الإعلامية من المدرسة الوطنية للمهندسين بتونس. وقع إتمام هذا العمل داخل صفوف الشركة Vneuron خلال المدة الممتدة من 8 فيفري إلى 7 جوان.

هدف هذا المشروع هو تطوير أداة لفهرست لوائح أسماء أشخاص طبيعيين وأشخاص معنويين (شركات) واتاحت البحث الغامض والصوتي على هاته اللوائح التي وقع فهرستها. وقع إعتماد الفلسفة الرشيقة Philosophy Agile كمنهجية عمل، وبتحديد وقع إعتماد منهجيات سكروم .SCRUM قمنا أيضا بإعتماد سي فور C4 كمنهجية لتخطيط هندسة البرمجيات خلال مراحل العمل المختلفة.

استخدمنا عددا من أطر التكنولوجيا و البرمجيات أهمها Spring و Angular و Elasticsearch و .Postgresql

Contents

Dedication . . . . . **I**

Acknowledgments . . . . . **II**

Résumé . . . . . **III**

Abstract . . . . . **IV**

General Introduction **1**

1 Project Overview **3**
1.1 Presentation of the host organization 4
1.2 Presentation of the project 4
1.3 Choice of Methodology 7

2 Analysis and Requirements Specification **12**
2.1 Existing Solutions 13
2.1.1 The System currently in use: Reis KYC 13
2.1.2 Alternative Solutions 14
2.2 Requirements Analysis 14
2.2.1 Functional Requirements 15
2.2.2 Non-functional Requirements 16
2.2.3 Product Back-log 16

3 Design **19**
3.1 Problem Analysis and Solution Design 20
3.1.1 Dealing with large XML files 20
3.1.2 Complex search setting 20
3.1.3 User Friendly Environment 20
3.2 Architectural Design 21
3.2.1 Global Application Architecture 21
3.2.2 MVC Design Pattern 21
3.3 Detailed Analysis Diagram 22
3.3.1 Context 22
3.3.2 Containers 22
3.3.3 Components 23
3.3.4 Code 23
3.4 Supplementary diagrams 26
3.4.1 Dynamic Diagram 26

3.4.2 Deployment diagram 26

4 Realization **32**
4.1 Work Environment 33
4.1.1 Hardware Environment 33
4.1.2 Software Environment 33
4.2 Implementation 36
4.2.1 Data Extraction and Preparation 36
4.2.2 Elasticsearch Mapping, Analyzers and the Search query 40
4.2.3 The Front-end 45
4.2.4 Customers due diligence 51
4.2.5 Chronogram of Performed Activities 52

Conclusion **54**

Webography **57**

Appendices **62**

A Technologies Used: Elasticsearch **63**

List of Figures

1.1 Name Matching Phenomenas[**13**] 5
1.2 Scrum Methodology 10

2.1 Search Interface of Reis KYC[**15**] 13
2.2 Search Results of Reis KYC 14
2.3 Search Results of Rosette Name Indexer 15

3.1 Model View Controller Design Pattern. 22
3.2 A High level view of how our system 23
3.3 The different containers and parts that are used by our solution. 24
3.4 A Decomposition of the API Application Solution 25
3.5 Repository Pattern and the Service Layer Design Pattern used by our backend 26
3.6 Security - Front-end Authentication/Authorization Login 27
3.7 Search Request Sequence Diagram between the front-end and the back-end 28
3.8 Customer Due Diligence 29
3.9 Development Deployment 30

4.1 European Security Council Sanctions List XML 37
4.2 United Nations Security Council Sanctions List XML 37
4.3 United Nations Security Council Sanctions List XML 2 37
4.4 United Stated OFAC Sanctions List XML 38
4.5 United Stated OFAC Sanctions List XML 38
4.6 NDJSON Output to send to Elasticsearch 38
4.7 Watchlist Agreed Upon Format containing all relevant data to us 39
4.8 Watchlist Mapping within Elasticsearch 40
4.9 Watchlist Mapping multi-fields within Elasticsearch 41
4.10 Elasticsearch Fuzzy Matching Analyzer 42
4.11 Elasticsearch Transliterate Analyzers 42
4.12 Elasticsearch Phonetic Analyzers 43
4.13 Elasticsearch Ngram Analyzers 43
4.14 Elasticsearch Search Query using all the defined Analyzers 44
4.15 Keycloak Admin Console Login 45
4.16 Keycloak Admin Console Add User 46
4.17 Keycloak Admin Console Assign role to user 46
4.18 Keycloak Admin Console Assign set password to newly created user 47
4.19 Keycloak Admin Console manage existing users 47
4.20 Application Login 48
4.21 Search Engine Interface 48
4.22 Search Engine Same Name in Multiple Languages Russian/English 49

4.23 Search Engine Missing Name Component 49
4.24 Search Engine Missing Spaces or Hyphens 50
4.25 Search Engine Truncated Name Components 50
4.26 Search Engine Phonetically Similar Name Components 51
4.27 Search Engine Customer Due Diligence Search Results with High Probability 52
4.28 Internship Chronogram 52

A.1 Inverse Index 65
A.2 How new Documents are saved into Elasticsearch 66
A.3 How the Analysis process works 66
A.4 NFC Canonical Decomposition Mode Input Exampple 67
A.5 NFC Canonical Decomposition Mode Query for testing it 67
A.6 NFC Canonical Decomposition Mode Output String 67
A.7 A Comparison of the two different inputs before and after 68
A.8 An example input to the ICU_Tokenizer 68
A.9 An example output to the ICU_Tokenizer 69
A.10 An example input to the Ngram_Tokenizer 70
A.11 An example output to the Ngram_Tokenizer 71
A.12 An example input to the Shingles token filter 72
A.13 An example output to the Shingles token filter 73
A.14 An example input name in arabic script to the ICU_Transform 74
A.15 An example output name from arabic arabic script to latin script by the ICU_Transform 74
A.16 An example input name in russian cyrillic script to the ICU_Transform . . 75
A.17 An example output name from russian cyrillic script to latin script by the ICU_Transform 75
A.18 An example input name in Chinese Hanzi Script to the ICU_Transform . 75
A.19 An example output name from Chinese Hanzi Script to latin script by the ICU_Transform 75
A.20 An example input name in korean Hangul Script to the ICU_Transform . 75
A.21 An example output name from Korean Hangul Script to latin script by the ICU_Transform 75
A.22 An example input name using the double metaphone encoder 76
A.23 An example output name after being encoded using the double metaphone phonetic encoder 76
A.24 An example input name using the beider morse encoder 76
A.25 An example output name after being encoded using the beider morse phonetic encoder 77

List of Acronyms

AML	*Anti Money Laundering*
API	*Application Programming Interface*
BKD	*k-dimensional B-tree*
BM25	*Okapi Best Matching25*
C4 Model	*Context, Containers, Components, and Code*
HTTP	*Hypertext Transfer Protocol*
ICU	*International Components for Unicode*
JSON	*JavaScript Object Notation*
KYC	*Know Your Customer*
LTR	*Learning to Rank*
MRA	*Match Rating Approach*
NDJSON	*Newline Delimited JSON*
NFC	*Normalization Form Canonical Composition*
NFD	*Normalization Form Canonical Decomposition*
NFKD	*Normalization Form Compatibly Decomposition*
NFKC	*Normalization Form Compatibly Composition*
NYSIIS	*New York State Identification and Intelligence System Phonetic Code*
OCR	*Optical Character Recognition*

OFAC *Office of Foreign Assets Control*

ORM *Operational Risk Management*

PEP *Politically Exposed Person*

REIS KYC *Reis™ Risk & Compliance Suite - Know You Customer*

REST *Representational State Transfer*

SDN *Specially Designated Nationals*

StAX *Streaming API for XML*

TFIDF *Term Frequency–Inverse Document Frequency*

UML *Unified Modeling Language*

XML *Extensible Markup Language*

General Introduction

If you opened a bank account recently, took a plane, signed a mobile contract or even booked a concert ticket, your name will have been checked against national and international watch-lists and sanctions lists before the account usage or service was even authorized.

Many laws and acts such as the US's FCPA, USA PATRIOT, EARs; the UK's POCA, Terrorism Act of 2006 and the EU's Third Money Laundering Directive[**58**] as well as many more analogous national and international acts and legislation seeking to detect and deter terrorism funding, money laundering, illicit traffic, or simply as part of government's foreign policies on regimes and countries such as the recent Russia, Iran and China economic sanctions replacing military action wherever possible, in short, these laws require that an increasingly broader community of organizations, mainly financial institutions, shops, customs, airports, mobile network operators, 3PL providers, trade exporters and many others have effective systems and controls in place to ensure that entities found in these watch-lists are accurately identified and reported to the authorities, and to deny them service, or conduct enhanced customer due diligence.

Increasingly vigilant regulatory enforcement is also making it difficult to not abide by these rules or to have a bad solution in place, for example, failing to report to the authorities or to deny these entities service, or worse yet, knowingly breaking the law can cause a company to be penalized heavily, with BNP Paribas as an example having received a record fine totaling $8.9 billion for violating U.S. economic sanctions alone, or Goldman Sachs's $6.9 billion for bribery charges, or UBS's $5.1 billion fine over terrorism financement charges. All this means means that, financial regulations is something that no corporation, no matter how big, can afford to ignore. Thus, in order to keep an organization clean in both the regulatory and legislative sense, knowing which of its' customers, business partners, contractors, employees, or counter-parties pose a heightened risk enables it to take the necessary steps to protect the business and its brand and reputation in the market.

And with regulations becoming increasingly strict, name screening has to be conducted in a growing number of sectors and against a growing number of lists, there are now hundreds of lists worldwide, mounting up to millions of individuals and entities, extending from 'politically exposed persons', to 'state owned companies' to 'dual use materials'. Financial institutions are being held to increasingly precise levels of accountability when it comes to the systems they use to detect and prevent financial crime. That means, more accurate name matching in a financial compliance system is a must have and not merely a plus as its effects are seen in the form of fewer false positives and false negatives

which reduce regulatory compliance costs and exposure to risk, and brings immediate ROI, whether the goal is AML, KYC, on-boarding, sanctions, VIP/internal affairs, insider trading or fraud.

Not to mention the many possible alternative use cases of name matching solutions, ranging from the support of screening and credentialing services to disaster relief, thus benefiting distribution and helping with fraud prevention, a solution to the name matching problems can have many use cases in the real world that can be helpful to any business...

The aim of this project is to develop a tool that indexes a list of names of people and entities and allows for fuzzy and/or phonetic searches on this list, and that also takes into account many name phenomena that make name matching harder in general. This is an internal solution for the host company Vneuron that I developed between the months of February and June.

Report Structure

This document is organized around five chapters:

In the first chapter, we will present the host organization Vneuron, as well as a general presentation of the project that we've been tasked with, as well as define the adopted methodology.

In the second chapter, we will present existing solutions. Talk about the project's specifications and requirements analysis and define the product back-log for the different sprints we've worked on.

In the third chapter, we discuss in detail the design of our application and the many different things wee took into account when developing our solution.

In the fourth chapter, we present the hardware and software work environment, as well as present our solution and its implementation

In the fifth chapter, we'll talk about what we learned from this experience, talk about perspectives, potential next steps, potential fixes for shortcomings, as well as how we could've done things differently in order for the solution to become even more efficient.

Chapter 1

Project Overview

Introduction

Analyzing and understanding the fostering conditions for this project is important in order to define a rational and strategic approach leading to the smooth development and success of the project. This first chapter will be devoted to the presentation of the host organization in which this work was carried out, the end goals and problems that we're aiming to solve and that were entrusted to us. We'll eventually conclude this chapter with a presentation of the methodology that we adopted in realizing this project.

1.1 Presentation of the host organization

Vneuron, the company resulting from the fusion of the two companies IP TECH and AYD 10 years ago, domain leaders in their respective domains in artificial intelligence solutions. It has clients and partners in the banking, finance, insurance, health, retail, industry and governance domains. It also has subsidiaries in France and Cameroon, and has clients from many different countries and in many different domains. This places it at an ideal position to become a provider of KYC and AML software solutions to its numerous partners.

Reis™ Risk & Compliance Suite is one of Vneuron's flagship solutions. Vneuron Compliance Solutions Suite deliver all the functionality expected by financial service regulators and partner institutions. In a simple, fast, hassle free and cost effective software suite. Based on the KYC and AML approaches, it allows for better searching, investigation and risk analysis.

1.2 Presentation of the project

Names are challenging to match, they vary in myriad ways and originate from many cultures and languages and are thus influenced by many customs, they also have a tendency to vary greatly in misspellings and alternative spellings.
And while there is only one correct spelling for most general words in a language, there are often no wrong name spellings, just several valid name variations. Making exact matching of names a fruitless approach.
New and unique names are also made all the time. Thus, there are no "rules" per se to correctly spell names. Not to mention the possibility for a name to be written in a script that is different from the name's native script, hence introducing a swathe of equally valid transliterations and problems with matching them in the Roman A-Z script.
But these are not the only things that must be taken into consideration, as we also have to deal with human and system errors, handwritten forms that are OCR scanned tend to have frequent substitution errors between similar looking characters or character sequences (like 'q' and 'g', 'm' and 'r n', or 'b' and 'l i'). Manually entered data through a keyboard can have a neighboring key typed instead (for example 'e' and 'r'). Data entred over the phone especially for long and complicated foreign names (like 'Kyzwieslowski')

might even compound these problems further. Sometimes, even limitations in input fields length can force people to abbreviate, use initials, or even disregard parts of a name altogether. Sometimes, worst of all, the people themselves deliberately provide wrong or modified names. Not to mention the different Unicode characters that look the same to the human eye but are actually different.
As you can see, a good name matching solution must simultaneously take these variables into account before even starting because failure to do so may result in more false positives and false negatives than acceptable, it also needs to provide a score that indicates the relevance of the match.

The different name matching phenomena

Thre are many name matching phenomena, most important among them are the following (see Fig 1.1):

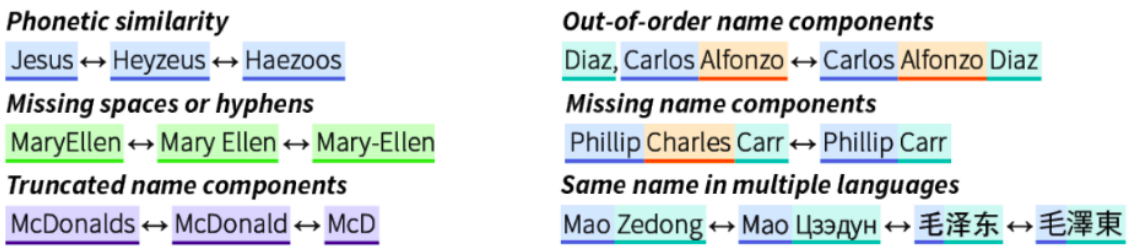

Figure 1.1: Name Matching Phenomenas[**13**]

- The first obvious phenomena that must be taken into consideration are accidental typos, letter substitutions, transpositions, character deletions or insertions, even of punctuation and white-spaces (See Fig 1.1 Missing Spaces or Hyphens) as well as similar looking unicode characters that are actually different. A name should still be matched as long as it is close enough to the original spelling-wise.
  For example, if a name component's length is between 3 and 5, that means one edit is allowed, and if the name is even longer, then two edits are allowed. This will deal with typos and 'incorrect' spellings of a name. This is what is called fuzzy matching. The fuzziness is low so that we don't actually match two totally different names
  Examples of fuzzy matching: ('O'Connor', 'OConnor' and 'O Connor'), ('Khalil', 'Kalil'), ('Cyndy' and 'Cindy')...

- The second phenomena that must be taken into consideration are the equally valid spellings of the same names, that is, the different valid ways the same name can be written and still be pronounced the same way. This is what's called Phonetic matching (see Fig 1.1 Phonetic Similarity). (An alternative approach would be to provide a dictionary of same names)
  For example: ('Gail', 'Gale' and 'Gayle'), ('Sinclair' and 'St. Clair'), ("Cairns", "Kearns", "Kerns")...

- The third phenomena that we're supposed to deal with are truncated name components (see Fig 1.1 Truncated name components), if we enter a truncated name into

our search engine, it should give slightly higher importance to the names that also start with the same characters/have the same start, instead of returning nothing, or worse, returning false or erroneous data. This might also be helpful when dealing with name initials. (We have went even a step further, names that share the same substrings/Ngrams will have a better chance of matching)

- The fourth phenomena that we must take into account is out-of-order name components (See Fig 1.1 Out-of-order name components). As you know, many young Tunisians still make the mistake of placing the First Name in place of the Last Name, unless specified explicitly to enter "Family Name", "Given Name". So we need a solution where order doesn't matter.
  Example of swapped word order: ("Ichiro Suzuki" and "Suzuki Ichiro")

- The fifth phenomena that should not be problematic is, missing name components (See Fig 1.1 Missing name components)! We should still have matches even if a name component doesn't have a match, or is simply missing. Of course, the more matching components/tokens the better.

- The last phenomena that I personally added is matching the same name in multiple languages (See Fig 1.1 Same name in multiple languages). Most names in international sanctions lists will be written in a Latin-based script, but just in case someone provides his name in a different script, we should still be able to handle it and find a match if one exists. (By first transliterating the name into the Latin script, and then matching as normal)

Aims and Objectives

This project is decomposed into these different sub-challenges that need to be solved one after the other, starting with data extraction, to developing a back-end and a front-end to help with using our search tool, as well as optimizing the way search, analysis and scoring works in general... We have followed an Agile Philosophy and a SCRUM methodology, and solved the challenges inherent in this project in 3 big sprints:

- 1. Data Extraction: The goal of this sprint is the extraction of sanctioned entities names and relevant data from sanctions lists (UN, EU, OFAC…) and to save them into Elasticsearch. As well as the extraction of Vneuron's client names test data with their relevant data from an internal file and saving them into Postgresql… In other terms, this sprint is concerned with the development of a Java application that extracts data and stores each into its respective datastore… We also developed a generic class/format for storing the names from the different sources. (Data Wrangling)
  Technologies used: Springboot + Spring Data JPA + Spring Data Elasticsearch + Dom4J + OpenCSV + Jackson + Elasticsearch + Logstash + Kibana...

- 2. Search: The goal of this portion is the development of a front-end + back-end that allows us to search for entities by entering their names (currently with a

basic placeholder analyzer)… We also wrote all the code that allows us to search for entities, as well as automated the scanning of clients from the test database with the watchlists and the storage of the search result hits into Postgresql. We have also further enhanced the mapping used by Elasticsearch. Technologies used: Angular + Spring Web + Spring Data JPA + Spring Data Elasticsearch + Elasticsearch + Postgresql + Kibana

- 3. Scoring + Sorting: In this step, we focused on writing all of our different analyzers with the purpose of solving all the different name phenomena thus improving recall as well as improving how the scoring and sorting of results works… By working on a two step approach: High recall first, and high precision second, we managed to have good performance and also lowering the risk of false positives and false negatives on our system...
  Technologies used: Elasticsearch + Elasticsearch analysis-icu plugin + Elasticsearch analysis-phonetic plugin + Elasticsearch Learning to Rank plugin...

- 4. Security and Portability: We developed Authorization and Authentication to protect sensitive data using Spring Security, JWT, Angular Auth Guard and Keycloak... We have also containerized our application with all its dependencies, added Grafana and Prometheus for monitoring among other things. And at the end, we even refactored the solutions packages and classes.
  Technologies used: Spring Security + JWT + Docker + Angular Auth Guard + Keycloak + Prometheus + Grafana...

1.3 Choice of Methodology

An adequate methodology choice is essential for the proper management of any project. In software engineering, there are two process models for the software life-cycle which are classical methodologies and agile methodologies. For the choice of a methodology, it is undeniable that no model is better than the other and that the choice is made according to certain criteria such as the nature and size of the project and the client. Traditional approaches leave no room for the unexpected. That is, if the requirements change or evolve, developers should follow what is stipulated at the start and what was agreed on before the start of the project. If the customer requests new functionality or if the project needs to be revised due to changing needs, delays and cost overruns will have to be taken into consideration. As a consequence, more than 80% of projects executed according to these classic methodologies experience delays, budget overruns, sometimes even worse, total failure, for not having been able to meet customer expectations. On the other hand, agile methodologies rely on iterative and adaptive development cycles according to the evolving needs of the customer, as well as involving the employees and the customer in the project development, making it possible to meet customer expectations in less time. These methodologies thus give an undeniable edge in productivity as well as product success and evolvability. In this project we have used the unified process and agile methodologies (Scrum) characterized by their flexibility and ease of use in large projects.

The Assumed Methodology

Faced with the complexity of the project and the field, we decided that an agile methodology would be the most appropriate, and that's for multiple reasons. Indeed, such a methodology allows for more flexibility in project management, functional intermediate products and improved customer relations.

