Bachelor’s Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

An application for healthcare taxonomy translation

Lukáš Kaufmann

Supervisor: Ing. Petr Křemen, Ph.D.
Field of study: Software engineering
May 2024

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Kaufmann Lukáš**	Personal ID number:	**510649**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Software Engineering and Technology**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

An application for healthcare taxonomy translation

Bachelor’s thesis title in Czech:

Aplikace pro překlad taxonomie v oblasti zdravotní péče

Guidelines:

Goal of the thesis is to design and implement a prototype for community-driven translation of the international healthcare classification SNOMED-CT to czech. The tool should support the work of the translator (e.g. by offering automated translations), allow collaboration on the translations and release the final and approved terminology in the format compatible with the SNOMED-CT standard.

1. Familiarize yourself with SNOMED-CT, the RF2 format and related technologies and tools
2. Design a system that will support translators (as described above)
3. Implement the prototype as a web application (preferably Java - backend, ReactJS - frontend)
4. Design automated unit and integration tests and test the functionality with prospective users.

Bibliography / sources:

1. SNOMED-CT, https://www.snomed.org/
2. Chang E, Mostafa J. The use of SNOMED CT, 2013-2020: a literature review. J Am Med Inform Assoc. 2021 Aug 13;28(9):2017-2026. doi: 10.1093/jamia/ocab084. PMID: 34151978; PMCID: PMC8363812.https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8363812/
3. Arcan, Mihael; Dragoni, Mauro; Buitelaar, Paul. ESSOT: an expert supporting system for ontology translation, 2016. https://aran.library.nuigalway.ie/bitstream/handle/10379/14884/NLDB2016OTTO.pdf?sequence=1

Name and workplace of bachelor’s thesis supervisor:

Ing. Petr Křemen, Ph.D. Knowledge-based Software Systems FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **07.02.2024** Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Petr Křemen, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt	Student’s signature

Acknowledgements

I would like to thank my supervisor Ing. Petr Křemen, Ph.D. for his guidance and assistance in completing this thesis. Thanks to my friends who helped me test the application. Also, thanks to my parents for their support and understanding.

Declaration

I hereby declare that this work is my own and that I have cited all the sources used. For some boilerplate parts of the code, I have used the AI tool GitHub Copilot.

Lukáš Kaufmann
Prague, 24th of May 2024

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu. Pro určité základové a opakující se prvky kódu jsem využil UI nástroj GitHub Copilot.

Lukáš Kaufmann
V Praze, 24. května 2024

Abstract

This thesis aims to build a tool that can be used to translate the medical taxonomy SNOMED-CT. It walks the reader through the process from analysis to testing and can also help the reader get a rough idea about the software development process. Alongside the development process, a short discussion and description of GraphQL can be used to gather a rough idea about it. The result is an application which has undergone automated and manual testing. From which the back end key components are almost fully covered and the front has been rated by testers as usable but occasionally slow.

Keywords: translation, ReactJS, Java, cooperation, ontology, SNOMED-CT, GraphQL, Spring Boot

Supervisor: Ing. Petr Křemen, Ph.D.

Abstrakt

Tato práce si stanoví za cíl vytvořit nástroj pro překlad medicínské taxonomie SNOMED-CT. Na cestě za slpněním tohoto cíle představí proces vytvoření takové aplikace od analýzy po testování. Představí použité technologie, hlavně představí GraphQL jako API technologii. Popisuje terstování jak automatizované, tak manuální včetně diskuze na vásledky. Výsledkem je aplikace, která plní většinu požadovaných funkcionalit a byla ohodnocena testery jako použitelná, ale místy pomalá.

Klíčová slova: překlad, ReactJS, Java, spolupráce, ontologie, SNOMED-CT, GraphQL, Spring Boot

Překlad názvu: Aplikace pro překlad taxonomie v oblasti zdravotní péče

Contents

Figures

Tables

Chapter 1

Introduction

In the world of modern medicine, much of the paperwork is either written on paper or is a creative work of each medical practitioner describing the same things but each in their own words. So there are a lot of plans to go digital and move to Electronic health records (EHR). One such system is being implemented in the UK by the NHS [1, 2].

But with such a move come many questions. The major of which is how to describe diseases, symptoms, and other clinical terms. Using just text will make it very hard to search for relevant information and make it almost impossible to reliably transfer information between doctors and correctly interpret complex patient cases, including their medical histories. It is even harder to translate medical records if your doctor is in another country and does not speak the language.