Agile methodology

The agile methodology as a software development approach is iterative and incremental. It's based on the subdivision of a program into unitary subsystems to reduce coding complexity as much as possible. Nowadays, several agile methodolgies are used in the development of computer applications including XP, Crystal, Scrum and many others. Although all of these methodologies have the same operating principles, each has a unique set of techniques and tools.[**4**]

The four common values of agile methodologies

- **Responding to change over following a plan** Agile methodologies are characterized by the priority given to adaptability as well as accommodating possible changes rather than following an original plan. That is, adopting an iterative and incremental cycle allows teams to adapt to the context as well as to the changes that are bound to occur during a project. In other words, the initial planning and structure of the project must be flexible in order to allow the changes expected by the client; the first deliverables/artifacts of the project very often follow up on development requests.[**2**]

- **Working software over comprehensive documentation** Another common value of agile methodologies is the priority given to the production of functions rather than to documentation, because it is essential that the project works, it is a priority above all else. The delivery of an operational product of good quality because it is often tested, equipped with only strictly necessary documentation, as well as meeting the real needs of the client since deliverables are regularly submitted to their feedback.[**2**]

- **Customer collaboration over contract negotiation** Agile methodologies also advocate giving priority to collaboration with the client rather than contractual negotiation. Indeed, the involvement of the customer in the development process, allows both the customer and the developers to give their feedback on the fate of the application under development, thus canceling any "tunnel effect".[**3**]

- **Individuals and interactions over processes and tools** A fundamental notion in agile development contexts is that priority is given to people and interactions rather than to processes and tools. It is better to have a close-knit, communicating team made up of developers of different levels, than to have a team of experts who work in isolation.[**2**]

The Methodology of Choice: SCRUM

At the very beginning of this project, it was important to choose which agile methodology to adopt for the development of the application that is the subject of this report. In order to meet the evolving needs of the client, which may change due to changing contexts, we have chosen to use the SCRUM methodology (the methodology retained by the host company Vneuron as the internal work methodology for the design and development of projects.) in order to complete our project. Above all, this methodology makes it possible to deal with the various problems that surface during the execution of a project, ranging from poor planning to lack of communication between the various stakeholders, as well as frequent changes in needs. This is done by ensuring sustained collaboration between the project team and the client and the rapid delivery of technically stable software that is easy to use and modify.[**5**]
SCRUM allows us, if necessary, to add or modify functionalities at the level of a desired sprint without affecting the others. In addition, this methodology allows for better respect of the planned dates to close the different Sprints and not to exceed them.[**6**]

Scrum is beneficial to team leaders as well as to the customer since it allows for:

- More flexibility and responsiveness.
- A greater capacity to adapt to changes due to short iterations.
- The most important thing about SCRUM is that it takes into consideration both the theoretical and practical aspects and is very close to reality as a project planning methodology.
- Encouraging staff commitment by involving the entire team and calling upon them to participate constructively in the definition and distribution of tasks and planning so as to maximize the level of awareness and motivation, involvement drives motivation, that is through the team's involvement in the decision making.
- Improved visibility over the entire project: by making all information about the project accessible to the entire team.
- Improved quality of final products by reducing bugs and their scope.
- Giving the customer more flexibility in setting application priorities as some of them may only appear at advanced stages of development.
- Prioritizing the quality of the product rather than focusing on the deadline delivery of the latter.

We have performed a preliminary study in order to distribute the Stories of the product backlog by priority. We then relied on the following standards:

- Feasibility
- Dependence
- Complexity

- Priority in relation to our needs

The iterative incremental Scrum methodology is currently the most widely used, documented and trusted agile methodology today. Indeed, the principles of Scrum come from Rugby, hence this name which means "mêlée" in French. Like rugby, this methodology emphasizes team spirit to accomplish common goals. It is based on an iterative and incremental methodology for the development of IT products or applications. Scrum is essentially based on work loops that adapt to Customer feedback. This methodology favors decision making based on actual results rather than speculation. With Scrum, the duration of the project is spread over several work slices called sprints. These sprints can last between two to four weeks. During these different stages, the product should always be properly implemented and tested. A meeting between the Client and the project team is scheduled at the end of each sprint to present progress, collect feedback and plan next steps.[**7**]

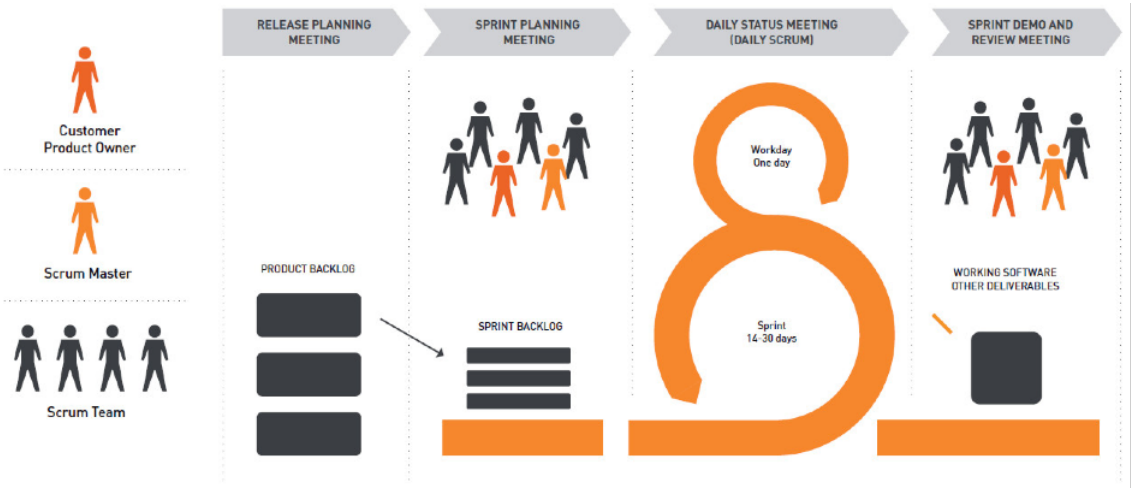

Figure 1.2: Scrum Methodology

Broadly speaking, Scrum defines a minimal set of actors, ceremonies and artefacts that make it possible to meet the main challenges of incremental development: planning, time management and obstacle management. Scrum is entirely driven by Business Values. Risk management, in particular, is carried out through this prism. Scrum defines three roles for a project.

- The "Product Owner" is the member of the Scrum team responsible for maximizing the value of the team's work. He is the official representative of the client within a Scrum project. He describes what needs to be done and why, but not how. He defines the needs of the product and can write the specifications. He is also responsible for defining and prioritizing "user stories" for each sprint. In our project, Mr. Ahmed Ben Romdhane my professional supervisor has played this role.

- The "Scrum Master" (the product manager): this is the person who must ensure the smooth running of the various Sprints of the release, and who must imperatively master Scrum. Mr. Ahmed Ben Romdhane, will also play the role of Scrum Master, although it is not recommended to occupy both positions at the same time, but this

is an internal project with a development team of 1, which allows for some simplifications, he acts as a facilitator between the development team and the Product Owner. The main task of the Scrum Master is to resolve diverging views between the team and the Product Owner while keeping the team responsive to customer feedback and creative in problem solving. He is also the Product Owner's advisor on how best to get the most out of the project team.

- The "Development team members": made up of people who will be responsible for implementing the various customer needs. Obviously, this team is responsible for delivering project artifact (deliverables) at the end of each sprint. In our project, the team is made up of myself alone. As defined by Scrum founder Jeff Sutherland, "the team has no leader". It is the team itself that organizes itself and establishes these tasks and the methodology to be followed in order to accomplish them.

With such a satisfactorily chosen methodology, we need a unified modeling language for modeling our project. To design our system, we chose UML (Unified Modeling Language) as a modeling language. Our choice was based on the strengths of this language, in particular its standardization and the various diagrams it offers. Also UML presents the best tool for schematizing complex systems in a simplified and standardized graphical and textual format. Indeed, UML is neither a process nor an approach, that's why we had to choose a methodology of design and development that we must adopt, and that was the C4 Model giving us the ability to present our solution through an "abstract-first" approach to diagramming software architecture, thus presenting us with both a high level idea of how the different parts of our project work together as well as a look into the details. Indeed the C4 model relies on both UML and ERD and is influenced by them.

Conclusion

This chapter has been devoted initially to the presentation of the company Vneuron as well as the Reis Development Team within which this project was done, the general framework of the achieved work and the objectives and problems that our work aims to solve, together with the chosen methodology to be used.

Chapter 2

Analysis and Requirements Specification

Introduction

In any system, functionalities must satisfy the needs of its users. These needs define the functions that users expect to be delivered by the system. The requirement specification forms the foundation on which the system architecture is built. Throughout this chapter we will define the needs that we need to satisfy, which is something that will facilitate sprint planning.

2.1 Existing Solutions

In this section, we review the web application that the company has developed and that client agencies use, as well as other systems that exist in the market.

2.1.1 The System currently in use: Reis KYC

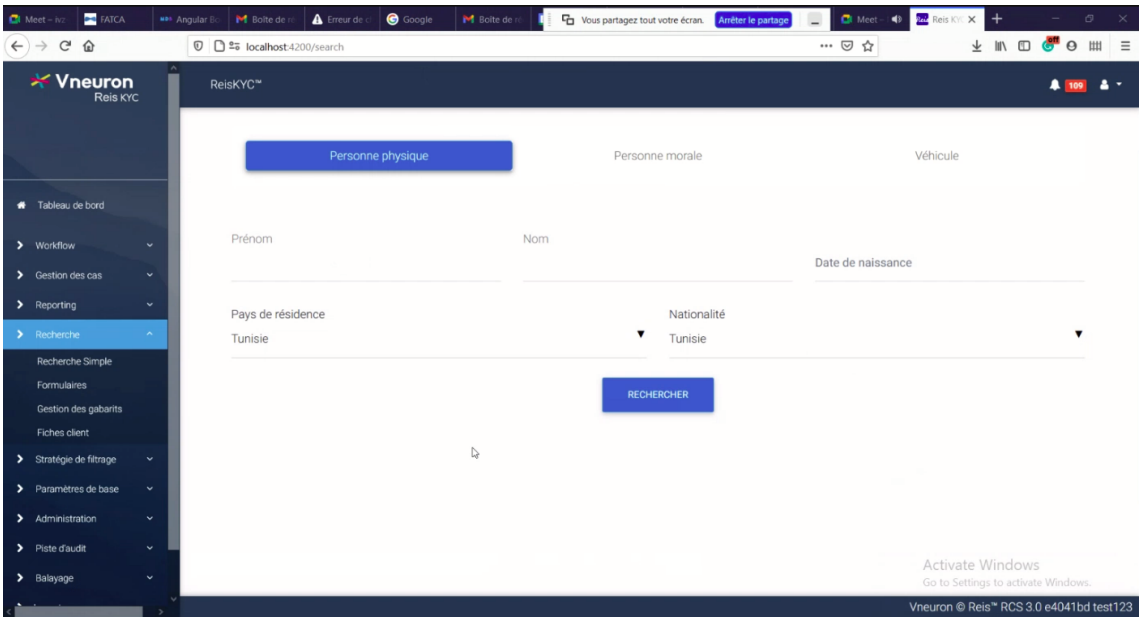

Figure 2.1: Search Interface of Reis KYC[**15**]

The system that's currently in use by our company, Reis™ Risk & Compliance Suite (See Fig 2.1 & Fig 2.2), which is also Vneuron's flagship solution to many of its clients, allows users to search for entities by specifying many details about the entity, such as the type of the entity, the name of the entity as well as some other attributes like nationality, country and date of birth. And it returns high risk/highly matched entities first, as well as specifying from what lists they are on through the use of tags (thus the black tags on top: PEP, OFAC..., which I guess requires doing some identity resolution to find the same individual in multiple lists, if we need different tags on the same entity). It also returns an Image of the individual if one is found. You can also see that the score is shown as a percentage.

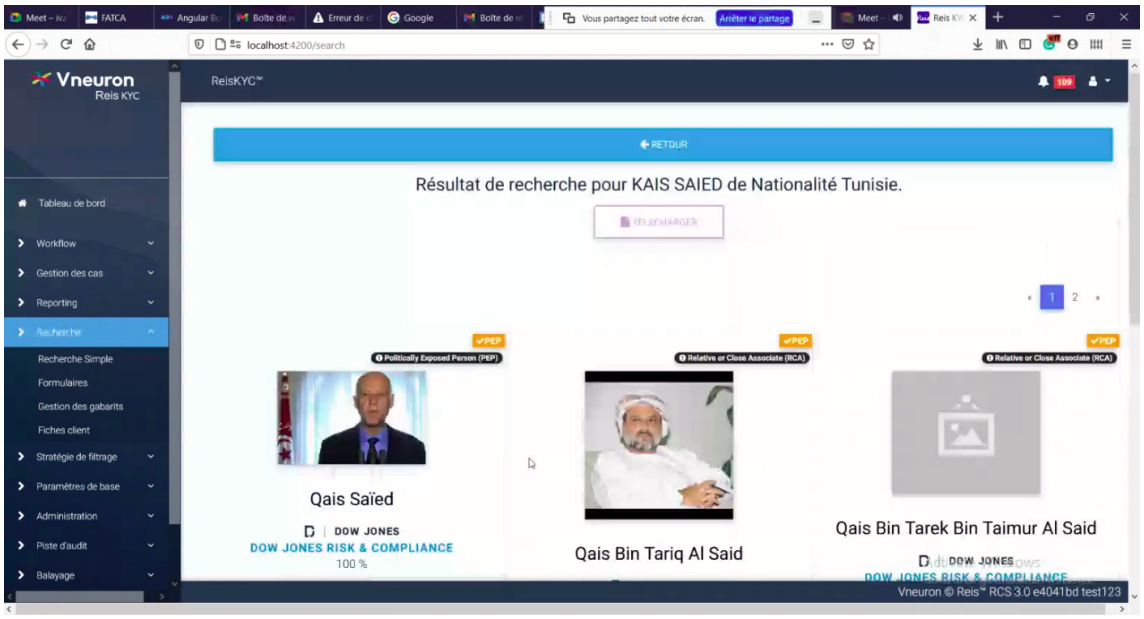

Figure 2.2: Search Results of Reis KYC

2.1.2 Alternative Solutions

There are many other leading risk and compliance solutions, such as LexisNexis KYC, Refinitiv, Red Flag Group, Unite US. Many name matching solutions are indeed in-house made. But many of these big names have their solutions powered by Rosette Name Indexer from Basis Technologies (See Fig 2.3). It's a good solution that even many federal agencies in the United States like the department of homeland security, department of defense and the office of the director of national security are using their solution instead of developing one themselves as well as many government departments, not to mention that LexisNexis KYC, Refinitiv, Red Flag Group, Unite US, all providers of KYC solutions are also using Basis Technologie's Rosette Name Index solutions to their advantage, as well as many other global corporations such as Airbnb, Sita and Toshiba that are not in the market of providing KYC... The reasons for the popularity of this solution is that it goes above and beyond with name indexing and matching, even taking into account the linguistic side of name variations for over 14 languages, something that no competitor solution can duplicate without domain expertise in linguistics.[**51**][**54**] Their solution is provided as a simple Java library, a Lucene plugin, a Solr plugin and as an Elasticsearch plugin.[**55**] My plans are humble in comparison. We plan on providing a generally competitive set of functionalities with whatever knowledge I was able to learn on Elasticsearch during the internship, that is, through the use of the provided data types, character filters, tokenizers, token filters, as well as my own developed analyzers. We'll also provide the solution within a Spring Boot application and an Angular Front-end to simplify testing and using it and to simulate what a real world solution needs to do.

2.2 Requirements Analysis

In this section, we will present our system's functional and non functional requirements.

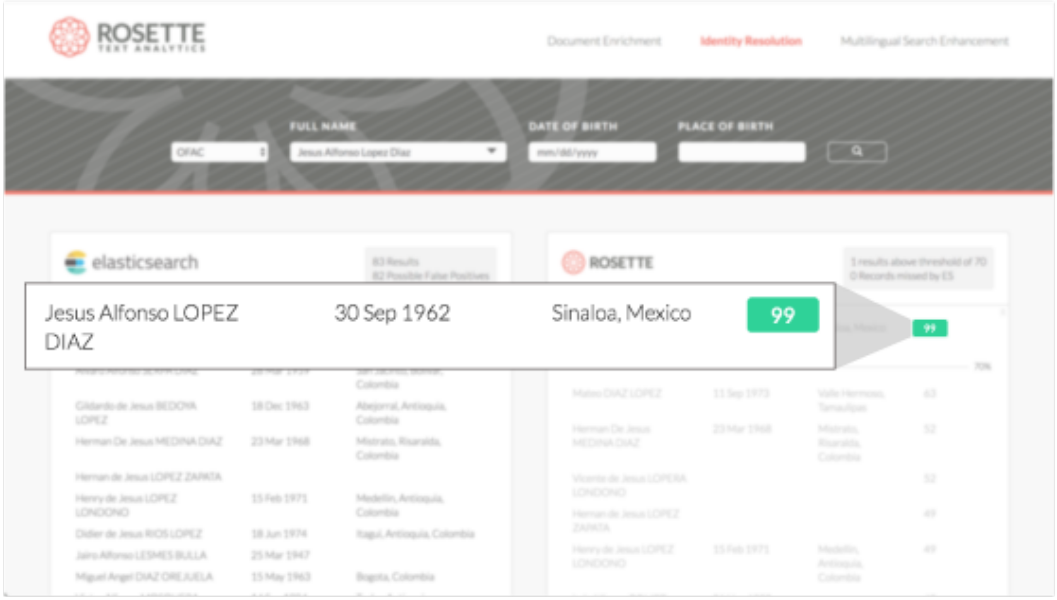

Figure 2.3: Search Results of Rosette Name Indexer

2.2.1 Functional Requirements

The aim of this project is the development of a web application allowing for name matching with watchlists for internal use by Vneuron, which is achievable by meeting the following requirements:

The System

- Automatically extract data from economic sanction lists
- Automatically extract relevant data from the customers list (test file)
- Automatically save extracted Watchlist data into Elasticsearch for easier searching
- Automatically save all the extracted Clients data into Postgresql
- Develop a backend that expose a Rest API that allows external applications/the application's front-end to search for sanctioned entities by name (without going into the details of how elasticsearch works or specifying an Elasticsearch search query)
- Atomatically search for matches between customers and the sanctions lists as soon as both data are stored in their respective databases, and store the search results/hits into Postgresql
- Povide good Mappings for the data that is generic for most economic sanction lists
- Provide good Analyzers that could solve most of the name phenomenons and challenges and deal with them

- Provide a good search request that combines the analyzers in such a fashion as to take into consideration all the name phenomena by giving a score boost to the more prioritary phenomena we want to deal with

2.2.2 Non-functional Requirements

In addition to the essential functionalities that must be carried out by our application, we must respect certain constraints and meet certain non-functional needs:

- Usability: The front end of the application should be user-friendly.
- Performance: must ensure the speed of search request and their reliability and be able to handle a large number of records.
- Technical requirement: the application must be developed using the Java language, we will use the Spring framework and many of its sub projects like spring boot, spring data and spring security for the backend; Angular for the front-end side; and Elasticsearch as a search engine and PostgreSQL as a database.
- Portability: Our application must be portable and usable on replicable systems (all dependencies should be specified without the need to hunt them down later on as well as enabling easy deployment)
- Security: Our application should provide users with simple security, authorization and authentication features to protect our data and application from unauthorized access

2.2.3 Product Back-log

The Scrum methodology defines the management of needs in the form of a list that presents all the expected features of the product. Features are often noted by a code or number to distinguish them more quickly. All of these features are called the Product Backlog. Agile development is based on the iterations which are named as part of Scrum the Sprints Backlog, each iteration has a set of features. Table 2.1 shows an extract from our project backlog:

Sprint	Backlog Feature
1. Data Extraction	Extraction of sanctioned entitie's names and relevant data from sanctions lists (UN, EU, OFAC…) and saving them into Elasticsearch. Extraction of Vneuron clients names and relevant data from a test file and saving them into Postgresql… In other terms, this sprint is concerned with the development of a Spring-based Java application that extracts data and stores it into its respective datastore… Technologies used: Spring Data JPA + Spring Data Elasticsearch + Kibana + Logstash Libraries used: Jackson + Dom4J + OpenCSV
2. Search	Development of a web application (front-end + back-end) that allows us to search for entities by entering their names (testing and using the application in a user friendly manner)… Automate the process of matching clients with the watchlists which happens as soon as the data is loaded into their respective databases and the the storage of the search result hits into Postgresql. Technologies used: Angular + Spring Web + Spring Data JPA + Spring Data Elasticsearch + Elasticsearch + Postgresql
3. Scoring & Sorting	Writing all our analyzers with the purpose of solving all the previously mentioned phenomena as well as improving how the scoring and sorting of results works… By working on a two step approach: High recall first, and high precision second, we can manage to have good performance and lower the possibility of returning false positives and false negatives... We also improved the mappings and search queries to take advantage of the new analyzers Technologies used: Elasticsearch Plugins Used: Phonetic Analysis Plugin + ICU Analysis Plugin + Learning to Index Plugin
4. Security, Portability and Monitoring	In this step, we developed Authorization and Authentication for both the front-end and back-end to protect sensitive data using Keycloak Identity and Access Management solution, Spring Security, Angular Auth Guard and JWT. We have also fully dockerized the application inside docker containers to help with portability, and added Prometheus and Grafana in order to monitor our application Technologies used: Spring Security, Angular AuthGuard, Keycloak, Docker, Prometheus, Grafana

Table 2.1: Sprints and the backlog of tasks.