There is a long history of creating medical classification systems. One such system is SNOMED-CT, which dates its roots back to 1955. The College of American Pathologists (CAP) was established to develop a nomenclature for anatomic pathology. Ten years later, they came up with the first system called *"Systemized Nomenclature of Pathology (SNOP)"* based on four key aspects: Topography (anatomic site affected), Morphology (structural changes associated with the disease), Etiology (the cause of the disease) and Function (physiological alterations associated with the disease). By 1975 SNOP was expanded with diseases and procedures and was renamed to *"Systemized Nomenclature of Medicine (SNOMED)"*. [3, p. 228] Finally in 1999 the CAP worked together with National Health Service of the United Kingdom (NHS) and merged their clinical terminologies to create SNOMED CT (Clinical Terminologies). [3, p. 234]

In 2007, International Health Terminology Standards Development Organisation (IHTSDO) was founded by nine countries (Australia, Canada, Denmark, Lithuania, Sweden, the Netherlands, New Zealand, the United Kingdom, and the United States). With the founding, they also acquired the rights to SNOMED CT to develop a global clinical terminology. Since its inception, the project has already expanded to 48 members, as can be seen in Figure 1.1.

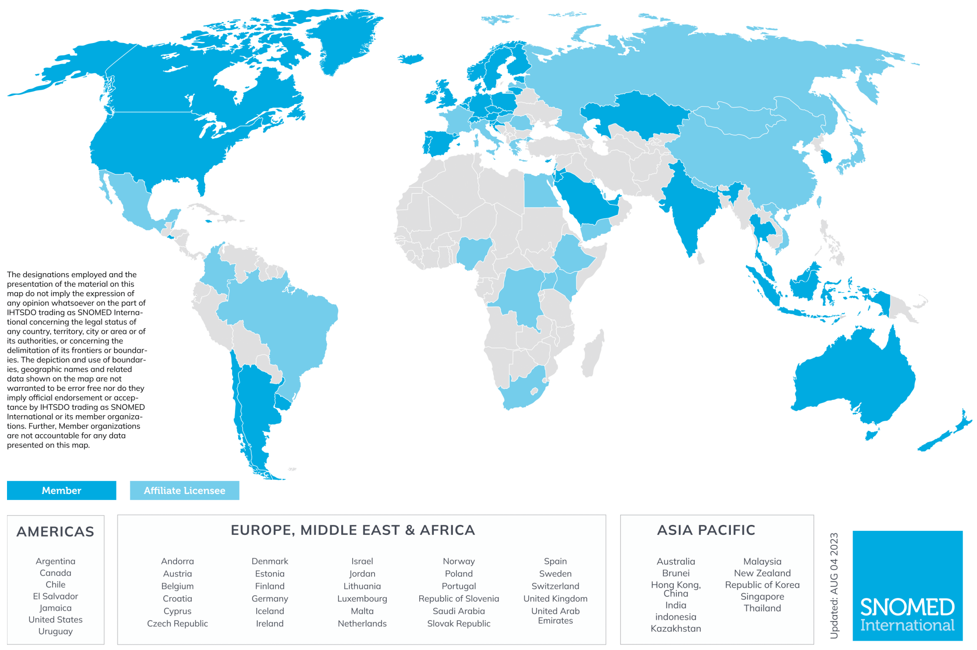

The designations employed and the presentation of the material on this map do not imply the expression of any opinion whatsoever on the part of IHTSDO trading as SNOMED International concerning the legal status of any country, territory, city or area or of its authorities, or concerning the delimitation of its frontiers or boundaries. The depiction and use of boundaries, geographic names and related data shown on the map are not warranted to be error free nor do they imply official endorsement or acceptance by IHTSDO trading as SNOMED International or its member organizations. Further, Member organizations are not accountable for any data presented on this map.
Member
Affiliate Licensee
AMERICAS
Argentina
Canada
Chile
El Salvador
Jamaica
United States
Uruguay
EUROPE, MIDDLE EAST & AFRICA
Andorra
Austria
Belgium
Croatia
Cyprus
Czech Republic
Denmark
Estonia
Finland
Germany
Iceland
Ireland
Israel
Jordan
Lithuania
Luxembourg
Malta
Netherlands
Norway
Poland
Portugal
Republic of Slovenia
Saudi Arabia
Slovak Republic
Spain
Sweden
Switzerland
United Kingdom
United Arab Emirates
ASIA PACIFIC
Australia
Brunei
Hong Kong, China
India
indonesia
Kazakhstan
Malaysia
New Zealand
Republic of Korea
Singapore
Thailand
Updated: AUG 04 2023
SNOMED International

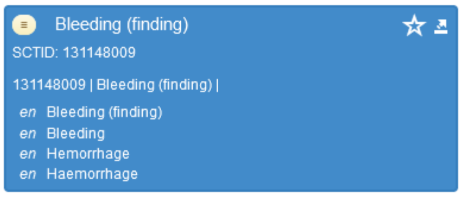
Figure 1.1: Members of SNOMED [4]

Because the system originated in the USA it is in English. That brings the need to translate the terminology into multiple languages. The origin of all terminologies, the base set, is referred to as the International Edition, which is used as the starting point for all other editions, extensions and translations.

Using the example of the concept which, in the International Edition, is referred to as "*Bleeding (finding)*" let us compare different national editions in Figure 1.2. Although the Concept ID has remained the same (131148009), the name and associated descriptions may have changed.

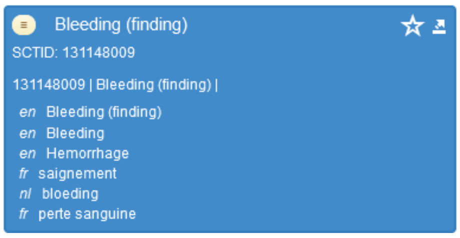
The aim of this thesis is to design and create an open source translation tool to help with the transition to SNOMED-CT as the unified clinical terminology. The existing tool from IHTSDO (described in Section 3.2.1) works as a summary of all existing published translations and requires their participation. The existing tool aims to provide a solution managed by the community and a simplified way to separate the set into parts that respect the specialities of the medical practitioners involved in the translation.

The thesis is divided into chapters. Chapter 2 describes the goals and the technical aspects that need to be considered. Chapter 3 introduces the SNOMED-CT landscape. Chapter 4 sets the requirements and describes the entities in the domain of the intended application. Chapter 5 describes the architecture and the technologies used to create the app. Chapter 6 describes the methodology, realisation, and results of undertaken testing. The works is concluded with Chapter 7.



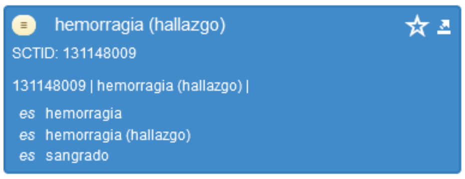
Bleeding (finding)
SCTID: 131148009
131148009 | Bleeding (finding) |
en Bleeding (finding)
en Bleeding
en Hemorrhage
en Haemorrhage

(a) : International edition



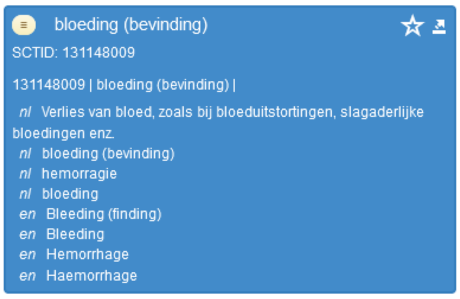
Bleeding (finding)
SCTID: 131148009
131148009 | Bleeding (finding) |
en Bleeding (finding)
en Bleeding
en Hemorrhage
fr saignement
nl bloeding
fr perte sanguine

(b) : Belgian edition



hemorragia (hallazgo)
SCTID: 131148009
131148009 | hemorragia (hallazgo) |
es hemorragia
es hemorragia (hallazgo)
es sangrado

(c) : Spanish edition



bloeding (bevinding)
SCTID: 131148009
131148009 | bloeding (bevinding) |
nl Verlies van bloed, zoals bij bloeduitstortingen, slagaderlijke bloedingen enz.
nl bloeding (bevinding)
nl hemorragie
nl bloeding
en Bleeding (finding)
en Bleeding
en Hemorrhage
en Haemorrhage

(d) : Netherlands edition

Figure 1.2: Comparison of different national editions

Chapter 2
Goals

The primary goal is to provide a platform for translations that can be adopted to meet the needs of the specific national health agency that adopts SNOMED-CT.

Tasking one person with translating such a large set would take a long time and might not be useful to the medical community. Because of that, this tool should use the concept of crowd-sourcing. However, it is key to have the right crowd, so there must be an administrative step of approving people to be able to translate.

With a large number of contributors, it must be possible not only to suggest translations, but also to rate existing ones and comment on them. Even though the contributors will most likely be from the medical field, and even with a good cause some may choose to deface the translations, as such there must also be a process to remove translations and comments which bear no relation to the term or topic at hand.