Conclusion

Throughout this chapter, we have looked into existing tools and solutions, most prominent among them was the host company's Vneuron Reis™ Risk & Compliance Suite, to satisfy the demands of users who need name matching and searching functionality, we have also seen a strong alternative commercial solution which is Basis Technologies Rosette Name Indexer. The next step was to collect, analyze and specify the functional and non-functional needs of our project. At the end of this chapter we have developed our product backlog in which we have split the sprints into "deliverable version" releases that we are going to finalize. We will now move on to drafting the specifications for the project.

Chapter 3

Design

Introduction

With the analysis and requirements specification finally performed, we shall elaborate the software design; a crucial phase in the development process. We will outline, first of all, some of the problems that we ran into and propose suitable solutions to them. Afterwards, we will focus on designing an appropriate application structure. We will present in a first occasion the application's architectural design : which is more of a general structure of the system to be conceived. In a second occasion, a detailed design: basically a refinement of the overall design of the different components will be presented.

3.1 Problem Analysis and Solution Design

In this subsection, we will describe some of the technical problems we encountered and give glimpses to the solution design to deal with them.

3.1.1 Dealing with large XML files

Our input files are big XML files and their size might exceed some computational limitations for some file systems and raise performance questions for the optimal manner to handle them. We found that the DOM4J Library is a suitable solution to deal with Watchlist XML files due to its performance and time efficiency. Providing us with a StAX parser that doesn't need to load the entire file in memory.
The observer pattern is highly compatible with this behavioural procedure. In the end, parsing the entire 600k lines XML files and building a corpus for the watchlist data takes less than 5 seconds in execution time.[**8**][**9**][**10**]

3.1.2 Complex search setting

Due to the number of name phenomena we'd like to take into consideration, as well as the large number of individuals in the watchlists, We definitely need a search and analysis engine, one such engine is Elasticsearch (alternatives could be anything Lucene based, such as Solr. But Elasticsearch is the easiest and is widely supported, offering lots of plugins that we can use.) We have used Elasticsearch, together with two plugins: Phonetic Analysis plugin and ICU Analysis plugin. As well as implemented a generic mapping with suitable Analyzers meant to help us match names across a wide range of name phenomenons.

3.1.3 User Friendly Environment

Its not enough to just work with Elasticsearch, we also need to develop a web application that allows people who are not skilled with Elasticsearch to search for names. For this, we've used the Spring and Angular Frameworks to hide the details of our solution from end users.

3.2 Architectural Design

In this section, we will present the software architectural design; as well as a global view of the solution; and we will highlight its modular organization.

3.2.1 Global Application Architecture

The application can be partitioned into these parts :

- Data scraping and data wrangling; consists of extracting the sanctioned entities from the sanctions lists XML dumps. And then converting data from a raw format; the extracted entities in XML format; into another format; a generic format to be used by Elasticsearch: by creating a Mapping that allows for more convenient processing of the data. (Despite the many different formats of XML inputs, we only have one universal document format in Elasticsearch)

- The development of the Elasticsearch's search: Here, we developed the mapping and analyzers to be used in our solution. Tested out the search query and improved the returned score. We have so far introduced about 11 analyzers that will hopefully take into consideration all possible name phenomena.

- The Development of a back-end around the previous sections; exposing a search REST API. Thus hiding the details of the Data stored in Elasticsearch and Postgres, by hiding the search query and the mapping fields and analyzers. As well as the relevant security code portions allowing the authentication and authorization of any attempts to use the rest api.

- The development of a front-end that consumes the back-end's REST API, allowing for searching entities by their names. As well as the relevant security code that allows it to authenticate and authorize access using keycloak.

- Keycloak as an identity and access management solution

  Our application was finally made into a containerized solution with all its dependencies placed within docker containers, thus helping with our program's portability.

3.2.2 MVC Design Pattern

Model-View-Controller is a well known architectural pattern; developed by Trygve Reenskaug in 1978-79. (See Fig 3.1) MVC separates the application into three interconnected parts:

- Model : encapsulates the fundamental behavior and data of the application domain

- View : renders the information into a human readable format displayed on the user interface

- Controller : manages the communication between the view and the model through the interpretation of the user input to requests for actions to the model

Due to its flexibility, this design pattern is suitable for interactive systems and can enhance the maintainability of the software.

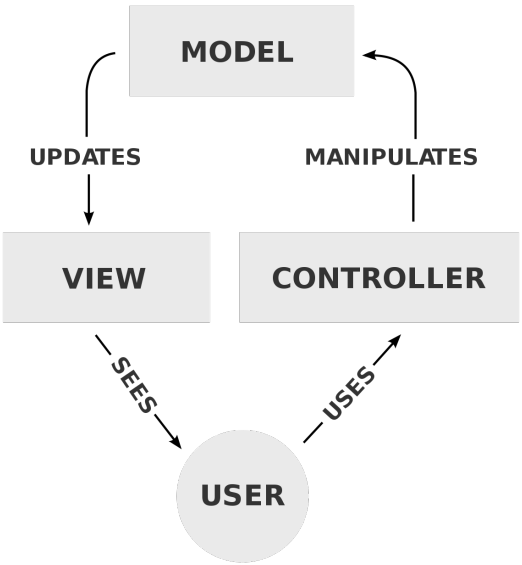

Figure 3.1: Model View Controller Design Pattern.

3.3 Detailed Analysis Diagram

In this section, we will elaborate the software detailed design; a refinement of the preliminary design presented earlier. We have used the C4 model because of its abstraction-first approach to diagramming software architecture, that means, we'll go from the general context of our application to the specifics.[**1**]

3.3.1 Context

This diagram (See Fig 3.2) shows that our system works in isolation and independently from any outside software systems. It is a system that is meant to be used by people who wish to search for names in watchlists and verify new names existence (like that of new customers for financial compliance).

3.3.2 Containers

This diagram (See Fig 3.3) shows the many different interconnected and importants parts of our solution, it's made up of a single-page front-end application developed using Angular (and deployed using Nginx). A spring-framework-based back-end that interacts with both Postgresql and Elasticsearch playing a central and primordial role in our application. And

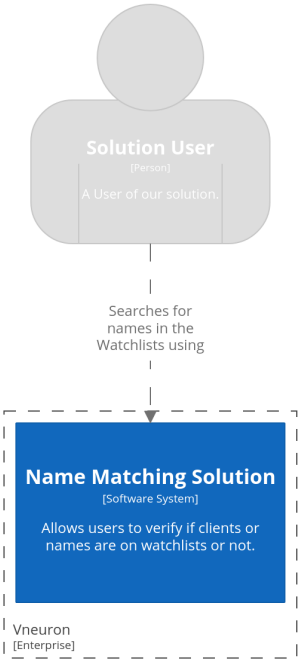

Figure 3.2: A High level view of how our system

Keycloak that plays the role of an authentication and authorization service. As well as two monitoring solutions, mainly Grafana and Prometheus.

3.3.3 Components

This diagram (See Fig 3.4) dives deeper into the API application/backend and further decomposes it into its constituent parts. It shows the many important parts of our application, mainly the security component that only allows authorized access to the REST API, the file parser component that takes the watchlist files and the client file and parses them into the generic formats that were agreed upon, we can also see the sanctioned entities search component that allows us to perform our complicated search requests without needing to specify the details as an Elasticsearch query requires and that is also protected from unauthorized access and usage, and lastly, we see the Customer Due Diligence component (balayage automatique des clients contre les listes noir) that scans each and every client against the watchlists and stores the search results into Postgresql.

3.3.4 Code

There are many different components used by our application, but the most simple way to understand the way our project works is by the design patterns used. We have mainly

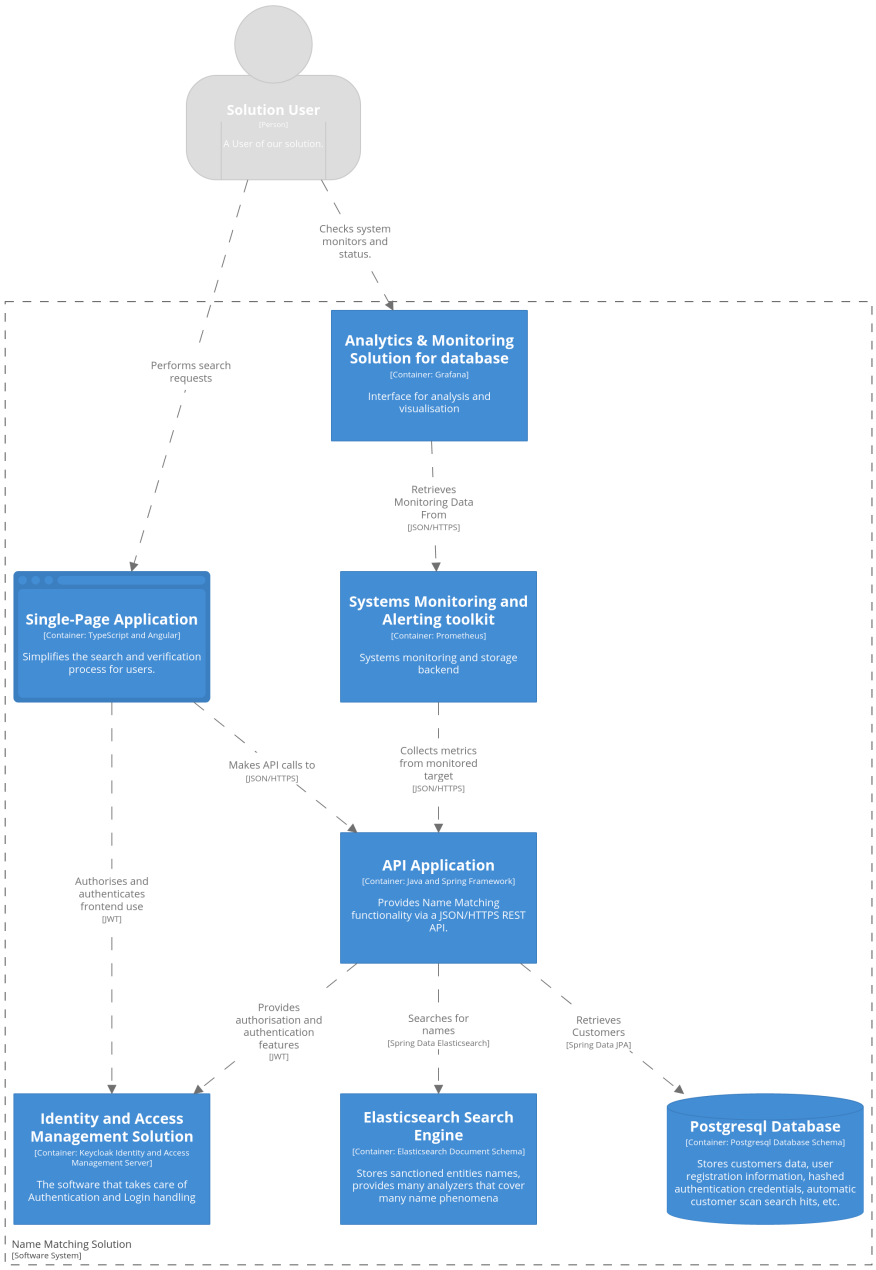

Figure 3.3: The different containers and parts that are used by our solution.

used the repository pattern as well as the service layer design pattern for our backend:- The repository pattern is a strategy for abstracting data access to both Postgresql and Elasticsearch, as well as to reduce code duplication that will happen if we don't use such a pattern. It's like putting a universal adapter in between our application and the data so it doesn't matter what data storage technology we use and can change it very easily (since we're also using Spring Data. As for the Service Layer Design Pattern, we used it because it provides us with the much needed separation of concerns, as well as security

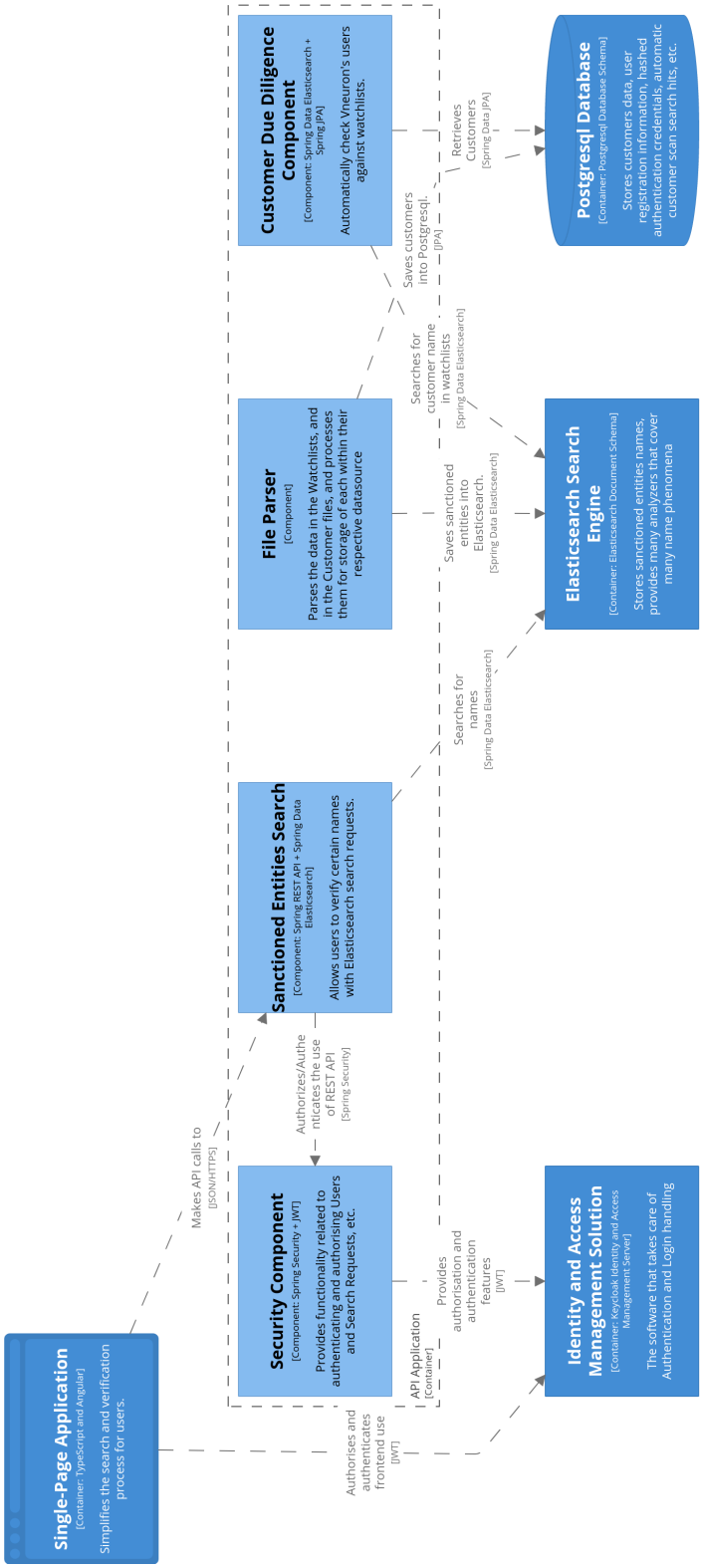

Figure 3.4: A Decomposition of the API Application Solution

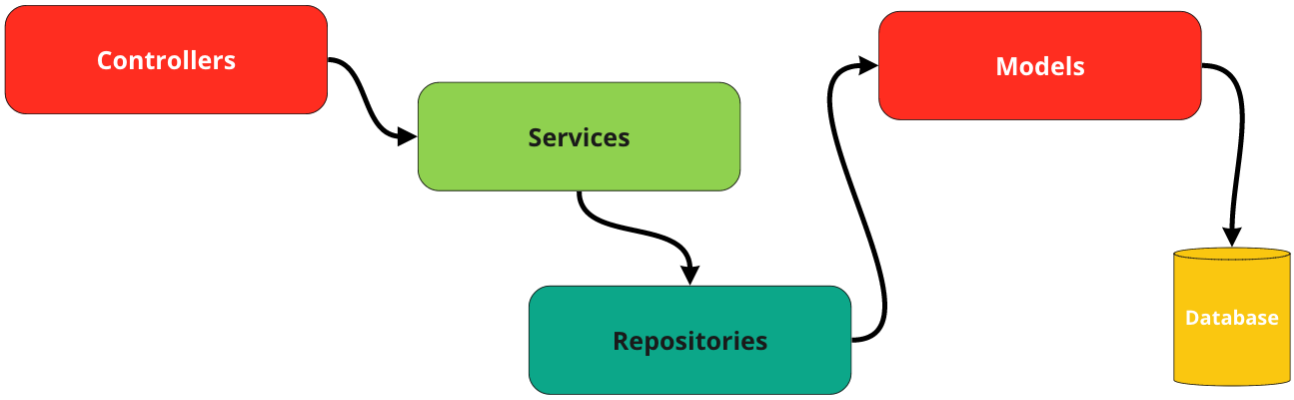

Figure 3.5: Repository Pattern and the Service Layer Design Pattern used by our backend

(since access to the database will have to pass from the controller that offer the REST API, to the services), as well as loose coopling. (The controller and respository's code doesn't have to change, only the services code can change through a different usage of the repositorie's API)

3.4 Supplementary diagrams

Supplementary Diagrams help provide a better picture on our proposed solution.

3.4.1 Dynamic Diagram

Dynamic Diagrams are similar to UML Communication Diagrams/Sequence Diagrams as it helps to demonstrate how the different elements collaborate together at runtime to implement our desired user stories. The following diagram (See Fig 3.6) shows how the different components communicate together in order to authenticate a user when he tries to access the front end.

This diagram (See Fig 3.7) demonstrates how searching for clients, even at the backend level works, and how things are protected and require an authorized user to access the data and to perform the search query.

The following diagram (See Fig 3.8) shows how the Automatic Scanning of customers takes place by selecting the data stored in the database, then emits the search requests to the backend, and lastly stores the search results back to the database.