From such a large set of possible translations, the preferred translation must be chosen by a group of translators qualified and responsible for the correctness, and it must be chosen whether to use any other as synonyms. With that, the national agency would review the translations. After the review process, a national set could be created and submitted to SNOMED as the official national edition.

2.1 Technical aspects

To make participation in the translation as easy as possible, the front end of such a platform will be a website. Since three key subjects are involved with the platform: the national health agency, the medical community of the given country, and the IT professionals who manage the platform; it is imperative to have a role-based access model, as shown in Figure 2.1.

The back end operating the platform is designed as a multi-tier service-orientated monolith application. The service-orientated structure makes a clear distinction between the areas of responsibility between the parts of the business logic behaviour handling. Packing the app as a monolith makes it slightly easier to develop, thanks to having direct access between parts of the code. The major benefit of monolith is ease of deployment, one server running Java is needed to run the app, or a container with the app makes it even easier. Unlike a microservice application, where multiple servers, a load balancing solution, and a communication channel would need to be established and maintained.

Using a relational database to store interim information about translations, comments, reviews, etc. To house the reference clinical terminology that is being translated, a SnowStorm [5] server with the desired version of the clinical terminology.

To make sure the platform works as intended, a plethora of tests should verify the functionality. The elementary functions should be covered by unit tests. For key functionalities integration tests will be used. To also verify complete end-to-end functionality, test scenarios tested manually will be used.

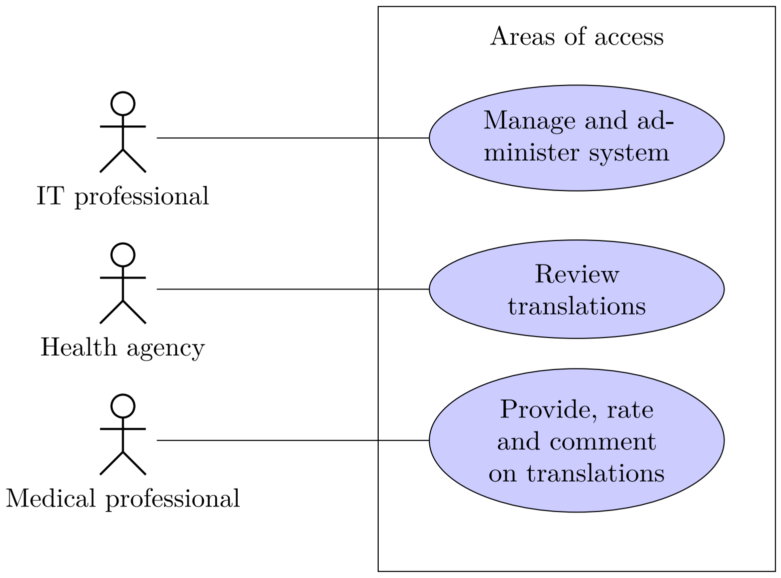


Figure 2.1: Areas of expertise

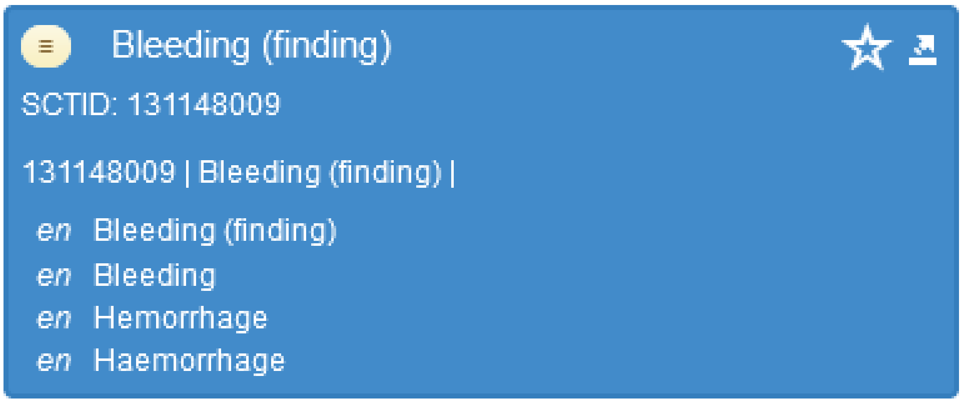
Chapter 3

SNOMED-CT

"SNOMED CT or SNOMED Clinical Terms is a systematically organised computer-processable collection of medical terms providing codes, terms, synonyms and definitions used in clinical documentation and reporting." [6]

3.1 Clinical terminology

The terminology consists of three essential parts: **Concepts**, **Relationships** and **Descriptions**. [7]
Concepts carry the essential clinical meaning. Every concept has its specific identifier that is unique and independent of the human-readable form, which might be in a different language. An example of a concept is shown in Figure 3.1, the unique identifier being marked as "*SCTID*" and for the finding of bleeding is *131148009*.


Bleeding (finding)

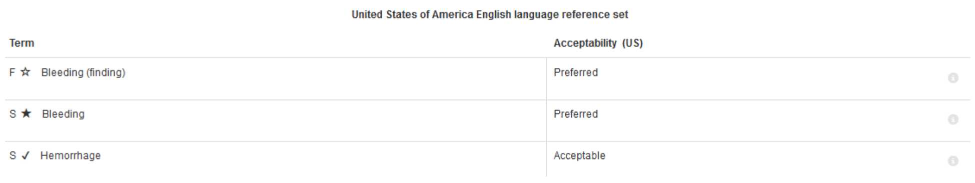
SCTID: 131148009

131148009 | Bleeding (finding) |

en Bleeding (finding)
en Bleeding
en Hemorrhage
en Haemorrhage

Figure 3.1: Concept example [8]

Descriptions are linked to concepts and provide a human-acceptable descriptor for concepts. Each concept has at least one Fully Specified Name (FSN) [9], which must represent the concept regardless of context and be unique. Along FSN there might exist any number of synonyms, which might not be unique, but in every language version of SNOMED-CT must be one preferred synonym. For concept *131148009* from Figure 3.1 we can look at all the descriptions in US English in Figure 3.2. Marked with the letter *F* at the beginning is the FSN, synonyms are marked with the letter S and of these one is marked as *preferred*.

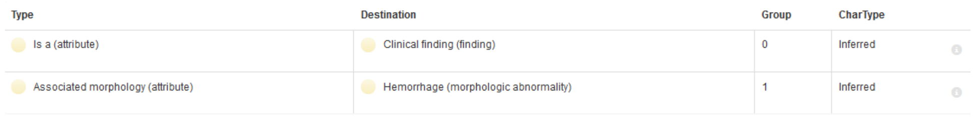


United States of America English language reference set

Term	Acceptability (US)
F ☆ Bleeding (finding)	Preferred
S ★ Bleeding	Preferred
S ✓ Hemorrhage	Acceptable

Figure 3.2: Descriptions in US English of concept *131148009* [8]

Relationships create a network of concepts and enable categorisation of them. Every concept has at least one relationship of the type "is a". Using the concept from Figure 3.1, Figure 3.3 shows that it has two relationships, one of type "*is a*" and another of type "*Associated morphology*".



Type	Destination	Group	CharType
Is a (attribute)	Clinical finding (finding)	0	Inferred
Associated morphology (attribute)	Hemorrhage (morphologic abnormality)	1	Inferred

Figure 3.3: Relationship of the concept of Bleeding (Finding) [8]

For hosting and working with the SNOMED-CT, International Health Terminology Standards Development Organisation (IHTSDO) developed a server, built on top of ElasticSearch, to host and serve the SNOMED-CT data set. [5]

3.2 Translation

Translating clinical terminology has its specifics, as it is key to ensure that translations follow the guidelines of SNOMED-CT: **understandability**, **reproducibility** and **usefulness** [10]. To cover understandability and reproducibility, it is key to gather feedback from other medical professionals. The usefulness must be evaluated by the health agency and other members of the medical community to verify that "*the concept has a practical value for users that is self-evident or can be easily explained*" [10, Section 1.1.1].

The existing tools can solve the same use case, but from rough User Interface (UI) evaluation of the two tools with available public-facing information [11, 12] those applications are primarily aimed at people knowledgeable about the SNOMED-CT structure, while this application aims to gather translations from those who use SNOMED-CT terms.

3.2.1 Existing tools

SNOMED itself provides a list of tools usable to translate [11]. Of those, only two provide some information publicly. The tools from Rhapsody (formerly CareCom) and termSpace have no publicly available information, the only option being to request a demo.

Reference Set and Translation Tool

SNOMED itself provides a tool for translating and collaborating. From my research, this tool is primarily optimised for translations of whole data sets or their sub-sections. Concerning a tool planned in the scope of this project and the following thesis, this tool looks like a helpful step after translating concepts and packaging them in an extension or a translation of the reference set.

Snow Owl B2i

B2i Healthcare is a for-profit company, however, information about their platform [13] is not only for SNOMED-CT but also for other healthcare terminology. In addition, there is a large amount of public information about their platform which presents its features.