3.4.2 Deployment diagram

This deployment diagram allows us to illustrate how our containers are mapped to infrastructure. It is based on the UML deployment diagram. and shows our software solutions different containers dependencies and how it looks like when deployed. (See Fig 3.9) A deployment node in our case is a containerised infrastructure (e.g. a Docker container),

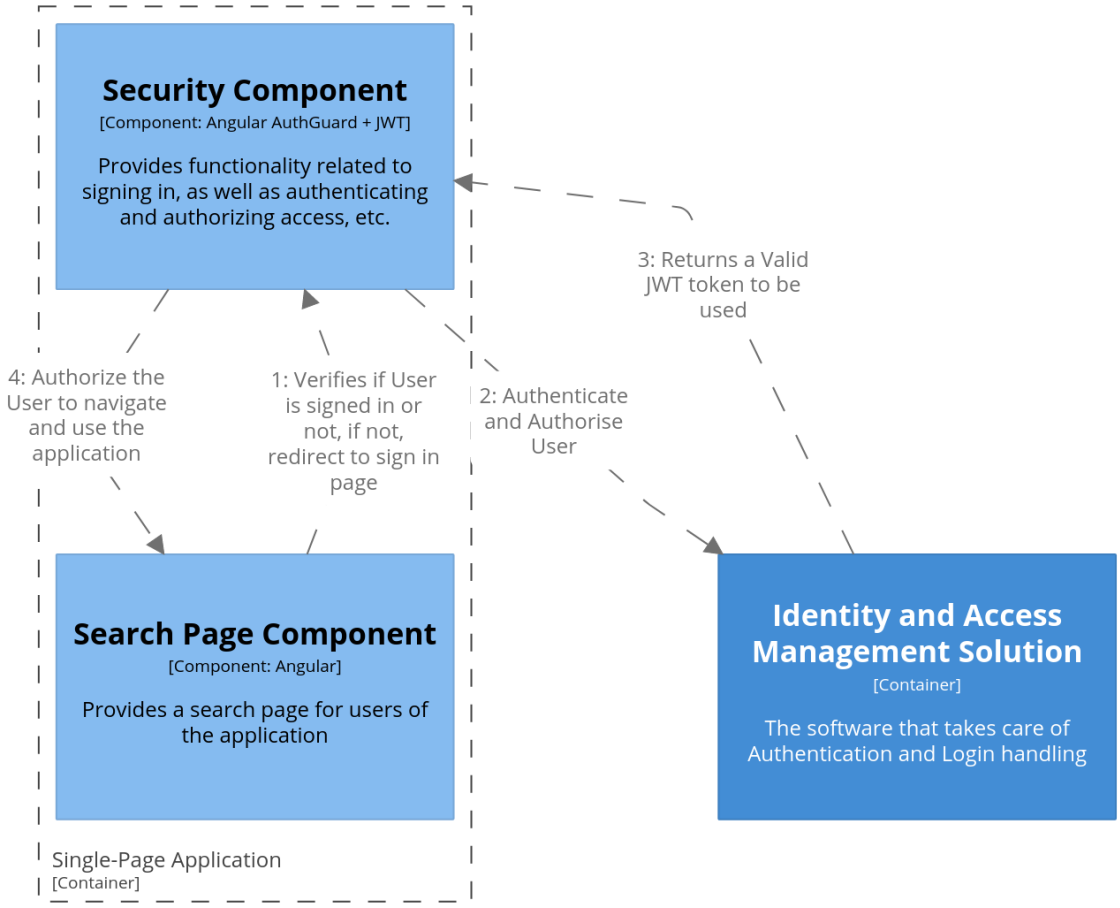

Figure 3.6: Security - Front-end Authentication/Authorization Login

within which an execution environment is running (e.g. a Postgres database server, an Elasticsearch search engine server, Java Spring Boot Web Application server, Keycloak Authorisation and Authentication server...):

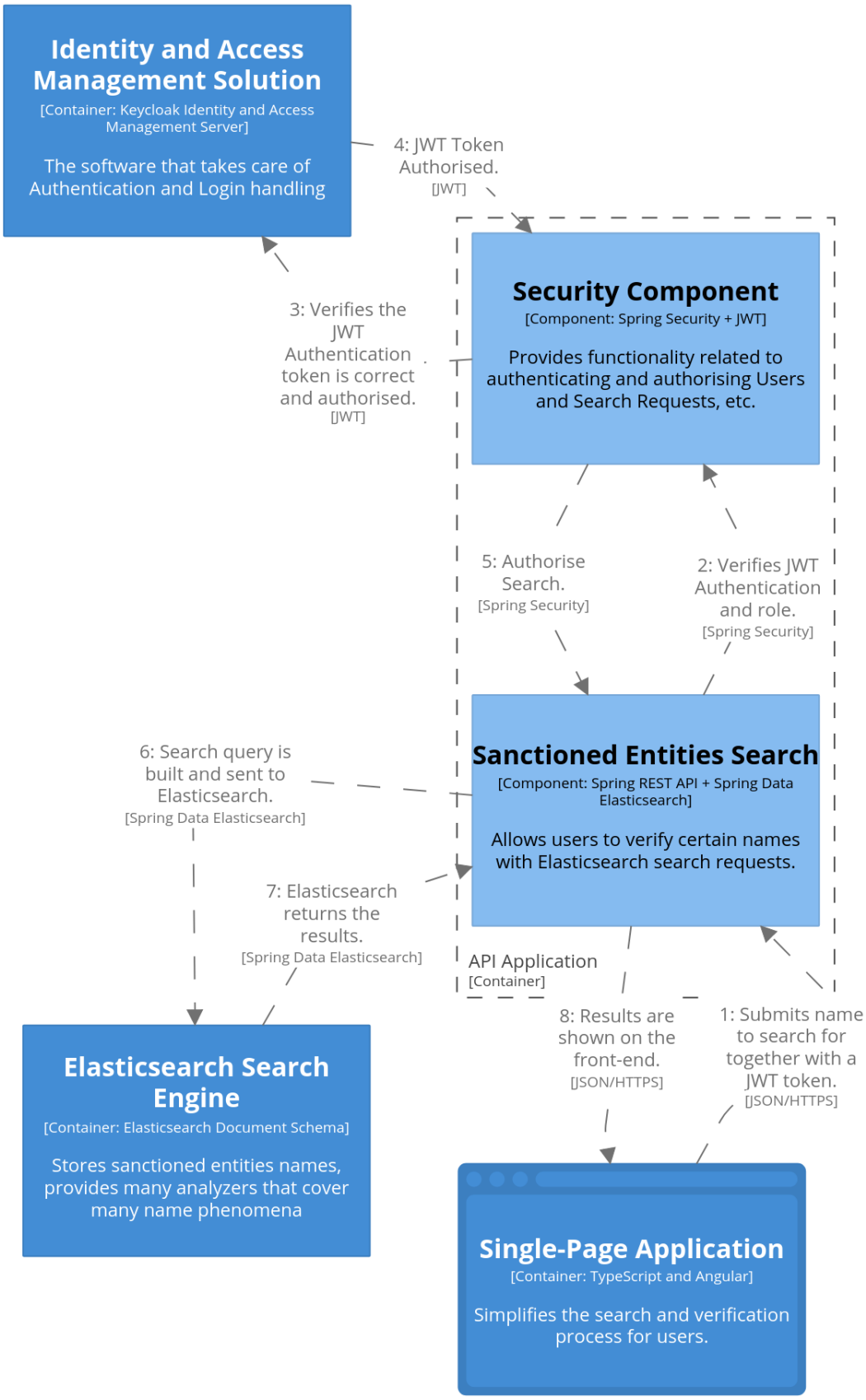

Figure 3.7: Search Request Sequence Diagram between the front-end and the back-end

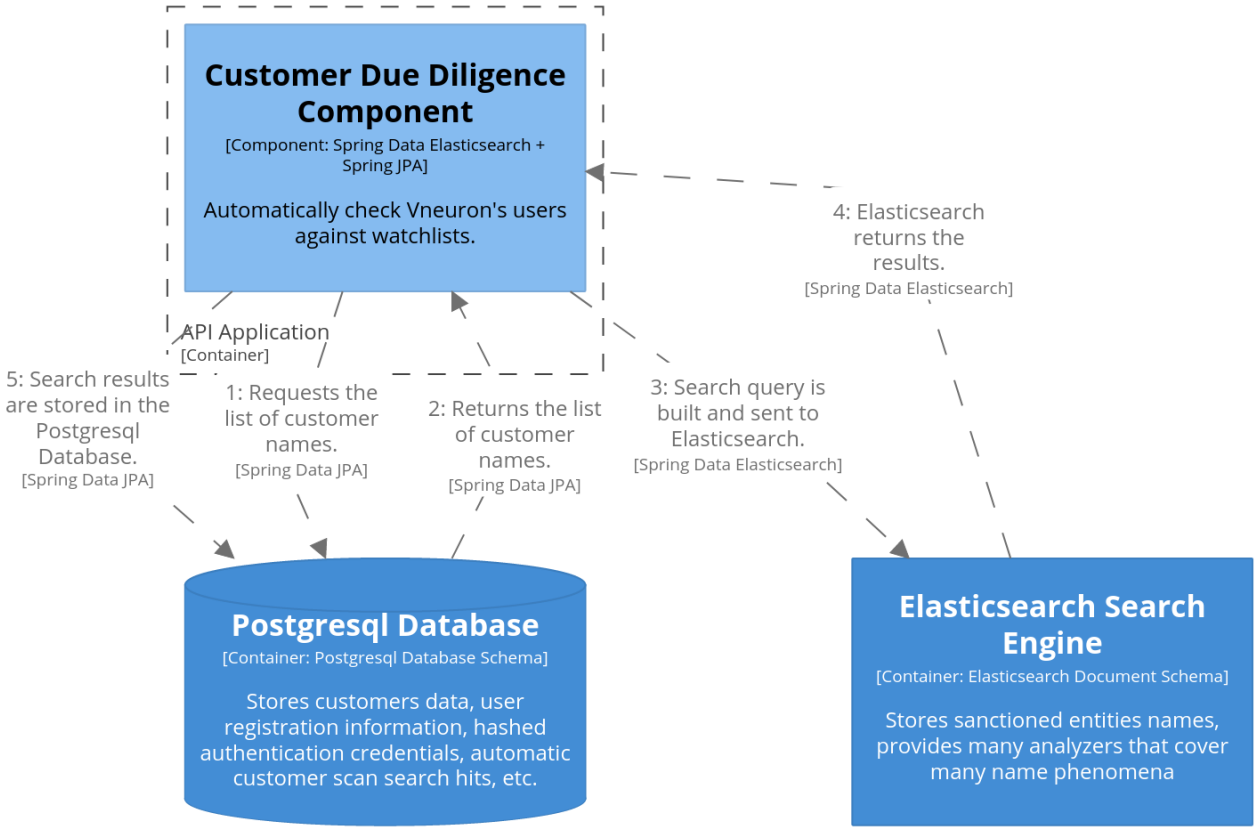

Figure 3.8: Customer Due Diligence

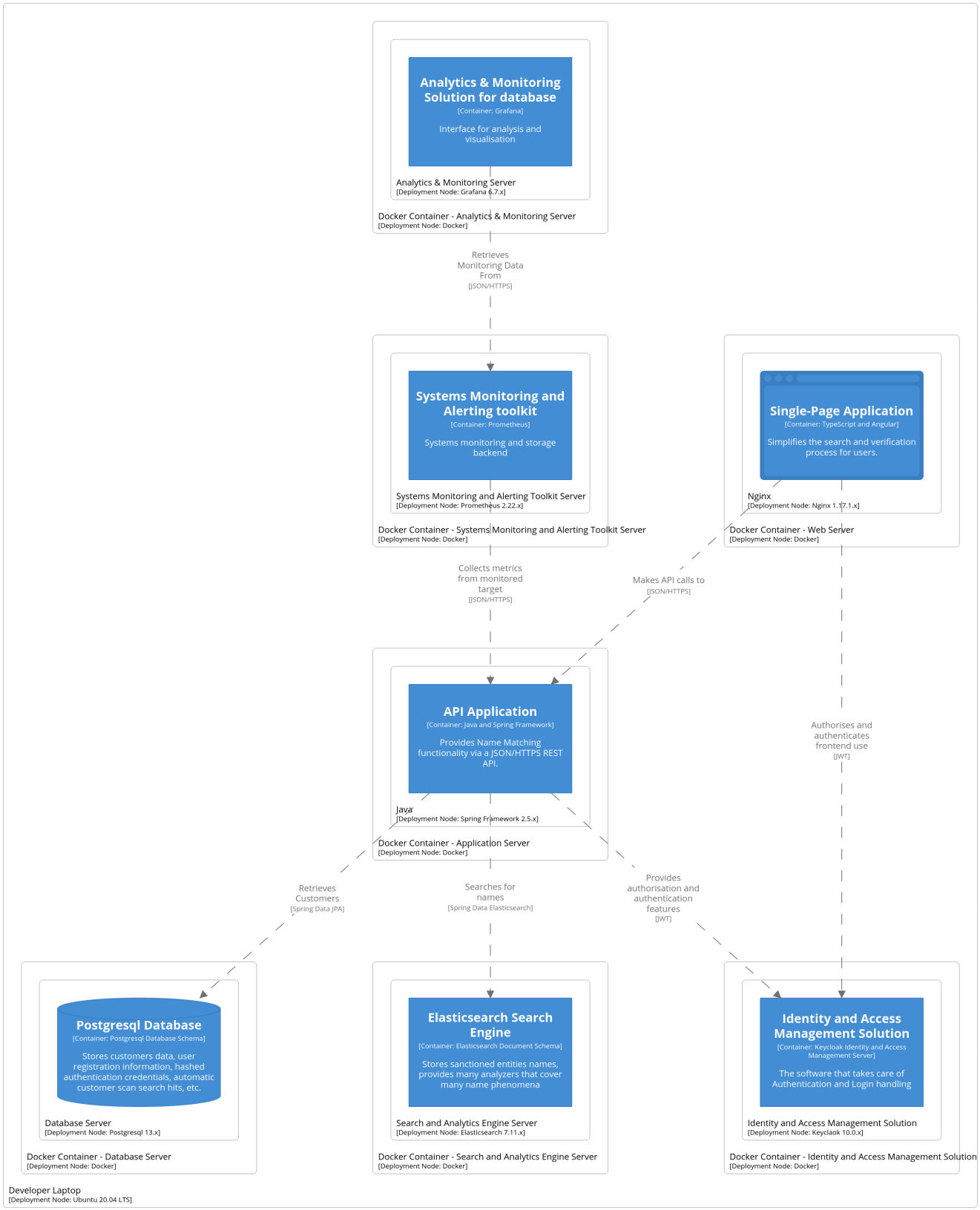

Figure 3.9: Development Deployment

Conclusion

In this chapter, we did sketch out some solutions to the issues we came across. Later on, we present both the architectural and detailed design of the final product. This will ease the process of implementing and building of our solution as detailed in the next chapter.

Chapter 4

Realization

Introduction

This phase consists of implementing the conceived design. We will present the work environment from both hardware and software perspectives. Afterwards, we will describe the work performed during the internship.

4.1 Work Environment

The software was developed using these technologies.

4.1.1 Hardware Environment

This work was performed on my personal laptop, which is a DELL Inspiron 5570 computer with the following characteristics:

- Memory : 32 Go
- Processor : Intel Core i5-8250U 8è Génération, up to 3.4 GHz, 6 Mo de mémoire cache
- Graphics Card : AMD Radeon 530 / Intel UHD Graphics 620 GT2
- Disk capacity: 1,25 To (256 Go SSD + 1 To HDD)

4.1.2 Software Environment

In this section, we enumerate the several software technologies we used throughout the development cycle.

Operating System

Throughout our project we used Pop!_OS 20.04 LTS, which is a Ubuntu-based linux distribution, to develop the backend, the front end and to develop many of Elasticsearch's analyzers that we use. For the dockerization part and security parts though, we had to use stock Ubuntu 20.04 LTS due to the better docker support and community presence.

Programming Language

We have used Java 8, which is an object oriented programming language. Java's main selling point is portability given it is run in a JVM java virtual machine irrespective of the operating system or computer architecture.

We also used TypeScript 4.2.4 as it is the language of choice for Angular. It adds optional static typing to JavaScript, to which it is a strict syntactical superset. It is designed for use in developing large applications and can also transcompile to JavaScript.

Integrated Development Environment

We have used IntelliJ IDEA Ultimate due to its professional nature, superior support for the Spring Framework, ease of use as well as the free license given to students like myself. Its main use case is developing Java enterprise applications.

Visual Studio Code is another IDE that we used. Its Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. In the last few days of the internship, we have also used WebStorm, another JetBrains IDE, and are really grateful for the free license given to students! This IDE is really easy to use and of superior quality and feel and responiveness compared to Visual Studio Code. Offering many IntelliSense features like automatically generated code, better code analysis and more relevant correction suggestions.

Parsers and Libraries

We have used the Dom4J open-source library for working with the watchlists XML data, and since it is compatible with the StAX standard, it can load the large XML files even inside low memory environments (like docker containers).

We have also used the OpenCSV library for working with the Demo Clients CSV files.

During the early stages of our work, we also used Jackson for generating the JSON documents that we used to send manually to Elasticsearch through Logstash or the command line.

Spring Framework

The open-source Spring Framework helps immensely with Java Enterprise Web Development, since it lets us focus on application-level business logic, it also abstracts away the hunt for dependencies of compatible nature and does all that magically under the hood.[**60**][**67**][**73**] We have particularly took advantage of the Spring Boot/Web for the REST API, Spring Data JPA (Postgres), Spring Data Elasticsearch (Elasticsearch) and Spring Security.[**69**][**71**] The Spring Framework is really a life savior for Java Enterprise Developers.[**72**][**73**]

Elasticsearch

Elasticsearch sits at the heart of our solution. It's a distributed multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. We have used Elasticsearch due to these three magical words: speed, scalability and resiliency.[**11**][**24**][**34**]

Let me explain, Elasticsearch is a near-real time search platform, and it excels at full-text search since it is Lucene-based, making it very well suited for time-sensitive use cases such as name matching (security analytics).[**38**][**40**] It was also built with scalability in mind providing baked in solutions for dealing with hardware and software failures, making

it super easy to use and scale without much configurations or risks. Not to mention the wide set of built-in features as well as the entire Elastic stack products and plugins and packs that can be used with it.[**41**][**42**]
We have used it to store the watchlists data, to index them, as well as to perform searches on the stored data.[**64**][**65**][**71**][**72**]

Kibana

We have used Kibana all along the way since it provides the Dev Tools that we used extensively to test many features and aspects of Elasticsearch, which was an essential tool helping us build our analysers, character filters, tokenizers and token filters one by one as well as to test and build our serach queries as we were building them.[**71**]

Logstash

Logstash is a data processing pipeline (ETL) that ingests data from multiple sources, transforms it, and then sends it out to multiple outputs or stashes. We have used it in the early stages with the hope that we might be able to develop an all Elasticsearch solution, and though we weren't successful, we did learn and use this technology and that in itself is very valuable.[**12**][**18**][**71**]

PostgreSQL

PostgreSQL uses an object-relational data model, supports user-defined objects such as data types, functions, operators, domains and indexes giving it an advantages over other SQL databases. It can for example, store multi-dimensional arrays among other things, making it perfect for our use case where we need to store alternative aliases and names instead of creating a separate table with a row for each of the aliases which is the case in normal SQL databases. As well as it's ability to store and manipulate JSONs, making it a perfect solution for development teams with changing requirements. These, among other things, were the reasons for why we chose to use PostgreSQL. We have used it in particular as a datastore for clients names and to also store other relevant data.[**71**]

Angular

Angular was used for the front end/client facing web application as a very famous, well supported and easy to use web application framework.[**66**][**68**]

Ngx Admin

The Ngx Admin dashboard was used to help bootstrap the development process of our application's front end, as it already provides many pre made components and interfaces that can be easily reusable.

Postman

Postman was used to help develop the different APIs as well as to test if security/JWT authentication was working correctly or not. It is a tool that simplifies each step of building an API helping its users create better APIs—faster.

Keycloak

Keycloak plays the role of an identity and access management solution.[**68**][**69**] It helps authenticate access to both the backend and front end and only allow authorised access to the frontend as well as to sensitive backend APIs.[**70**][**74**]

Nginx

Nginx was used to serve the Angular application after it was built. It's a high quality open source product that is often compared to Apache HTTP Server. We used it because of its performance edge over the competition.

Docker

Docker and containers in general enable application portability, scalability and delivery, as well as allow for more efficient use of system resources especially in comparison to Virtual Machines. We used docker mainly for its portability features, and mainly because it allows our solution to work on as many machines as possible without needing to hunt down dependencies.[**71**]

4.2 Implementation

In this section, we'll discuss more in-depth, what and how we solved each problem and requirement that was presented to us.

4.2.1 Data Extraction and Preparation

In this portion, we had to deal with many XML files, each with its proper Schema. So we had to develop a special parser for each of them. At the beginning, we tried using Logstash, but it was found to be non-optimal for doing what we needed (especially storing an array of aliases and alternative names, not to mention the many differences and variations even among the very same XML file). So we ended up using the Dom4J parser library together with Jackson and going full Java route afterwards, and later, after we have had enough knowledge and experience with Elasticsearch, we have dropped the Jackson solution and directly saved the sanctioned entities into Elasticsearch. I have placed these figures (Fig 4.1, 4.2, 4.3, 4.4, 4.5) to show that each file has its own structure and composition and schema, and that no general solution can work that we think of, as well as to show the challenges encountered in trying to find a generic data strcuture to store all the data

within:

We can see in this XML file (See Fig 4.1) that there's a tag per sanctioned entity

```
    <sanctionEntity designationDetails="" unitedNationId="" logicalId="13">
        <subjectType code="person" classificationCode="P"/>
        <nameAlias firstName="Saddam" middleName="" lastName="Hussein Al-Tikriti" wholeName="Saddam Hussein Al-Tikriti"
regulationLanguage="en" logicalId="17">
        </nameAlias>
        <nameAlias firstName="" middleName="" lastName="" wholeName="Abu Ali" function="" title="" nameLanguage="" stror
            <regulationSummary regulationType="regulation" publicationDate="2003-07-08" numberTitle="1210/2003 (OJ L169)
2003:169:0006:0023:EN:PDF"/>
        </nameAlias>
        <nameAlias firstName="" middleName="" lastName="" wholeName="Abou Ali" function="" title="" nameLanguage="FR" st
            <regulationSummary regulationType="regulation" publicationDate="2003-07-08" numberTitle="1210/2003 (OJ L169)
2003:169:0006:0023:EN:PDF"/>
        </nameAlias>
    </sanctionEntity>
```

Figure 4.1: European Security Council Sanctions List XML

called sanctionEntity, and a name "nameAlias" tag within it holding the name and name components as attributes. (There are no parent tag to group together the sanctionEntities or the nameAliases)

Figure 4.2 and Figure 4.3 Shows a different file, one where sanctioned entities are grouped under parent tags Individuaals and Entities depending on the type of the entity (personne physique vs persone morale). Also, the entity tags depends on the parent tag (Individual/Entity)

```
<INDIVIDUALS>
<INDIVIDUAL><DATAID>2913551</DATAID><FIRST_NAME>AMIR</FIRST_NAME><SECOND_NAME>ABDULLAH</SECOND_NAME><NAME_ORIGINAL_SCRIPT>امير عبد الله</NAME_ORIGINAL_SCRIPT>
<INDIVIDUAL_ALIAS><QUALITY>Low</QUALITY><ALIAS_NAME>Amir Abdullah Sahib</ALIAS_NAME></INDIVIDUAL_ALIAS>
</INDIVIDUAL>
```

Figure 4.2: United Nations Security Council Sanctions List XML

```
<ENTITY><DATAID>113231</DATAID><FIRST_NAME>AL FURQAN</FIRST_NAME>
<ENTITY_ALIAS><QUALITY>a.k.a.</QUALITY><ALIAS_NAME>Dzemilijati Furkan</ALIAS_NAME></ENTITY_ALIAS><ENTITY_ALIAS><QUALITY>a.k.a.</QUALITY><ALIAS_NAME>Dzem'ijjetul Furqan</ALIAS_NAME></-
ENTITY_ALIAS><ENTITY_ALIAS><QUALITY>a.k.a.</QUALITY><ALIAS_NAME>Association for Citizens Rights and Resistance to Lies</ALIAS_NAME></ENTITY_ALIAS><ENTITY_ALIAS><QUALITY>a.k.a.</-
QUALITY><ALIAS_NAME>Dzemijetul Furkan</ALIAS_NAME></ENTITY_ALIAS><ENTITY_ALIAS><QUALITY>a.k.a.</QUALITY><ALIAS_NAME>Association of Citizens for the Support of Truth and Supression of Lies</-
ALIAS_NAME></ENTITY_ALIAS><ENTITY_ALIAS><QUALITY>a.k.a.</QUALITY><ALIAS_NAME>Sirat</ALIAS_NAME></ENTITY_ALIAS><ENTITY_ALIAS><QUALITY>a.k.a.</QUALITY><ALIAS_NAME>Association for Education, Culture
and Building Society-Sirat</ALIAS_NAME></ENTITY_ALIAS><ENTITY_ALIAS><QUALITY>a.k.a.</QUALITY><ALIAS_NAME>Association for Education, Cultural, and to Create Society -Sirat</ALIAS_NAME></-
ENTITY_ALIAS><ENTITY_ALIAS><QUALITY>a.k.a.</QUALITY><ALIAS_NAME>Istikamet</ALIAS_NAME></ENTITY_ALIAS><ENTITY_ALIAS><QUALITY>a.k.a.</QUALITY><ALIAS_NAME>In Siratel</ALIAS_NAME></-
ENTITY_ALIAS><ENTITY_ALIAS><QUALITY>a.k.a.</QUALITY><ALIAS_NAME>Citizens' Association for Support and Prevention of lies - Furqan</ALIAS_NAME></ENTITY_ALIAS></ENTITY>
```

Figure 4.3: United Nations Security Council Sanctions List XML 2

Figure 4.4 shows yet another file, where the aliases of individuals are stored under a parent tag called akaList... as well as how certain tags can be missing or simple have a different name depending on the value within the sdnType!

We can see how the way data is stored differs from file to file, sometimes even depending on the type of the entity (legal person/entity vs physical person/individual). That have made the usage of Logstash close to impossible due to the little control that it gives, so we ended up finally using Dom4J to parse the XML data, we also ended up using the library Jackson to convert the parsed entities into an NDJSON format that we can send to Elasticsearch for storage (See Fig 4.6).

We also wanted a way to be able to refind and reaccess the entity in the original files if we needed more data for example, therefore we store an entity_id, entity_type and id_in_document. We also value the whole_names array (that stores all aliases

```xml
<sdnEntry>
  <uid>36</uid>
  <lastName>AEROCARIBBEAN AIRLINES</lastName>
  <sdnType>Entity</sdnType>
  <akaList>
    <aka>587
      <uid>12</uid>
      <type>a.k.a.</type>
      <category>strong</category>
      <lastName>AERO-CARIBBEAN</lastName>
    </aka>
  </akaList>
  <addressList>
    <address>
      <uid>25</uid>
      <city>Havana</city>
      <country>Cuba</country>
    </address>
  </addressList>
</sdnEntry>
```


Figure 4.4: United Stated OFAC Sanctions List XML


```xml
<sdnEntry>
  <uid>2676</uid>
  <firstName>Dr. Ayman</firstName>
  <lastName>AL ZAWAHIRI</lastName>
  <sdnType>Individual</sdnType>
  <akaList>
    <aka>
      <uid>4760</uid>
      <type>a.k.a.</type>
      <category>strong</category>
      <lastName>AL-ZAWAHIRI</lastName>
      <firstName>Ayman</firstName>
    </aka>
    <aka>
      <uid>4761</uid>
      <type>a.k.a.</type>
      <category>strong</category>
      <lastName>SALIM</lastName>
      <firstName>Ahmad Fuad</firstName>
    </aka>
    <aka>
      <uid>4787</uid>
      <type>a.k.a.</type>
      <category>strong</category>
      <lastName>AL-ZAWAHIRI</lastName>
      <firstName>Aiman Muhammad Rabi</firstName>
    </aka>
  </akaList>
</sdnEntry>
```


Figure 4.5: United Stated OFAC Sanctions List XML


```
{"index":{}}
{"entity_type":"P","entity_id":"UNSecurityCouncil-6908570","source_document":"UNSecurityCouncil","id_in_document":"6908570","whole_names":["Jang Chang Ha","CHANG CH
{"index":{}}
{"entity_type":"P","entity_id":"UNSecurityCouncil-6908571","source_document":"UNSecurityCouncil","id_in_document":"6908571","whole_names":["Jo Chun Ryong ","CHO  C
{"index":{}}
{"entity_type":"P","entity_id":"UNSecurityCouncil-6908565","source_document":"UNSecurityCouncil","id_in_document":"6908565","whole_names":["JO  YONG CHOL","Cho Yong
{"index":{}}
{"entity_type":"P","entity_id":"UNSecurityCouncil-6908566","source_document":"UNSecurityCouncil","id_in_document":"6908566","whole_names":["","KIM CHOL SAM"],"prima
{"index":{}}
{"entity_type":"P","entity_id":"UNSecurityCouncil-6908568","source_document":"UNSecurityCouncil","id_in_document":"6908568","whole_names":["","KIM SOK CHOL"],"prima
{"index":{}}
{"entity_type":"P","entity_id":"UNSecurityCouncil-6908558","source_document":"UNSecurityCouncil","id_in_document":"6908558","whole_names":["","Kim Hak Song","KIM SO
{"index":{}}
{"entity_type":"P","entity_id":"UNSecurityCouncil-6908573","source_document":"UNSecurityCouncil","id_in_document":"6908573","whole_names":["","KIM  SE GON"],"primar
```


Figure 4.6: NDJSON Output to send to Elasticsearch

and names of the entity) and the primary_name of the entity (The official name, if one exists...)

Eventually though, when we started developing the backend, we dropped the NDJSON solution, and used Spring Data to store the data directly to Elasticsearch. To do that,

we have developed the following java class with the following attributes, which holds all the data that is important to us when performing a search operation (See Fig 4.7):

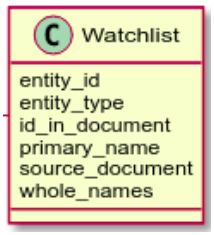

Figure 4.7: Watchlist Agreed Upon Format containing all relevant data to us

4.2.2 Elasticsearch Mapping, Analyzers and the Search query

Please Read Appendix A to learn more about about Elasticsearch, we assume Knowledge for how Elasticsearch uses analysers on mappings when storing and searching for data (a mapping is basically the schema/structure of similar looking documents). We also assume knowledge about Levenshtein Distance as well as some basic knowledge on phonetic algorithms, mainly Double Metaphone and Beider Morse. It will also be helpful if you have knowledge on shingles and Ngrams

The role analyzers play in Elasticsearch is important. But they must be defined prior to indexing. That's why we need to give careful thought to our analyzers so as to avoid having to reindex data. We have built on the previous watchlist class and developed a relevant mapping (See Fig 4.8) that takes advantage of Elasticsearch to store the parsed sanctioned entities into Elasticsearch[**23**][**25**][**26**][**39**] (it started out simple, but got better with the usage of multi_fields and analyzers, the next section will discuss that in more details:

```
{
  "properties": {
    "entity_id": {
      "type": "keyword"
    },
    "source_document": {
      "type": "keyword"
    },
    "id_in_document": {
      "type": "keyword"
    },
    "entity_type": {
      "type": "keyword"
    },
    "primary_name": {"type": "keyword"...},
    "whole_names": {"type": "keyword"...}
  }
}
```

Figure 4.8: Watchlist Mapping within Elasticsearch

Both primary_name and whole_names use the same kind of mapping, so we'll only use one of them for explanations (See Fig 4.9):

We mainly have two analyzer for searching names in a fuzzy way, one in its native script (the clean and clean_with_shingles analyzers). Four analyzers for dealing with names written in a non latin script (which are the cross_lingual analyzers) that also use phonetic analyzers to help bridge the gap after transliterating names. We also have

```
"whole_names": {
  "type": "keyword",
  "fields": {
    "clean": {
      "type": "text",
      "analyzer": "fuzzy_matching"
    },
    "clean_with_shingles": {"type": "text"...},
    "cross_lingual_double_metaphone": {
      "type": "text",
      "analyzer": "transliterate_metaphone"
    },
    "cross_lingual_double_metaphone_shingles": {"type": "text"...},
    "cross_lingual_beier_morse": {
      "type": "text",
      "analyzer": "transliterate_bider_morse"
    },
    "cross_lingual_beier_morse_with_shingles": {"type": "text"...},
    "phonetic_dm": {
      "type": "text",
      "analyzer": "double_metaphone"
    },
    "phonetic_dm_with_shingles": {"type": "text"...},
    "phonetic_bm": {
      "type": "text",
      "analyzer": "bider_morse"
    },
    "phonetic_bm_with_shingles": {"type": "text"...},
    "truncated": {
      "type": "text",
      "analyzer": "ngram_truncate"
    }
  }
}
```

Figure 4.9: Watchlist Mapping multi-fields within Elasticsearch

4 phonetic analyzers (which are phonetic_analyzers) that help with finding names that have the same pronunciation, and the last one is the truncated Analyzer that helps raise the relevance of names where substrings match. Each Analyzer (except for the truncated analyzer) has two version, a normal version, and a with shingles version, and that is to help with names written with missing spaces/concatenated together, as well as to account for weird punctuation... That means, I won't be explaining the shingles analyzers since understanding the normal version is enough

4.2.2.1 The Analyzers

Analyzers are designed specifically with the purpose of solving many of the name matching phenomena that we encounter everywhere. [**16**][**17**][**32**][**43**][**44**] The fuzzy_matching and fuzzy_matching_with_shingles analyzers (See Figure 4.10) help with performing, as the name suggests, fuzzy matching. In short, it normalizes and tokenizes the input in a standard manner after which, we can perform a fuzzy query on both the indexed document and the search term.[**36**][**45**]

The next 4 analyzers (See Figure 4.11) that we'll explain are the transliterate analyzers

```
"fuzzy_matching": {
  "type": "custom",
  "tokenizer": "nfd_normalized",
  "filter": [
    "lowercase",
    "icu_normalizer",
    "icu_folding",
    "unique"
  ]
},
"fuzzy_matching_with_shingles": {"type": "custom"...},
```

Figure 4.10: Elasticsearch Fuzzy Matching Analyzer

(transliterate_metaphone, transliterate_metaphone_with_shingles, transliterate_bider_morse, and the transliterate_bider_morse_with_shingles), these analyzers have the role of transliterating names from non-latin scripts to latin, and then apply a phonetic encoder to turn the problem into a kind of phonetic search [**47**]

```
"transliterate_metaphone": {
  "type": "custom",
  "tokenizer": "nfd_normalized",
  "filter": [
    "lowercase",
    "icu_transliterate",
    "phonetic_double_metaphone",
    "unique"
  ]
},
"transliterate_metaphone_with_shingles": {"type": "custom"...},
"transliterate_bider_morse": {
  "type": "custom",
  "tokenizer": "nfd_normalized",
  "filter": [
    "icu_transliterate",
    "lowercase",
    "phonetic_double_metaphone",
    "unique"
  ]
},
"transliterate_bider_morse_with_shingles": {"type": "custom"...},
```

Figure 4.11: Elasticsearch Transliterate Analyzers

Other than these, we also have another 4 analyzers (See Figure 4.12), which are responsible for phonetic matching of names written in the latin script! We used mainly the double metaphone and the beider morse phonetic analyzers because we assume these are the best for our use case.

Finally, we have an Ngram analyzer (see Figure 4.13) that will help us raise the relevance of names that share many of the same subparts. Thus helps with fuzzy searching a bit, as well as when truncated names are searched for

```
"double_metaphone": {
  "type": "custom",
  "tokenizer": "nfd_normalized",
  "filter": [
    "lowercase",
    "my_shingle",
    "phonetic_double_metaphone",
    "unique"
  ]
},
"double_metaphone_with_shingles": {"type": "custom"...},
"bider_morse": {
  "type": "custom",
  "tokenizer": "nfd_normalized",
  "filter": [
    "lowercase",
    "phonetic_double_metaphone",
    "unique"
  ]
},
"bider_morse_with_shingles": {"type": "custom"...},
```

Figure 4.12: Elasticsearch Phonetic Analyzers

```
"ngram_truncate": {
  "type": "custom",
  "char_filter": [
    "nfd_normalizer"
  ],
  "tokenizer": "ngram_tokenizer",
  "filter": [
    "lowercase"
  ]
}
```

Figure 4.13: Elasticsearch Ngram Analyzers

These analyzers help provide all the needed functionality to our application, as you can probably see, they do cover most of the possible name phenomena mentioned in the beginning of the report. The next figure (See Figure 4.14) Shows how all these Analyzers come into play with our search query and the different score boosts given to each.

```
{
 "bool" : {
  "should" : [
    { "match": {"whole_names.clean": {"query": "Khalil Sleimi", "fuzziness": "AUTO", "boost": 7}} },
    { "match": {"whole_names.clean_with_shingles": {"query": "Khalil Sleimi", "fuzziness": "AUTO", "boost": 4}} },
    { "multi_match": {
      "query": "Khalil Sleimi",
      "fields": ["whole_names.cross_lingual_double_metaphone^17", "whole_names.cross_lingual_double_metaphone_with_shingles^7",
        "whole_names.cross_lingual_beier_morse^7", "whole_names.cross_lingual_beier_morse_with_shingles^3",
        "whole_names.phonetic_dm^7", "whole_names.phonetic_dm_with_shingles^4", "whole_names.phonetic_bm^4",
        "whole_names.phonetic_bm_with_shingles^2"] } } ,
    { "match": { "whole_names.truncated": "Khalil Sleimi" }}      ]
  }
}
```

Figure 4.14: Elasticsearch Search Query using all the defined Analyzers

4.2.3 The Front-end

In this subsection, we'll talk about the front-end, and since we use Keycloak to help secure our application, then I'll start by that. That is, I'll show the entire experience of using our application

First of all, you need to open up Keycloak Admin Console on port 8080 and use a predefined admin password. (admin:admin) (See Figure 4.15)

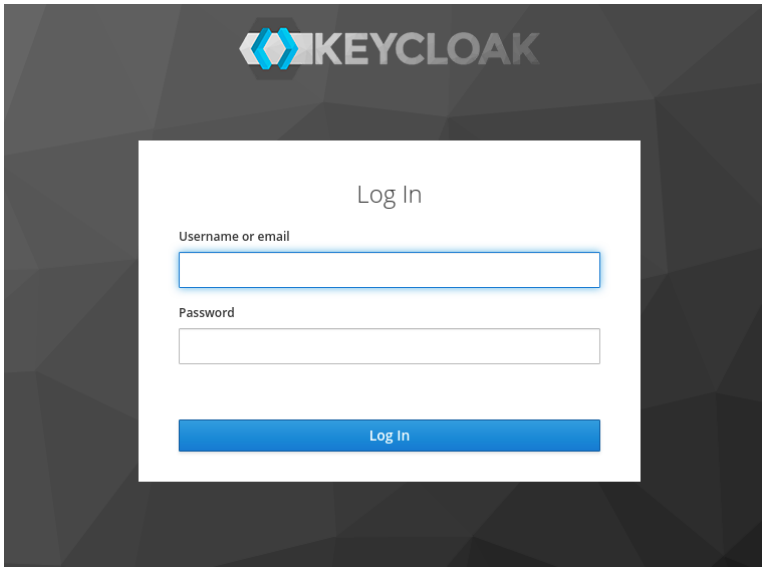

Figure 4.15: Keycloak Admin Console Login

After that, you need to create a new user account to be used to log into the application: (See Figure 4.16)

Also, you need to assign it the role that allows you to search and view data: VISITOR (See Figure 4.17)

You also need to set a password from within the application: (See Figure 4.18)

Keycloak also gives you the ability to manage users accounts, change their password, change their roles...: (See Figure 4.19)

It also gives you the ability to do many other things, but we only needed user management, authorization and authentication, as well as a realm for the frontend

When you try to acccess the angular front-end that allows you to perform searches (it's served using Nginx, so it's using port 80), the page gets redirected automatically to keycloak which asks you to enter your credentials. (See Figure 4.20)

And when the credentials are valid, as well as when the entered user has the desired role assigned to him, you get redirected to the search page: (See Figure 4.21)