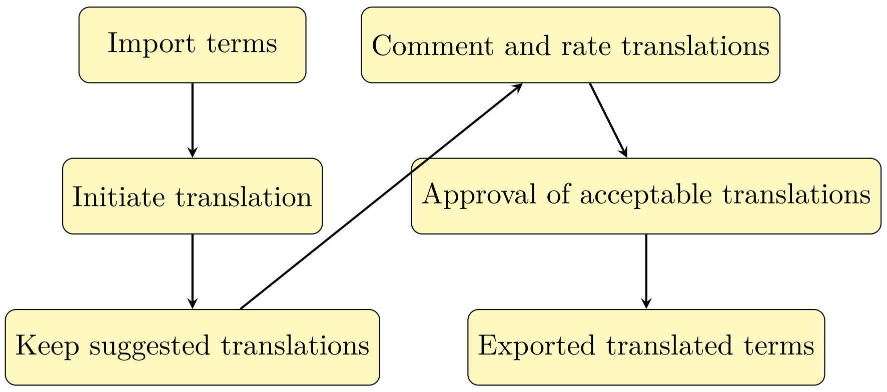
This information is a great resource for understanding the basics of translating and working with medical terminologies.

Chapter 4

Analysis

4.1 Functionalities

To help visualise and explore the functionalities needed, a basic workflow will help. It provides an overview of the rudimentary functionalities needed for this software.



Import terms
Comment and rate translations
Initiate translation
Approval of acceptable translations
Keep suggested translations
Exported translated terms

Figure 4.1: Initial idea workflow

The workflow in Figure 4.1 gives an overview from which a list of functionalities can be built.

Functionality 1 (Set preparation). *The first two steps in Figure 4.1 create the need to prepare a translation set. Import all terms relevant to the set, create a structure of Translation Manager(s) and Moderator(s) if need be, and onboard Translators to the set.*

Functionality 2 (Get translation). *Gather suggestions, ratings, and comments from the translators.*

Functionality 3 (Review translations). *Lock the suggestions to not be modified during the review. Review the suggestions and, if acceptable, mark them as an approved translation. If there are terms that are unacceptable, those can be returned to the translators and allow for the gathering of more suggestions.*

Functionality 4 (Set export). *Package the whole set to be published as either an extension or an edition.*

4.2 User roles

To be able to design the platform, an expansion of the high-level functionality from Section 4.1 is needed. The initial breakdown step is to define use cases.

To be able to define the use cases, we need to define roles that can interact with the platform.

User Role 1. *A system administrator is an IT specialist managing the technical aspects of the platform, such as the export and import of the terms to be translated.*

User Role 2. *The translation manager role should be filled by a person knowledgeable of the scope of the translation, most likely appointed by the national health agency to supervise the translation process.*

User Role 3. *When the scope and/or the number of users contributing grows sufficiently large, a moderator role might be needed to help moderate the ratings of translation suggestions. A moderator will be assigned to a specific translation set.*

The actions of the moderator, User Role 3, will include the removal of a comment if it is not relevant, aggressive, or other conduct not helpful with translations; If the user proves to be unreliable or damaging to the effort, his ratings could be removed to clean up the data.