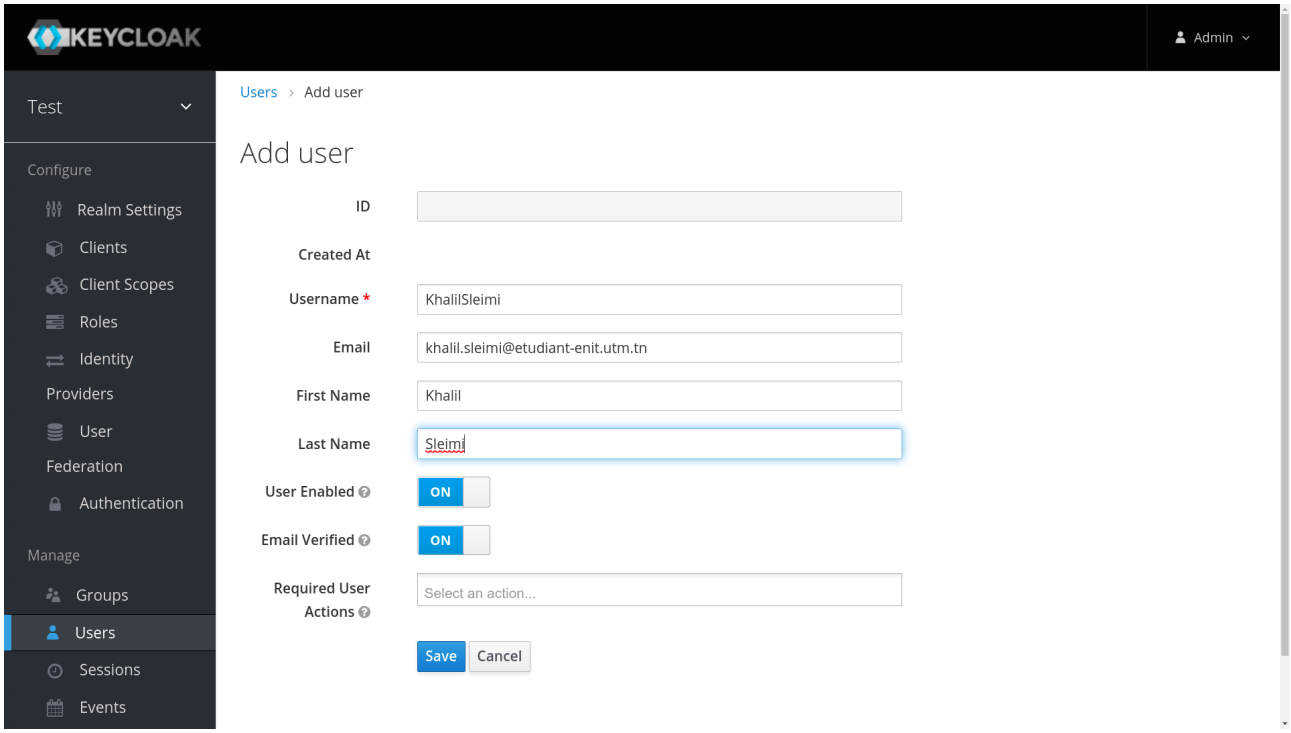

Figure 4.16: Keycloak Admin Console Add User

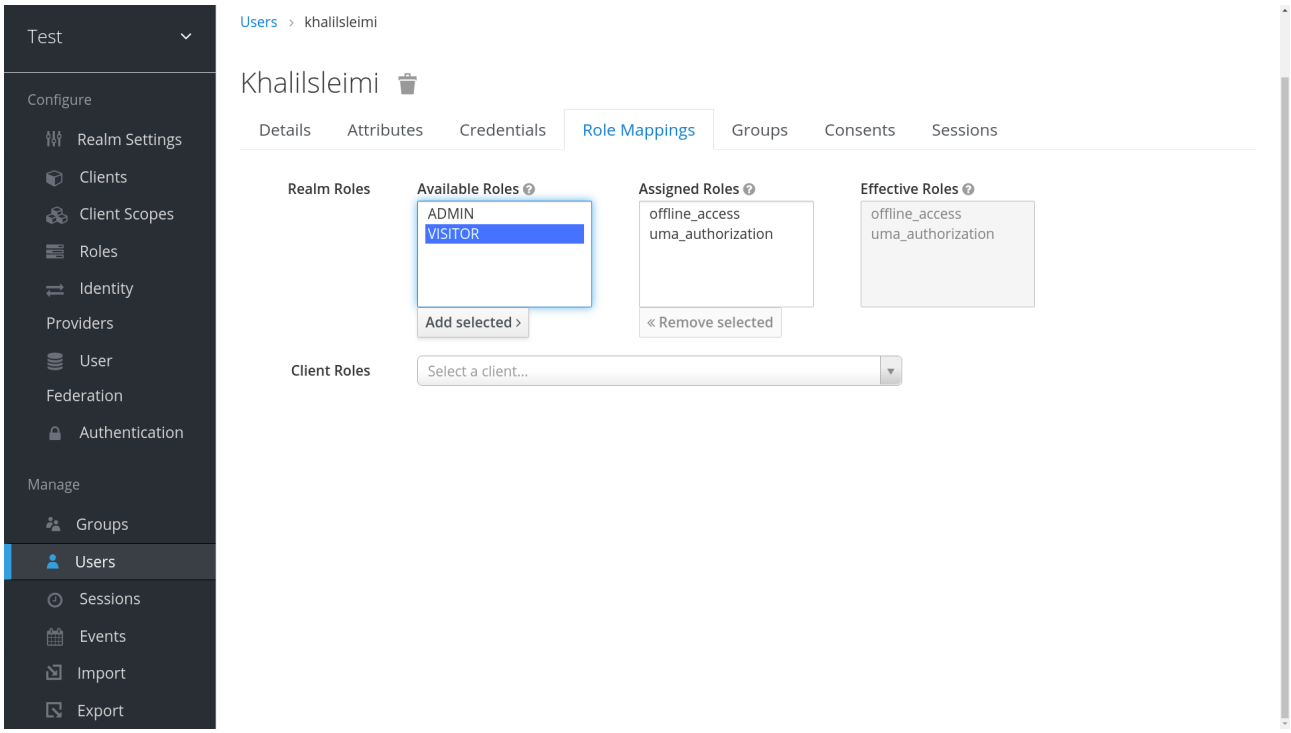

Figure 4.17: Keycloak Admin Console Assign role to user

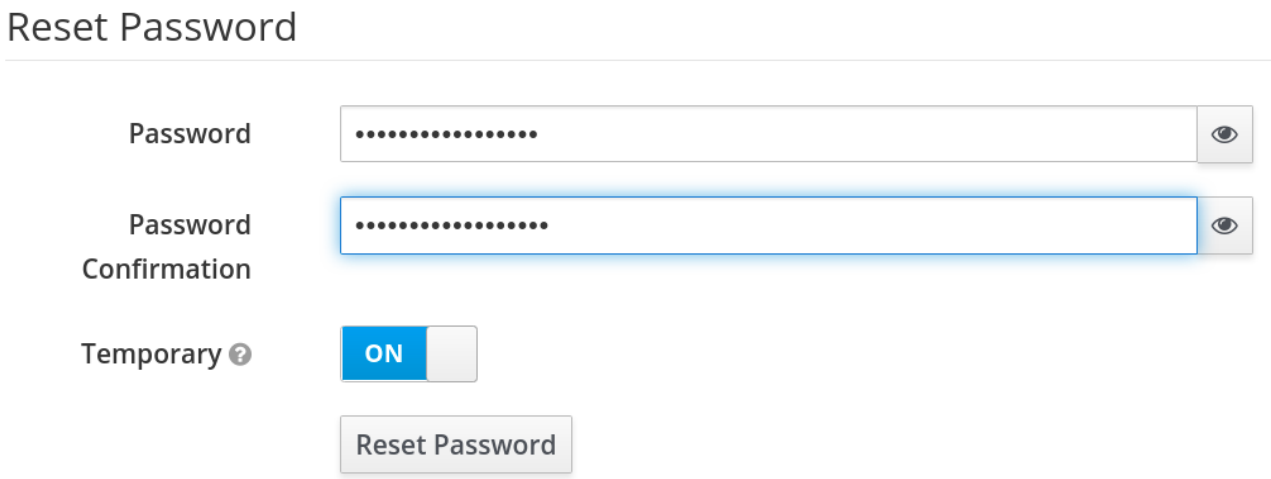

Figure 4.18: Keycloak Admin Console Assign set password to newly created user

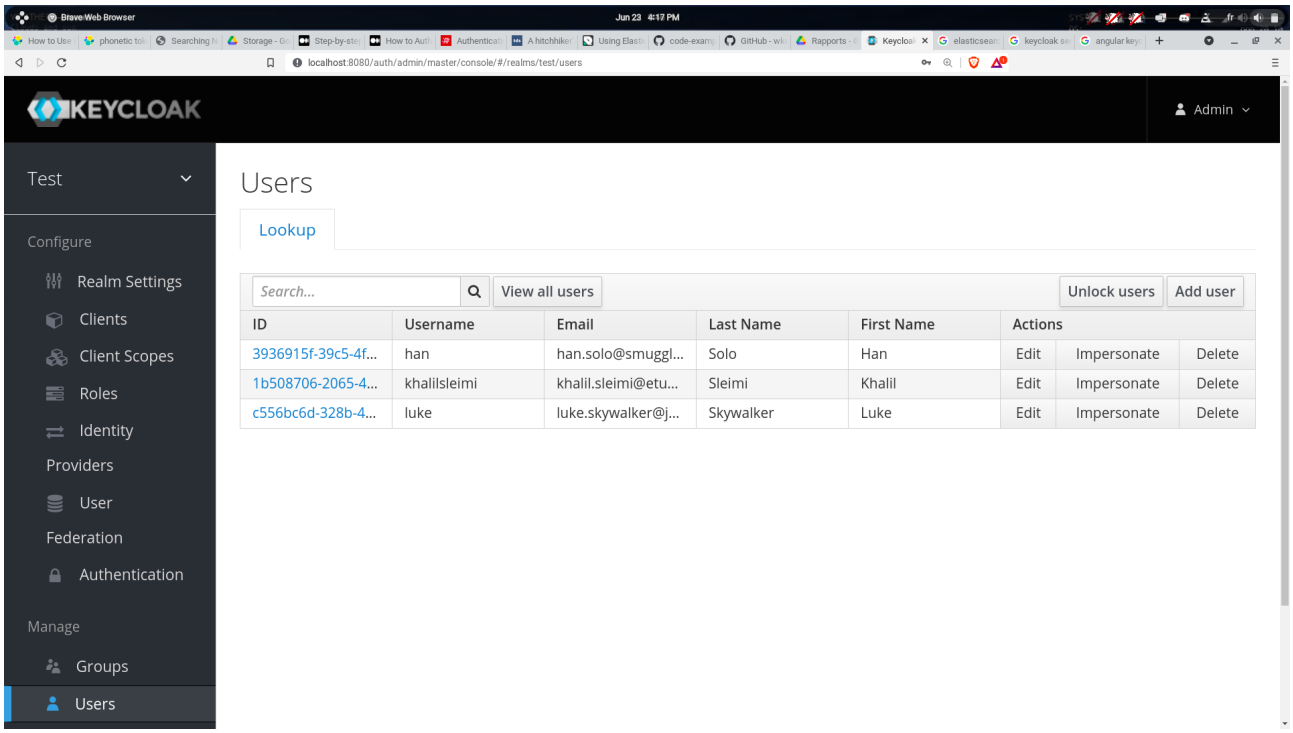

Figure 4.19: Keycloak Admin Console manage existing users

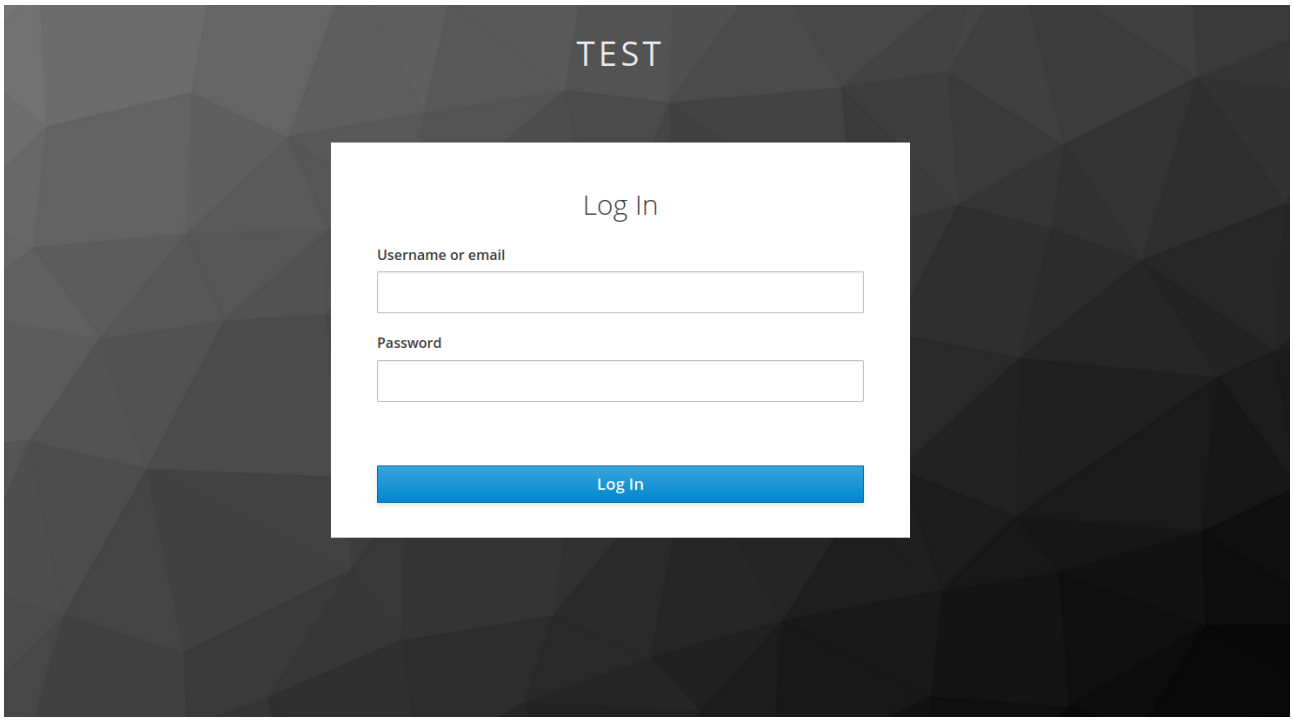

Figure 4.20: Application Login

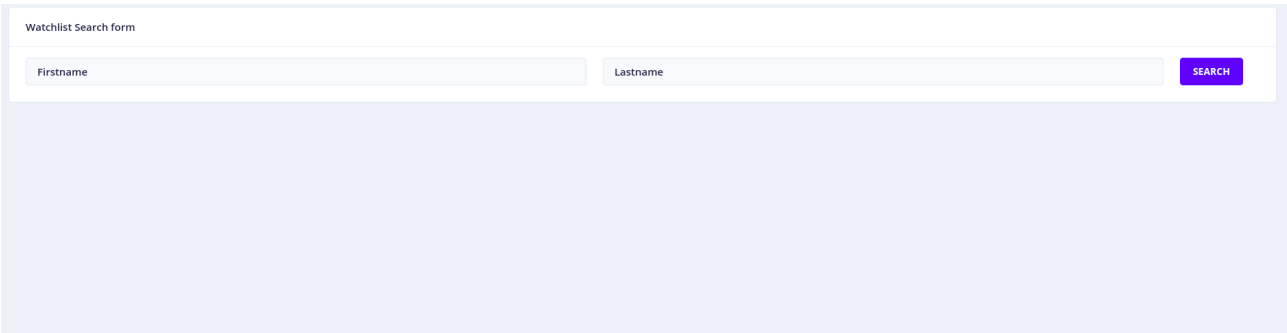

Figure 4.21: Search Engine Interface

Now that we're logged into the frontend, it's time to test how our solutions solves the name phenomena mentioned in Figure 1.1. We'll start first of all by entering a name in Russian and show you the top two returned results (See Figure 4.22: neither of the results having a name variant in Russian as is clear from the Whole Names list, and still, the first two results are both correct matches!)

Watchlist Search form

Амир Абд аль-Рахман | аль-Маула аль-Салби | SEARCH

Search Results for Амир Абд аль-Рахман аль-Маула аль-Салби | 1452 Matches Found

Actions	ID	Source	ID In Document	Type	Primary Name	Whole Names	Score
+	ID	Source	ID In Documer	Type	Primary Name	Whole Names	Score
	EUSecurityCouncil-123766	EUSecurityCouncil	123766	P	Ustadh Ahmad	Amir Muhammad Sa'id 'Abd-al-Rahman Muhammad al-Mula,Amir Muhammad Sa'id Abdal-Rahman al-Mawla,Hajji Abdallah, `Abdul Amir Muhammad Sa'id Salbi,Abu 'Umar al-Turkmani,أمير محمد سعيد عبد الرحمن المولى,Ustadh Ahmad,al-Hajj Abdullah Qardash,Hajji Abdullah Al-Afari,Abdullah Qardash,Muhammad Sa'id `Abd-al-Rahman al-Mawla,Abu 'Abdullah Qardash,Abu Ibrahim al-Hashimi al-Qurashi,Al-Ustadh	325.1466
	UNSecurityCouncil-6908841	UNSecurityCouncil	6908841	P	Amir Muhammad Sa'id Abdal-Rahman al-Mawla	,Hajji Abdallah,Amir Muhammad Sa'id 'Abd-al-Rahman Muhammad al-Mula, `Abdul Amir Muhammad Sa'id Salbi,أمير محمد سعيد عبد الرحمن المولى,Abu 'Umar al-Turkmani,al-Hajj Abdullah Qardash,Ustadh Ahmad,Hajji Abdullah Al-Afari,Abdullah Qardash,Muhammad Sa'id `Abd-al-Rahman al-Mawla,Amir Muhammad Sa'id Abdal-Rahman al-Mawla,Abu Ibrahim al-Hashimi al-Qurashi,Abu 'Abdullah Qardash,Al-Ustadh	325.1466

Figure 4.22: Search Engine Same Name in Multiple Languages Russian/English

Next, We'll be searching for a name with a missing name component, and we'll do it with a famous name:

Watchlist Search form

oussema | laden | SEARCH

Search Results for oussema laden | 2561 Matches Found

Actions	ID	Source	ID In Document	Type	Primary Name	Whole Names	Score
+	ID	Source	ID In Docum	Type	Primary Name	Whole Names	Score
	EUSecurityCouncil-833	EUSecurityCouncil	833	P	Bin Laden Usamah Bin Muhammad	Ben Laden Osama,Osama bin Ladin,Usama bin Muhammad bin Awad bin Ladin,Shaykh Usama Bin Ladin,Abu Abdallah Abd Al-Hakim,Ben Laden Usama,Usamah Bin Muhammad Bin Ladin,Osama bin Muhammad bin Awad bin Ladin,Ben Laden Ossama,Usama bin Laden,Usama Bin Muhammed Bin Awad, Osama Bin Laden,Bin Laden Usamah Bin Muhammad,Al Qaqa,Bin Laden Osama Mohamed Awdh,Usama Muhammed Awad Bin Laden,Usama bin Ladin	567.3309
	EUSecurityCouncil-119516	EUSecurityCouncil	119516	P	Hamza Usama Muhammad bin Laden	Hamza Usama Muhammad bin Laden	385.5744
	UNSecurityCouncil-6908767	UNSecurityCouncil	6908767	P	HAMZA USAMA MUHAMMAD BIN LADEN	,حمزة أسامة محمد بن لادن,HAMZA USAMA MUHAMMAD BIN LADEN	367.39508
						la base,Ιολαμικό Ίδρυμα Σωτηρίας,Група за опазване на светите места,Ал Кайда,baza,Fundação da Salvação Islâmica,ΑΛ Κάιντα/Ιολαμικός στρατός,Rede de Osama bin Laden,Organização de Osama bin Laden,The Group for the	

Figure 4.23: Search Engine Missing Name Component

As you can see (Figure 4.23), we still get the relevant name we were looking for even if we forget to write a name component. This is due to how elasticsearch works, it doesn't need every single name component to be present (because it counts a score, meaning that the relevance scores of the correct name will be lower but will still match)

As for out-of-order name components, it doesn't matter how you order the names components, it'll always be the same due to the way elasticsearch tokenizes the input (This doesn't need a demonstration as it's self evident, though the score will definitely be lower since the tokens positions don't match).

Now, we only need to prove that problems of phonetic search, missing spaces or hyphens and truncated name components are still solvable using our solution.

Let's say we want to match "Houssem Ben Mohamed Naceur Ben Mohamed TRABELSI" (since it only has only one name variant present, we will know for sure if it works well or not). Let's search for that name by deleting first all spaces (an edge case):

Watchlist Search form

HoussemBenMohamedNaceurBenMohamedTRABELSI

SEARCH

Search Results for HoussemBenMohamedNaceurBenMohamedTRABELSI | 2975 Matches Found

Actions	ID	Source	ID In Document	Type	Primary Name	Whole Names	Score
+	ID	Source	ID In Document	Type	Primary Name	Whole Names	Score
	EUSecurityCouncil-6081	EUSecurityCouncil	6081	P	Houssem Ben Mohamed Naceur Ben Mohamed TRABELSI	,Houssem Ben Mohamed Naceur Ben Mohamed TRABELSI	287.34256
	EUSecurityCouncil-6058	EUSecurityCouncil	6058	P	Mohamed Naceur Ben Mohamed Ben Rhouma TRABELSI	,Mohamed Naceur Ben Mohamed Ben Rhouma TRABELSI	194.49292
	EUSecurityCouncil-6061	EUSecurityCouncil	6061	P	Mohamed Imed Ben Mohamed Naceur Ben Mohamed TRABELSI	,Mohamed Imed Ben Mohamed Naceur Ben Mohamed TRABELSI	186.65228
	EUSecurityCouncil-6066	EUSecurityCouncil	6066	P	Mohamed Mourad Ben Mohamed Ben Rehouma TRABELSI	,Mohamed Mourad Ben Mohamed Ben Rehouma TRABELSI	151.47696

Figure 4.24: Search Engine Missing Spaces or Hyphens

The correct name is displayed first in the list! (See Figure 4.24). The reason for such matches include the shingles as well as the Ngram analyzers. As for hyphens instead of spaces, it's easy to know why it works perfectly, it's because the tokenizers we use will automatically tokenize input whenever it encounters certain characters, among which is the hyphen, another self-evident situation.

Another edge case worth testing is truncating name components, for that, we'll search for "Hous b Moha Naceu b Moha TRABEL" expecting to match the full name "Houssem Ben Mohamed Naceur Ben Mohamed TRABELSI" (with every single name component having some part of it truncated).

Search Results for Hous b Moha Naceu b Moha TRABEL | 2780 Matches Found

Actions	ID	Source	ID In Document	Type	Primary Name	Whole Names	Score
+	ID	Source	ID In Document	Type	Primary Name	Whole Names	Score
	EUSecurityCouncil-6081	EUSecurityCouncil	6081	P	Houssem Ben Mohamed Naceur Ben Mohamed TRABELSI	,Houssem Ben Mohamed Naceur Ben Mohamed TRABELSI	354.51837
	EUSecurityCouncil-6058	EUSecurityCouncil	6058	P	Mohamed Naceur Ben Mohamed Ben Rhouma TRABELSI	,Mohamed Naceur Ben Mohamed Ben Rhouma TRABELSI	349.06042
	EUSecurityCouncil-6061	EUSecurityCouncil	6061	P	Mohamed Imed Ben Mohamed Naceur Ben Mohamed TRABELSI	,Mohamed Imed Ben Mohamed Naceur Ben Mohamed TRABELSI	338.93018
	EUSecurityCouncil-6050	EUSecurityCouncil	6050	P	Mohamed Ben Moncef Ben Mohamed TRABELSI	,Mohamed Ben Moncef Ben Mohamed TRABELSI	257.43597

Figure 4.25: Search Engine Truncated Name Components

As you can see in figure 4.25, even this edge case is taken care of and returns the correct result.

Finally, let's talk about phonetic similarity, which is harder to test since it takes time to come up with alternative name spellings, so to test phonetic similarity, let's try writing the name as "husam bin muhemd nesir bin muhemid treblsi" which is clearly the same name from a human's perspective and see the results:

Search Results for husam bin muhemd nesir bin muhemid treblsi | 2817 Matches Found

Actions	ID	Source	ID In Document	Type	Primary Name	Whole Names	Score
+	ID	Source	ID In Docume	Type	Primary Name	Whole Names	Score
	EUSecurityCouncil-6081	EUSecurityCouncil	6081	P	Houssem Ben Mohamed Naceur Ben Mohamed TRABELSI	,Houssem Ben Mohamed Naceur Ben Mohamed TRABELSI	580.53424
	EUSecurityCouncil-6058	EUSecurityCouncil	6058	P	Mohamed Naceur Ben Mohamed Ben Rhouma TRABELSI	,Mohamed Naceur Ben Mohamed Ben Rhouma TRABELSI	449.07706
	EUSecurityCouncil-6061	EUSecurityCouncil	6061	P	Mohamed Imed Ben Mohamed Naceur Ben Mohamed TRABELSI	,Mohamed Imed Ben Mohamed Naceur Ben Mohamed TRABELSI	448.54776

Figure 4.26: Search Engine Phonetically Similar Name Components

As you can see (In Fugure 4.26), the correct result is still the first one followed by very similar names.

This is wonderful! We can search across languages and scripts and even make errors when writing the name, better yet, write names however they sound like phonetically and without necessarily writing it using correct spellings, and still, get the desired results!

Another name phenomena is spelling mistakes, but these are taken into consideration through the use of the fuzzy parameter (remember the clean and clean with shingles analyzers, remember also the search query), and I don't see it deserves its own explanation since it is too self evident...

After these search demonstrations (as well as the details and explanations given in the previous section and in Appendix A about the analyzers, character filters, tokenizers and token filters, we can clearly see that we have addressed every single required name matching phenomena that we were required to solve!

4.2.4 Customers due diligence

Another part of our project was focused on developing customers due diligence, where we take the demo list of Vneuron customers and check them against the blacklists, and then store the search results within the postgres database. (See Figure 4.27 for a list of very similar sometimes even true positives, that is, clients in the test database that really exist in the wathclists) It is also important to note that our solution is capable of performing all of this (Customer Due Diligence) in about 117 seconds to be exact total, there's a total of about 11177 customers and a total of 12022 sanctioned entities from the watchlists. That means, our solution is capable of checking around 100 customers against the full

	watchlist_id	client_name	score 1	watchlist_name
15	EUSecurityCouncil-7217	Hong Dong International Hong Dong International N	876.8582	Leader (Hong Kong) International Trading Limited
16	UNSecurityCouncil-690768	Hong Dong International Hong Dong International N	876.8582	<null>
17	EUSecurityCouncil-7220	ETS MOHAMED OULD MOHAMED LEMINE ETS MOHAMED OULD MOHAMED …	875.427	Hamad el Khairy
18	UNSecurityCouncil-3013119	MOHAMED EL BECHIR MOHAMED EL BECHIR ETS OULD MOHAMED LEMI…	874.9026	HAMADA OULD MOHAMED EL KHAIRY
19	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
20	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
21	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
22	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
23	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
24	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
25	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
26	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
27	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
28	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
29	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
30	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
31	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
32	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
33	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
34	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
35	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
36	EUSecurityCouncil-7565	ANATOLIY IVANOVYCH ANTONOV	867.39105	Anatoly Ivanovich ANTONOV
37	EUSecurityCouncil-5676	CONTROLE CONSELLS INGENIERIE-SARL CONTROLE CONSELLS INGEN…	866.0251	Sahel Consultant Engineers
38	EUSecurityCouncil-7133	EL KHERCHY MOHAMED LEMINE MOHAMED YAHYA	857.9702	Mohammed Yahya MALA
39	UNSecurityCouncil-6908802	GENERAL FORUM OF THE ARABIC AND GENERAL FORUM OF THE ARAB…	855.1591	<null>
40	EUSecurityCouncil-122209	GENERAL FORUM OF THE ARABIC AND GENERAL FORUM OF THE ARAB…	855.1591	ISLAMIC STATE WEST AFRICA PROVINCE (ISWAP)
41	EUSecurityCouncil-6478	Mohamed Mahmoud Jaafar LALLA MOHAMED MAHMOUD JAAFAR N	847.61835	Mohamed Ali JAFARI
42	EUSecurityCouncil-7133	Mohamed Yahya Mohamed Yahya AHMED OUMEIRY MOHAMED YAHYA	846.39984	Mohammed Yahya MALA
43	EUSecurityCouncil-6903	British Arab Commercial Bank British Arab Commercial Bank…	843.9477	Banca Comercială Siriano- Libaneză
44	EUSecurityCouncil-6875	TEGUEDI MOHAMED MAAROUF TEGUEDI N	840.72955	Mohamed MAAROUF

Figure 4.27: Search Engine Customer Due Diligence Search Results with High Probability

12022 sanctioned entities per second! This performance and speed is really amazing and it is all thanks to Elasticsearch, especially considering the number of analyzers we have.

4.2.5 Chronogram of Performed Activities

The following figure enlists chronologically the activities performed during the internship. Here, we enumerate the major parts of the application that we focused most on per monthly basis. The development of Security features as well as the containerisation of

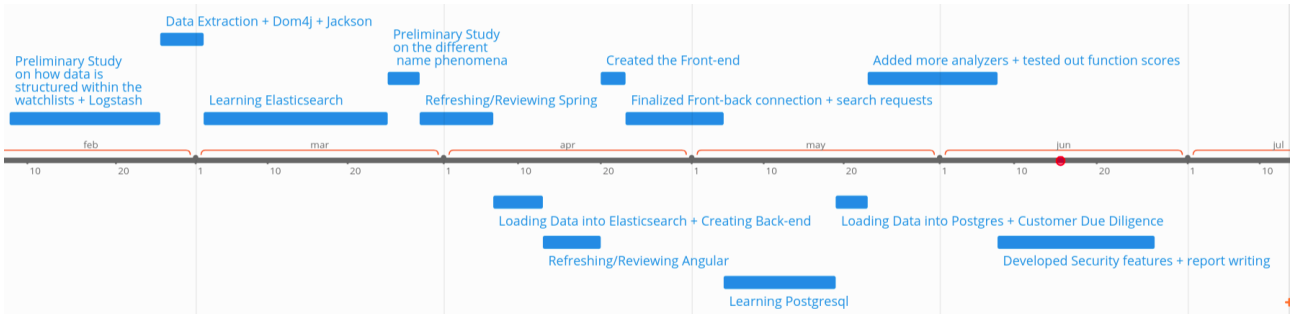

Figure 4.28: Internship Chronogram

our application was all done after the internship's end, as well as the addition of around 5 extra analyzers for a total of 11.

Conclusion

In this chapter, we outlined the work environment : both the hardware and software components were presented. Furthermore, a detailed explanation of the implementation was introduced.

Conclusion

In this report, we have fully presented our work, which is part of our graduation project. The main objective of this project is to design and build a name indexing and matching solution for internal use by our host company Vneuron as well as a wrapping Back-end and Front-end to simplify and help interaction with the system.

We began this report with a study of the general context of the project, which allowed us to identify the problem that gave rise to the need for such a system. Then, we specified the functional and non-functional needs that our application will have to meet. Then, we proposed our solution and its design. We described the static, dynamic and deployment views of our system thanks to the C4 architectural design approach.

Once the design was completed, we showed the different technologies used in the realization of this project, we went on to the implementation and we demonstrated all of this with screenshots the interfaces of our solution, the different Elasticsearch Analyzers and the mapping used, and even added an appendix for those who want to read and get more details about Elasticsearch.

This project was an enriching and fruitful experience which allowed us to acquire new skills of great value in web development and in particular on the Spring and Angular Framework, on the Elastic stack, on Containerisation solutions such as Docker and on Security and authentication with Spring Security, AuthGuard and Keycloak . On the other hand, we were able to improve our knowledge acquired throughout our academic career and learn more about agile methodologies and in particular the Scrum methodology. Thus, this internship, which represents a new professional experience, was beneficial to us; it was a new chance to consolidate our technical skills and to put our theoretical knowledge into practice. We were able to find solutions and overcome the encountered difficulties.

We learned how to be autonomous, open to others, and more aware of professional life which requires punctuality, seriousness and hard work. We took responsibility and we developed the application by meeting the defined needs while respecting the specifications, requirements and deadlines.
Like any IT project, constraints were overcome throughout the project. It have been a challenge to acquire new technical knowledge and to develop operational capacities. These challenges concern in particular the familiarization with the new Frameworks and the mastery of certain technical tools such as the Elastic stack, Spring Framework, Angular Framework, Postgresql...

To conclude, our application can be easily improved and enhanced upon to become a much better solution, our work so far has focused mainly on recall, and we're very proud with the outcomes, next steps could be to work on the Precision and scoring side of the equation.[**62**] We found out that there's a plugin, called Learning to Rank, that allows us to upload machine learning models that are trained on matching names which we can definitely learn about and even develop a solution given enough time. We also know what we would do differently if given a second chance and more experience and domain knowledge: We'll write our own Elasticsearch plugin by taking advantage of the open-source nature of the plugins we used and needed, allowing us greater flexibility, control and abstraction over custom mappings with custom data types and custom analyzers we would also use linguistic knowledge and expertise in a better solutions.

Indeed, we are planning to work more and more on the application and hopefully turn it into a more professional solution.

Webography

[1] C4 Model: An "abstraction-first" approach to diagramming software architecture, https://c4model.com/ - 8 February 2021

[2] The Agile Coach - Atlassian's no-nonsense guide to agile development https://www.atlassian.com/agile - 8 February 2021

[3] Agile 101 https://www.agilealliance.org/agile101/ - 8 February 2021

[4] Manifesto for Agile Software Development https://agilemanifesto.org/ - 8 February 2021

[5] Scrum - Learn how to scrum with the best of 'em https://www.atlassian.com/agile/scrum - 8 February 2021

[6] What is Scrum? - A Better Way Of Building Products https://www.scrum.org/resources/what-is-scrum - 8 February 2021

[7] The 2020 Scrum Guide https://scrumguides.org/scrum-guide.html - 8 February 2021

[8] European Union Sanctions List: https://webgate.ec.europa.eu/europeaid/fsd/fsf/public/files/xmlFullSanctionsList/content?token=n002he02 - 9 February 2021

[9] United Nations Sanctions List: https://scsanctions.un.org/resources/xml/en/consolidated.xml - 9 February 2021

[10] U.S. Department of Treasury Specially Designated Nationals list: https://www.treasury.gov/ofac/downloads/sdn.xml - 9 February 2021

[11] Complete Guide to Elasticsearch: https://www.udemy.com/course/elasticsearch-complete-guide/ - 10 February 2021

[12] Data Processing with Logstash (and Filebeat) : https://www.udemy.com/course/processing-events-with-logstash/ - 10 February 2021

[13] An Overview of Fuzzy Name Matching Techniques: https://www.rosette.com/blog/overview-fuzzy-name-matching-techniques/ - 18 February 2021

[14] AI Code Completions for IntelliJ: https://www.codota.com/ - 18 February 2021

[15] Vneuron's name matching technology: https://www.vneuron.com/compliance/technology/#name-screening - 19 February 2021

[16] A Guide to Effective Customer Screening: https://www.oracle.com/us/industries/financial-services/ofs-customer-screening-guide-3590304.pdf - 19 February 2021

[17] Watchlist Screening : https://risk.lexisnexis.com/financial-services/financial-crime-compliance/watchlist-screening - 19 February 2021

[18] How does the xml plugin for logstash work?: (asked by me) https://www.reddit.com/r/elasticsearch/comments/lnivdi/how_does_the_xml_plugin_for_logstash_work/ - 21 February 2021

[19] B-Tree Indexes: https://www.youtube.com/watch?v=NI9wYuVIYcA - 4 Mars 2021

[20] KD tree algorithm: how it works: https://www.youtube.com/watch?v=TLxWtXEbtFE - 4 Mars 2021

[21] KD-Tree Nearest Neighbor Data Structure: https://www.youtube.com/watch?v=Glp7THUpGow - 4 Mars 2021

[22] B-Tree Tutorial - An Introduction to B-Trees: http://www.epfl.ch - 4 Mars 2021

[23] Field data types: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-types.html - 4 Mars 2021

[24] Elasticsearch Cheatsheet : Example API usage of using Elasticsearch with curl: https://gist.github.com/ruanbekker/e8a09604b14f37e8d2f743a87b930f93 - 8 Mars 2021

[25] Fields: https://www.elastic.co/guide/en/elasticsearch/reference/7.11/multi-fields.html - 10 Mars 2021

[26] Index templates: https://www.elastic.co/guide/en/elasticsearch/reference/current/index-templates.html - 11 Mars 2021

[27] Phonetic Analysis Plugin: https://www.elastic.co/guide/en/elasticsearch/plugins/current/analysis-phonetic.html - 11 Mars 2021

[28] Built-in analyzer reference: https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html - 11 Mars 2021

[29] ICU Analysis Plugin: https://www.elastic.co/guide/en/elasticsearch/plugins/7.11/analysis-icu.html - 11 Mars 2021

[30] Java High Level REST Client: https://www.elastic.co/guide/en/elasticsearch/client/java-rest/7.13/java-rest-high.html - 11 Mars 2021

[31] Phonetics in Elastic(Search): https://medium.com/@purbon/phonetics-in-elastic-search-3a46da4bb7dc - 11 Mars 2021

[32] Human names searching - how to improve results: https://discuss.elastic.co/t/human-names-searching-how-to-improve-results/233902 - 11 Mars 2021

[33] Elasticsearch Archives - Coding Explained: https://codingexplained.com/coding/elasticsearch - 11 Mars 2021

[34] Elasticsearch 7.x Cheatsheet: https://elasticsearch-cheatsheet.jolicode.com/ - 11 Mars 2021

[35] Improving an elasticsearch query on human names and addresses: https://stackoverflow.com/questions/62075428/improving-an-elasticsearch-query-on-human-names-and-addresses - 11 Mars 2021

[36] An Elegant and Efficient Way to Fuzzy Search Names in Elasticsearch: https://www.rosette.com/blog/fuzzy-search-names-in-elasticsearch/ - 11 Mars 2021

[37] ElasticSearch Phonetic Algorithms: https://www.linkedin.com/pulse/elasticsearch-phonetic-algorithms-daniel-ranallo/?trk=related_artice_ElasticSearch%20Phonetic%20Algorithms_article-card_title - 11 Mars 2021

[38] ElasticSearch API cheatsheet: https://blog.frankel.ch/elasticsearch-api-cheatsheet/ - 11 Mars 2021

[39] Fuzzy query: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-fuzzy-query.html - 11 Mars 2021

[40] Preparing for the Elastic Certified Engineer Exam - Get Elasticsearch Certified: https://www.youtube.com/watch?v=hsaLZSKCkF0 - 11 Mars 2021

[41] Preparing for the Elastic Certified Analyst Exam - Get Elasticsearch Certified: https://www.youtube.com/watch?v=jktWd93DHAQ - 11 Mars 2021

[42] Elastic Certified Engineer Exam - My Experience and How I Prepped : https://www.linkedin.com/pulse/elastic-certified-engineer-exam-my-experience-how-i-surbhi-mahajan/ - 11 Mars 2021

[43] All About Analyzers, Part One: https://www.elastic.co/blog/found-text-analysis-part-1/ - 15 Mars 2021

[44] All About Analyzers, Part Two: https://www.elastic.co/blog/found-text-analysis-part-2 - 15 Mars 2021

[45] How to Use Fuzzy Searches in Elasticsearch: https://www.elastic.co/blog/found-fuzzy-search - 15 Mars 2021

[46] Help needed with the query : http://elasticsearch-users.115913.n3.nabble.com/help-needed-with-the-query-td3177477.html#a3178856 - 15 Mars 2021

[47] Three Principles for Multilingal Indexing in Elasticsearch: https://greg.blog/2013/05/01/three-principles-for-multilingal-indexing-in-elasticsearch/ - 15 Mars 2021

[48] Implementing a Linkedin like search as you type with Elasticsearch: https://spinscale.de/posts/2020-05-29-implementing-a-linkedin-like-search-as-you-type-with-elasticsearch.html - 17 Mars 2021

[49] Unicode® Standard Annex #15 - Unicode Normalization Forms: https://unicode.org/reports/tr15/ - 17 Mars 2021

[50] Multi-match query: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-multi-match-query.html - 17 Mars 2021

[51] Rosette Name Indexer : https://s3.amazonaws.com/resources.rosette.com/datasheets/Rosette-RNI-Comparison-EN.pdf - 23 Mars 2021

[52] Fuzzy Name Matching with Rosette : https://www.slideshare.net/ChristopherMack2/fuzzy-name-matching-with-rosette - 23 Mars 2021

[53] Intelligent Name Matching for AML & KYC: https://s3.amazonaws.com/resources.rosette.com/datasheets/Intelligent-Name-Matching-Technical-Guide.pdf - 23 Mars 2021

[54] RNI Elasticsearch Plugin Guide : https://support.rosette.com/hc/en-us/sections/360011331991-RNI-Elasticsearch-Plugin-Guide - 23 Mars 2021

[55] Rosette Plugin for Elasticsearch: https://github.com/rosette-api/rosette-elasticsearch-plugin - 25 Mars 2021

[56] Filter search results: https://www.elastic.co/guide/en/elasticsearch/reference/7.11/filter-search-results.html - 25 Mars 2021

[57] Similarity module: https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules-similarity.html - 25 Mars 2021

[58] Are You Prepared for New KYC Requirements?: https://www.rosette.com/use-cases/kyc-for-financial-institutions/ - 25 Mars 2021

[59] Elasticsearch Plugins and Integrations: https://www.elastic.co/guide/en/elasticsearch/plugins/current/index.html - 25 Mars 2021

[60] Spring Data Elastic Search vs Java High Level REST Client: https://stackoverflow.com/questions/62382457/spring-data-elastic-search-vs-java-high-level-rest-client - 25 Mars 2021

[61] Spring Data Elasticsearch - Reference Documentation: https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/ - 25 Mars 2021

[62] Understanding Names with Neural Networks - May 2020 : https://www.slideshare.net/basistech/understanding-names-with-neural-networks-may-2020 - 25 Mars 2021

[63] ElasticSearch - Searching For Human Names: https://stackoverflow.com/questions/20632042/elasticsearch-searching-for-human-names - 25 Mars 2021

[64] Tips for Elastic Certified Engineer Exam: https://github.com/zyongqing/elastic-certified-engineer - 25 Mars 2021

[65] Elasticsearch Engineer I - Lab Kit / Version 6.2.2: https://github.com/AlexJ-94/Engineer2 - 25 Mars 2021

[66] Angular - The Complete Guide (2021 Edition): https://www.udemy.com/course/the-complete-guide-to-angular-2/ - 30 Mars 2021

[67] Spring Data Elasticsearch - Examples: https://github.com/spring-projects/spring-data-examples/tree/main/elasticsearch/example - 30 Mars 2021 2021

[68] Step-by-step guide how integrate Keycloak with Angular application: https://wkrzywiec.medium.com/step-by-step-guide-how-integrate-keycloak-with-angular-application-d96b05f7dfdd - 22 June 2021

[69] How To Authenticate Your Spring Boot Application With Keycloak: https://betterprogramming.pub/how-to-authenticate-your-spring-boot-application-with-keycloak-1e9ccb5f2478 - 22 June 2021

[70] Authentication and authorization using the Keycloak REST API : https://developers.redhat.com/blog/2020/11/24/authentication-and-authorization-using-the-keycloak-rest-api - 22 June 2021

[71] A hitchhiker's guide to Spring Boot, Elasticsearch, Logstash, Kibana, PostgreSQL and Docker: https://towardsdatascience.com/a-hitchhikers-guide-to-spring-boot-elasticsearch-logstash-kibana-postgresql-and-d - 22 June 2021

[72] Using Elasticsearch with Spring Boot: https://reflectoring.io/spring-boot-elasticsearch/ - 22 June 2021

[73] Spring Boot Code Examples: https://github.com/thombergs/code-examples/tree/master/spring-boot - 22 June 2021

[74] Keycloak Security Example: https://github.com/wkrzywiec/keycloak-security-example - 22 June 2021

Appendices

Appendix A

Technologies Used: Elasticsearch

Introduction

Elasticsearch sits at the center of our solution since most of the work we have done in this project was on Elasticsearch mappings, analyzers and search queries.
So, what is Elasticsearch?
Elasticsearch is an index, a search engine, an analytics database and a big data solution all at the same time, it's also fast and scalable, free and open-source, or in more simple terms, it's kind of like Google. Elasticsearch allows the storage, search, and analysis of huge volumes of data in a near real-time fashion, giving back answers in milliseconds. It's able to achieve this feat of fast search responses due to the way it performs searches, instead of searching the text directly, it searches an index. It is also schema-free, using a structure based on documents instead of tables and schemas and comes with extensive REST APIs for storing and searching the data. At its core, you can think of Elasticsearch as a JSON-processing server, taking in JSON requests and giving back JSON data.

A.1 Concepts of Elasticsearch

In this section, we'll Introduce the most important concepts of Elasticsearch relevant to our work:

A.1.1 Documents

Just like other NO-SQL databases, Elasticsearch is based on Documents, storing data in JSON formats with the structure of the document being represented by the different fields.

A.1.2 Indices

An index is a collection of documents sharing similar characteristics. Indices represent the highest level entity that you can query against in Elasticsearch. Indices can be thought of as being equivalent to a database in a relational database schema. Documents within the same index are ordinarily logically related. In the context of an e-commerce website, for example, you can have an index for Customers, one for Products, one for Orders, and so on. An index is associated to a name that we can use to refer to the index while performing usual operations against the documents in it such as indexing new documents, searching for documents matching search queries, updating documents, and deleting documents. In other words, an index is a collection of documents, with a document being a collection of fields, and fields themselves acting as key-value pairs that contain our data.