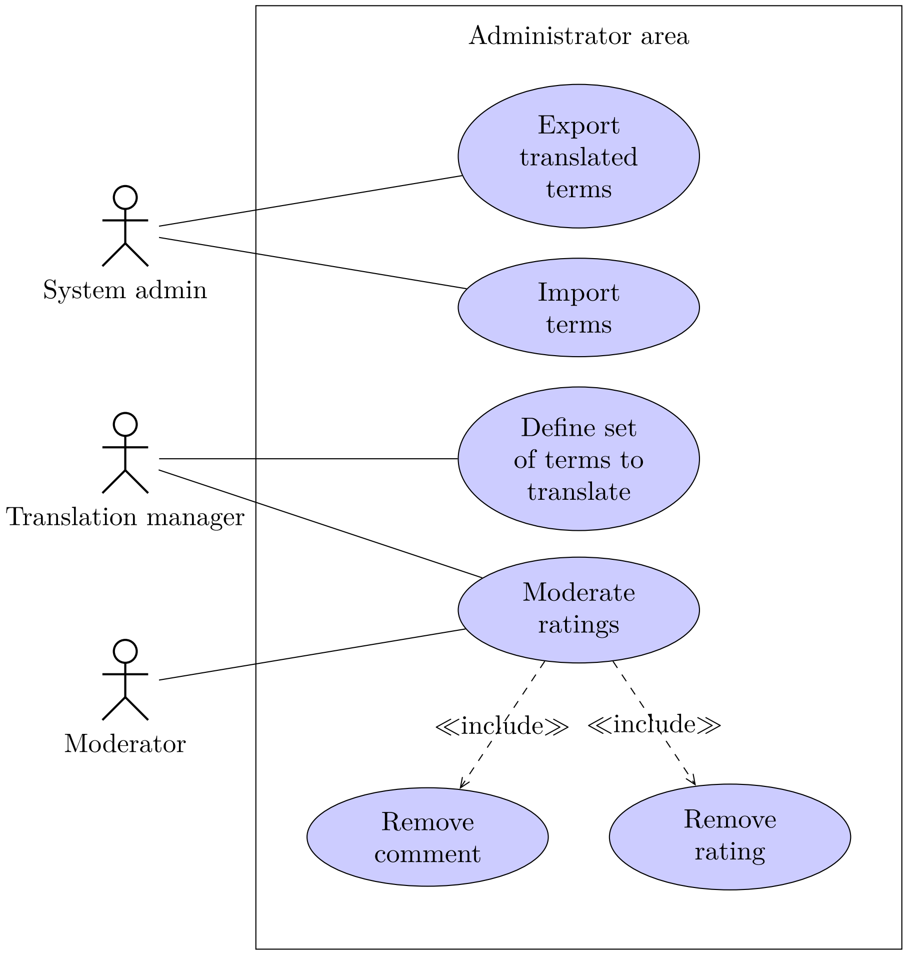


Figure 4.2: Administrator use cases

User Role 4. *The translator is the elementary participant as their role is the key to creating translations. They will be able to suggest translations, rate what others have suggested, and crucially comment on the suggestions that spark a helpful debate.*

User Role 5. *To properly verify the translations when the set of terms is being reviewed, the translation manager may employ verifiers who will go through the suggestions and approve the translations that make sense, but those that are unrelated or non-sensical will be removed.*

The approved translations will then be packaged into an approved set and given to a professional review board of the national health agency, and if it passes that review, it will be packaged as an extension or a localised edition of SNOMED-CT.

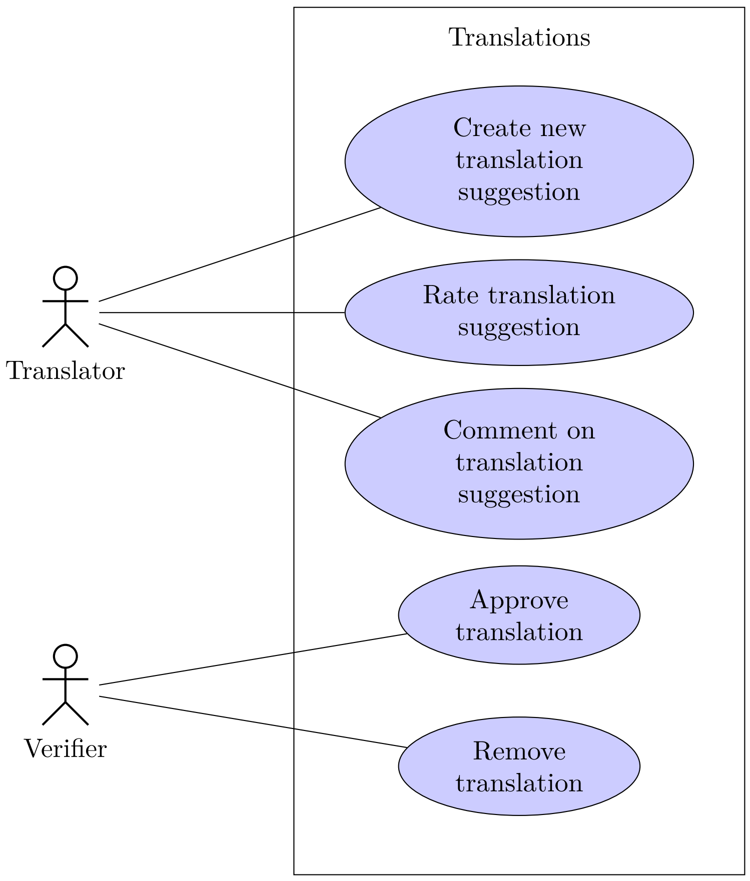

Translations
Create new translation suggestion
Rate translation suggestion
Comment on translation suggestion
Approve translation
Remove translation
Translator
Verifier