A.1.3 Inverted Index

As the name implies, its inverted, that means, you provide the keyword, and you get all the documents it is found in. Instead of providing the document and getting all

the keywords present there. An inverted index lists every single distinct keyword that appears in any document within a given index and identifies all of the documents within which such a keyword occurs in. The way Elasticsearch indexes data is by having an optimized and dedicated data structure for each and every indexed field. Text fields, for example, use inverted indices to store keywords, and BKD trees are used to store numeric or geo fields. These dedicated and optimised data structures for each field's data are what makes Elasticsearch so fast in returning search and query results.[**19**][**20**][**21**][**22**] Figure A.1 shows how an Inverted index looks like, you just have to specify the term(s) you're looking for and Elasticsearch's inverted index gives you back all the documents within which the searched terms exist. This is similar to textbook indexes (where you look for a term at the end of the book, and it points you back to the different pages where said term was mentioned).

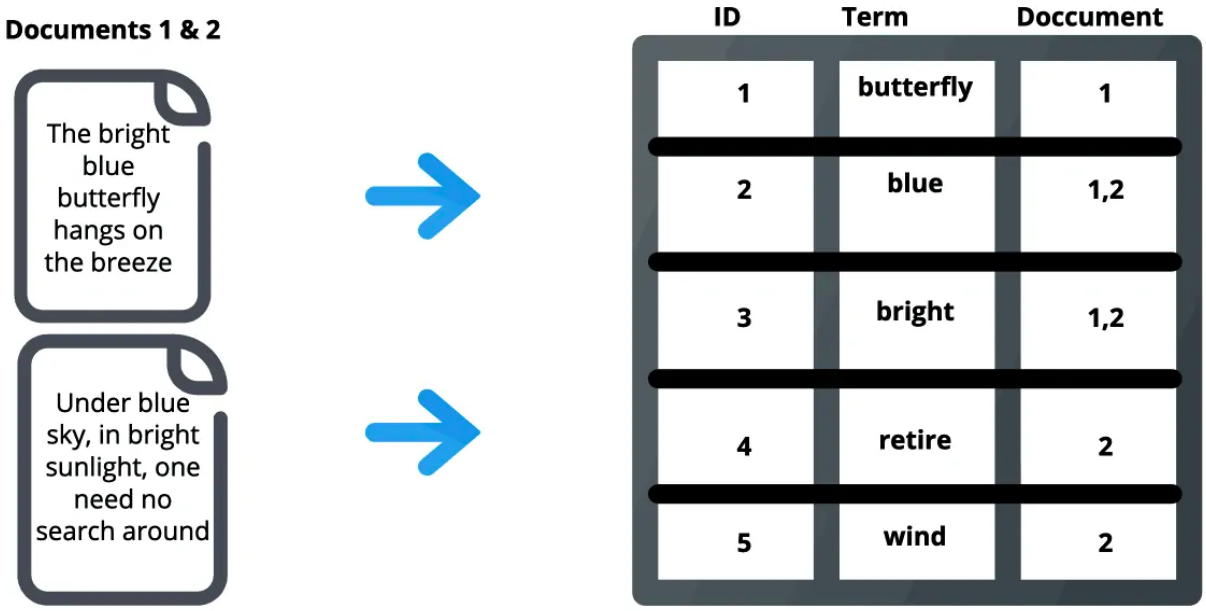

Figure A.1: Inverse Index

A.2 Introduction to Analysis

When new documents are sent to Elasticsearch for storage, they first pass by an Analysis process (See Figure A.2), with the different fields being run through an analysis process that depends on the field type and mapping. In the case of full text fields, the analysis process involves tokenizing text into terms as well as normalizing the text (through lower-casing for example). Such an operation is essential to make text easier to search. We can fully control the analysis process since we can build and choose what analyzers we wish to use, which is exactly what we have done in this project. In reality, Elasticsearch stores and uses the results of the analysis process and not the original document. In particular, the output of the analyzed text fields is stored within the inverted index, which is later used for search and retrieval. Meaning that whenever we perform a search query request, we're actually searching through the results of the analysis process and not the original

Figure A.2: How new Documents are saved into Elasticsearch

documents.

A.2.1 Analysis in more detail

A typical full text analyser is decomposed of these 3 subparts (see Figure A.3) character filters, token filters, and a tokenizer. Together, they perform the aforementioned analysis process. An analyzer is basically a packaging of these three building blocks, with each one of them changing the input stream. So when indexing a document, it goes through the flow of Figure A.3. In this project we've built many analyzers which we'll discuss in

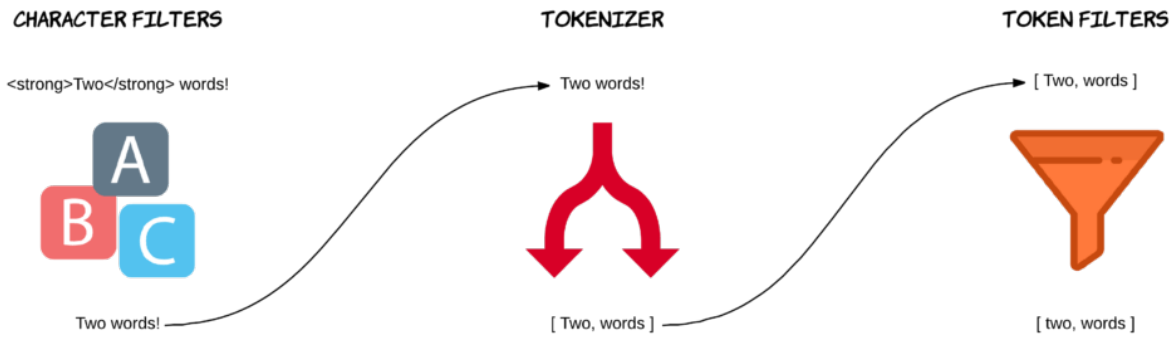

Figure A.3: How the Analysis process works

the Elasticsearch analyzers design section among other things. But before that, let's first explain the functionality of some of the constituent parts.

A.2.2 Character Filters

Character filters preprocess the stream of characters before passing it to the tokenizer.

A character filter receives the original text as a stream of characters and can transform the stream by adding, removing, or changing characters (and hence the name, character filter).

ICU Normalizer: NFD - Normalization Form Canonical Decomposition - Decomposition Mode :

This filter's implementation is provided through the ICU plugin. (Of course, the choice of of the parameters is mine) The first step for any name matching solution is definitely Unicode normalization. That is, to make two seemingly matching characters that don't due to different Unicode codes, match. Especially by using Canonical equivalence of characters.[**49**] This character filter decomposes the input/each character canonically into its constituent parts. We'll see an example test for this character filter that we used in our project. This is an example input String input string meant to show how the character C Cedilla can be encoded within a the Unicode string "Ç Ç": We'll now send it to

```
Constituent codepoints:
  0022  QUOTATION MARK
  00C7  LATIN CAPITAL LETTER C WITH CEDILLA
  0020  SPACE
  0043  LATIN CAPITAL LETTER C
  0327  COMBINING CEDILLA
  0022  QUOTATION MARK
```

Figure A.4: NFC Canonical Decomposition Mode Input Exampple

be analyzed through the icu_normalizer and see the unicode characters as they come out by using kibana's devtools (See Figure A.5): The output of the character filter (Figure

```
GET /_analyze
{
  "char_filter": [
    {
      "type": "icu_normalizer",
      "name": "nfc",
      "mode": "decompose"

    }
  ],
  "text": "Ç Ç"
}
```

Figure A.5: NFC Canonical Decomposition Mode Query for testing it

A.6, shows the Unicode characters the were outputted) is clearly different. The Cedilla part was separated. Now (See Figure A.7), we'll show what happens when you try to

```
Constituent codepoints:
  0022  QUOTATION MARK
  0043  LATIN CAPITAL LETTER C
  0327  COMBINING CEDILLA
  0020  SPACE
  0043  LATIN CAPITAL LETTER C
  0327  COMBINING CEDILLA
  0022  QUOTATION MARK
```

Figure A.6: NFC Canonical Decomposition Mode Output String

match the first two Ç cedillas from before the query was sent to kibana and from after.

You can clearly see, that even though the first two c cedillas look the same, they aren't actually, but after passing them into the character filter, they match! This explains why we used this character filter, as we wanted a way to normalize inputs. There are also

```
> "Ç" == "Ç"
<· false
> "Ç" == "Ç"
<· true
```

Figure A.7: A Comparison of the two different inputs before and after

other possible normalization forms. I have used what I believe to be the best. If you want to spend more time reading on alternatives, here's a list: Unicode Equivalence basically states that some sequences of code points represent essentially the same character. Canonically equivalent means same meaning and appearance (like the ç here). Compatibility means different appearances but same meaning(like ff and ff). This is the full list of possibilities Normalization Form Canonical Decomposition (NFD). Normalization Form Canonical Composition (NFC). Normalization Form Compatibly Decomposition (NFKD). Normalization Form Compatibly Composition (NFKC).

I have called this custom character filter nfd_normalizer.

A.2.3 Tokenizers

A tokenizer receives a stream of characters, breaks it up into individual tokens (usually individual words), and outputs a stream of tokens. This is responsible for breaking a full name into name components that will be tested as we go. We can only have one tokenizer in an analyser.

ICU_Tokenizer Tokenizers tokenizes text into words on word boundaries, as well as uses certain characters to tokenize the input. In our case, the ICU_Tokenizer tokenizes on words according to the UAX #29 specification (check it for a full list of word boundaries). It also allows us to specify a custom dictionary of word breakers if we so want. By using this tokenizer, we can separate a name into name components whenever we find a white space, punctuations, hyphens or other standard word separators. In figures A.8 and A.9, we show how this tokenizer is capable of tokenizing an example name:

```
GET /_analyze
{
  "tokenizer": "icu_tokenizer",

  "text": "Abu Ibrahim al-Hashimi al-Quraishi"
}
```

Figure A.8: An example input to the ICU_Tokenizer

We can also see that this tokenizer specifies the position of each token generated (together with a start and end positions)

```
{
  "tokens" : [
    {
      "token" : "Abu",
      "start_offset" : 0,
      "end_offset" : 3,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    {
      "token" : "Ibrahim",
      "start_offset" : 4,
      "end_offset" : 11,
      "type" : "<ALPHANUM>",
      "position" : 1
    },
    {
      "token" : "al",
      "start_offset" : 12,
      "end_offset" : 14,
      "type" : "<ALPHANUM>",
      "position" : 2
    },
    {
      "token" : "Hashimi",
      "start_offset" : 15,
      "end_offset" : 22,
      "type" : "<ALPHANUM>",
      "position" : 3
    },
    {
      "token" : "al",
      "start_offset" : 23,
      "end_offset" : 25,
      "type" : "<ALPHANUM>",
      "position" : 4
    },
    {
      "token" : "Quraishi",
      "start_offset" : 26,
      "end_offset" : 34,
      "type" : "<ALPHANUM>",
      "position" : 5
    }
  ]
}
```


Figure A.9: An example output to the ICU_Tokenizer

Ngram_tokenizer I have used an Ngram tokenizer to help in recognising names with misspellings (as long as many subparts of the name are the same). The ngram tokenizer first breaks text down into words whenever it encounters one of a list of specified characters, then it emits N-grams of each word of the specified length.

N-grams are like a sliding window that moves across the word - a continuous sequence of characters of the specified length. It is therefore helpful for giving more weight to words that share many of the same ngrams. (See figures A.10 and A.11)

```
POST _analyze
{
  "tokenizer": {
        "type": "ngram",
        "min_gram": 2,
        "max_gram": 3,
        "token_chars": [
          "letter",
          "digit"
        ]
      },
  "text":     "Abu Ibrahim al-Hashimi al-Qurashi"
}
```

Figure A.10: An example input to the Ngram_Tokenizer

```
{
  "tokens" : [
    {
      "token" : "Ab",
      "start_offset" : 0,
      "end_offset" : 2,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "Abu",
      "start_offset" : 0,
      "end_offset" : 3,
      "type" : "word",
      "position" : 1
    },
    {
      "token" : "bu",
      "start_offset" : 1,
      "end_offset" : 3,
      "type" : "word",
      "position" : 2
    },
    {
      "token" : "Ib",
      "start_offset" : 4,
      "end_offset" : 6,
      "type" : "word",
      "position" : 3
    },
    {
      "token" : "Ibr",
      "start_offset" : 4,
      "end_offset" : 7,
      "type" : "word",
      "position" : 4
    },
    {
      "token" : "br",
      "start_offset" : 5,
      "end_offset" : 7,
      "type" : "word",
      "position" : 5
    },
    {
      "token" : "bra",
      "start_offset" : 5,
      "end_offset" : 8,
      "type" : "word",
      "position" : 6
    },
    {
      "token" : "ra",
      "start_offset" : 6,
      "end_offset" : 8,
      "type" : "word",
      "position" : 7
    },
    {
      "token" : "rah",
      "start_offset" : 6,
      "end_offset" : 9,
      "type" : "word",
      "position" : 8
    },
    {
      "token" : "ah",
      "start_offset" : 7,
      "end_offset" : 9,
      "type" : "word",
      "position" : 9
    },
    {
      "token" : "ahi",
      "start_offset" : 7,
      "end_offset" : 10,
      "type" : "word",
      "position" : 10
    },
    {
      "token" : "hi",
      "start_offset" : 8,
      "end_offset" : 10,
      "type" : "word",
      "position" : 11
    },
    {
      "token" : "him",
      "start_offset" : 8,
      "end_offset" : 11,
      "type" : "word",
      "position" : 12
    },
    {
      "token" : "im",
      "start_offset" : 9,
      "end_offset" : 11,
      "type" : "word",
      "position" : 13
    },
```


Figure A.11: An example output to the Ngram_Tokenizer

A.2.4 Token Filters

Token filters accept a stream of tokens from a tokenizer and can modify tokens (eg lowercasing, transliteration...), delete tokens (eg remove stopwords) or add tokens (eg synonyms).

Shingle Shingles are effectively word-nGrams. Given a stream of tokens, the shingles filter will create new tokens by concatenating adjacent terms. Shingles effectively give us the ability to match names even if their constituent parts are concatenated (spaces missing). By building such shingles into the index, you can avoid creating and testing for concatenated names at query time and save some processing time/speed. (For example, this is helpful when someone writes two name components concatenated without putting a space between them, or separates them using a character that our tokenizer doesn't consider as a word boundary or something unaccounted for like that...

Here's an example for how the Shingles token works in action (See Figure A.12 and A.13, of course, it'll be used with other normalizing character filters and token filters, to lowercase the inputs for example among other things...)

```
GET /_analyze
{

  "tokenizer": "icu_tokenizer",
  "filter": [
    {
        "type": "shingle",
        "max_shingle_size": 4,
        "min_shingle_size": 2,
        "output_unigrams": "true",
        "token_separator": ""
    }
  ],

  "text": "Rutherford Birchard Hayes "
}
```

Figure A.12: An example input to the Shingles token filter

```
{
  "tokens" : [
    {
      "token" : "Rutherford",
      "start_offset" : 0,
      "end_offset" : 10,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    {
      "token" : "RutherfordBirchard",
      "start_offset" : 0,
      "end_offset" : 19,
      "type" : "shingle",
      "position" : 0,
      "positionLength" : 2
    },
    {
      "token" : "RutherfordBirchardHayes",
      "start_offset" : 0,
      "end_offset" : 25,
      "type" : "shingle",
      "position" : 0,
      "positionLength" : 3
    },
    {
      "token" : "Birchard",
      "start_offset" : 11,
      "end_offset" : 19,
      "type" : "<ALPHANUM>",
      "position" : 1
    },
    {
      "token" : "BirchardHayes",
      "start_offset" : 11,
      "end_offset" : 25,
      "type" : "shingle",
      "position" : 1,
      "positionLength" : 2
    },
    {
      "token" : "Hayes",
      "start_offset" : 20,
      "end_offset" : 25,
      "type" : "<ALPHANUM>",
      "position" : 2
    }
  ]
}
```


Figure A.13: An example output to the Shingles token filter

ICU_Transform Transforms are used to process Unicode text in many different ways, an important way for us is the ability to transliterate in our case. [**29**]

This transform (See Figure A.14 and Figure 1.15) transliterate characters in other scripts into Latin, and separates accents from their base characters using NFD (We talked about earlier), removes the accents, and then puts the remaining text into an unaccented form. This is a pretty basic character to character transliteration which is good enough for our use case (I was the one who proposed we also work with different scripts than latin).

We need something that can transliterate from Arabic, Russian, Chinese and Korean (most terrorists are originally from one of these ethnicities/origins) into Latin.

Of course, there are special transliterators for each of these languages, but in order to make the most out of them, we need to have a language detector which I don't know how to do and how to select a different analyzer depending on such an output. So I set on only this simple generic transliterator that is still very good in my opinion.

In figures A.14 and A.15, we test out a name written in Arabic Script, and can see the result of the analysis process is the name transliterated into latin.

```
GET /_analyze
{
  "tokenizer": "keyword",
  "filter": [
    {
      "type": "icu_transform",
      "id": "Any-Latin; NFD; [:Nonspacing Mark:] Remove; NFC"
    }
  ],

  "text": "أبو ابراهيم الهاشمي القريشي سيبرس"
}
```

Figure A.14: An example input name in arabic script to the ICU_Transform

```
{
  "tokens" : [
    {
      "token" : "abw abrahym alhashmy alqryshy sybrs",
      "start_offset" : 0,
      "end_offset" : 33,
      "type" : "word",
      "position" : 0
    }
  ]
}
```

Figure A.15: An example output name from arabic arabic script to latin script by the ICU_Transform

In figures A.16 and A.17, we test out a name written in Russian Cyrillic script, and can see the result of the analysis process is the name transliterated into latin.

```
"text": "Русское Имперское Движение"
```

Figure A.16: An example input name in russian cyrillic script to the ICU_Transform

```
"token" : "Russkoe Imperskoe Dvizenie",
```

Figure A.17: An example output name from russian cyrillic script to latin script by the ICU_Transform

And in figures A.18 and A.19, we test out a name written in Chinese Hanzi script, and can see the token filter returns the name transliterated into latin. Finally, in figures

```
"華信船務(香港)有限公司"
```

Figure A.18: An example input name in Chinese Hanzi Script to the ICU_Transform

```
"hua xin chuan wu (xiang gang) you xian gong si",
```

Figure A.19: An example output name from Chinese Hanzi Script to latin script by the ICU_Transform

A.18 and A.19, we test out a name written in Korean Hangul script, and we can see the token filter returns the name transliterated into latin!

```
"조선 대성 은행"
```

Figure A.20: An example input name in korean Hangul Script to the ICU_Transform

```
"joseon daeseong eunhaeng",
```

Figure A.21: An example output name from Korean Hangul Script to latin script by the ICU_Transform

Phonetic Encoders: Double Metaphone and Beider-Morse I have used mainly two phonetic encoders in this project, double metaphone and Beider-Morse. I suggest you read about them, as well as about Levenshtein distance. I'll show how an example input for the name "Joanne" gets treated by both these algorithms. (See Figures A.22 and A.23 to see how the name gets encoded using the double metaphone algorithm, and Figues A.24 and A.25 to see how the name get encoded using the Beider Morse Analyzer.)[**27**][**31**][**37**]

These phonetic encoders try to encode the sound that a certain string of text encodes.

```
GET /_analyze
{
  "tokenizer": "keyword",
  "filter": [
    {
      "type": "phonetic",
      "encoder": "double_metaphone",
      "replace": true
    }
  ],

  "text": "JOANNE"
}
```

Figure A.22: An example input name using the double metaphone encoder

```
{
  "tokens" : [
    {
      "token" : "JN",
      "start_offset" : 0,
      "end_offset" : 6,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "AN",
      "start_offset" : 0,
      "end_offset" : 6,
      "type" : "word",
      "position" : 0
    }
  ]
}
```

Figure A.23: An example output name after being encoded using the double metaphone phonetic encoder

```
GET /_analyze
{
  "tokenizer": "keyword",
  "filter": [
    {
      "type" : "phonetic",
      "encoder" : "beider_morse",
      "replace": true
    }
  ],

  "text": "JOANNE"
```

Figure A.24: An example input name using the beider morse encoder

```
{
  "tokens" : [
    {
      "token" : "ioDni",
      "start_offset" : 0,
      "end_offset" : 6,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "ioYni",
      "start_offset" : 0,
      "end_offset" : 6,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "ioan",
      "start_offset" : 0,
      "end_offset" : 6,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "ioani",
      "start_offset" : 0,
      "end_offset" : 6,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "ioon",
      "start_offset" : 0,
      "end_offset" : 6,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "iooni",
      "start_offset" : 0,
      "end_offset" : 6,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "xoani",
      "start_offset" : 0,
      "end_offset" : 6,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "xooni",
      "start_offset" : 0,
      "end_offset" : 6,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "zoDni",
      "start_offset" : 0,
      "end_offset" : 6,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "zoan",
      "start_offset" : 0,
      "end_offset" : 6,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "zoani",
      "start_offset" : 0,
      "end_offset" : 6,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "zoon",
      "start_offset" : 0,
      "end_offset" : 6,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "zooni",
      "start_offset" : 0,
      "end_offset" : 6,
      "type" : "word",
      "position" : 0
    }
  ]
}
```


Figure A.25: An example output name after being encoded using the beider morse phonetic encoder

Conclusion

We have seen many character filters so far, many tokenizers and token filters, as well as introduced Elasticsearch in general and the different things that make such a search engine unique. I hope that knowledge of these different aspects can help understand my work more in depth.