Figure 4.3: Translation use cases

4.3 Functional requirements

From Figures 4.2 and 4.3 we can build a list of requirements:

Functional Requirement 1 (Terms definition). *The platform shall allow the administrator to define which terms are available for translation.*

Functional Requirement 2 (Export a set). *The platform shall allow the administrator to export a set of translated terms.*

Functional Requirement 3 (Create set). *The platform shall allow the translation manager to create a set of terms to be translated.*

Functional Requirement 4 (Moderate). *The platform shall allow the translation manager and moderator to moderate comments and ratings.*

Functional Requirement 5 (New translation suggestion). *The platform shall allow the user to create new translation suggestions.*

Functional Requirement 6 (Rate translation suggestion). *The platform shall allow the user to rate existing suggestions.*

Functional Requirement 7 (Comment on translation suggestion). *The platform shall allow the user to comment on existing suggestions.*

Functional Requirement 8 (Approve translation). *The platform shall allow the verifier to approve a suggestion.*

Functional Requirement 9 (Remove translation). *The platform shall allow the verifier to remove a suggestion.*

Non-Functional Requirement 1 (RBAC). *The platform follows a role-based access control (RBAC) model.*

4.4 Domain model

Since all parts of the terminology are concepts that have terms associated with them, *Term* is the keystone to translating concepts.

The translations themselves are relatively simple in terms of their domain. All that is needed is a translation, its rating, and comments as shown in the part of the model in Figure 4.4.

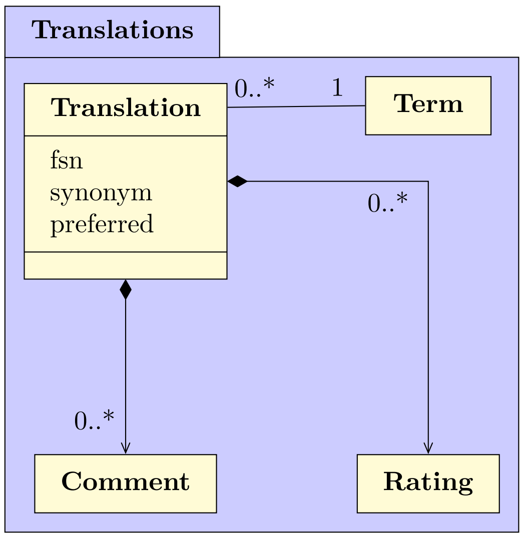


Figure 4.4: Domain model of the translation space (only metadata for Translation shown)

The complexity could increase when building the final set. However, to build the set, a SnowStorm server should be used. Despite that to allow for saving some terms but returning others to gather more feedback or better translation suggestions, we need to save some metadata with those translations. Such metadata will be part of the translation entity in Figure 4.4.

The properties in the model of the Translation entity are all Boolean flags to represent various metadata: fsn, synonym, and preferred are metadata relating directly to SNOMED-CT as explained in Section 3.1.

Building a set of terms to translate, the translation manager, who is just a type of user, will mark those concepts as part of the translation set, with the need to store the original wording to have a starting point for the translations. The terms store the concept ID from SNOMED-CT to prevent duplication and allow linking, however, the FSN is also stored to reduce the number of API calls. Those entities and their relations are shown in Figure 4.5.

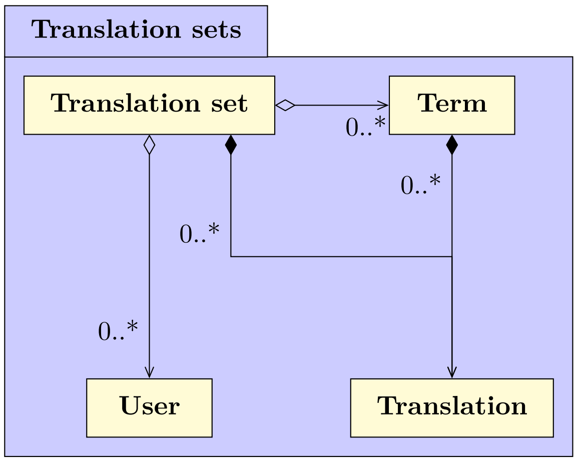


Figure 4.5: Domain model of the translation set space

To get the terms from the SnowStorm server into the application, an import has to take place. Due to expected longer execution times and possible errors, it is better to keep track of the import job and track its state and times of start and end, to allow for evaluation of the time taken. The target refset is stored as the ID of the refset in the SnowStorm server.

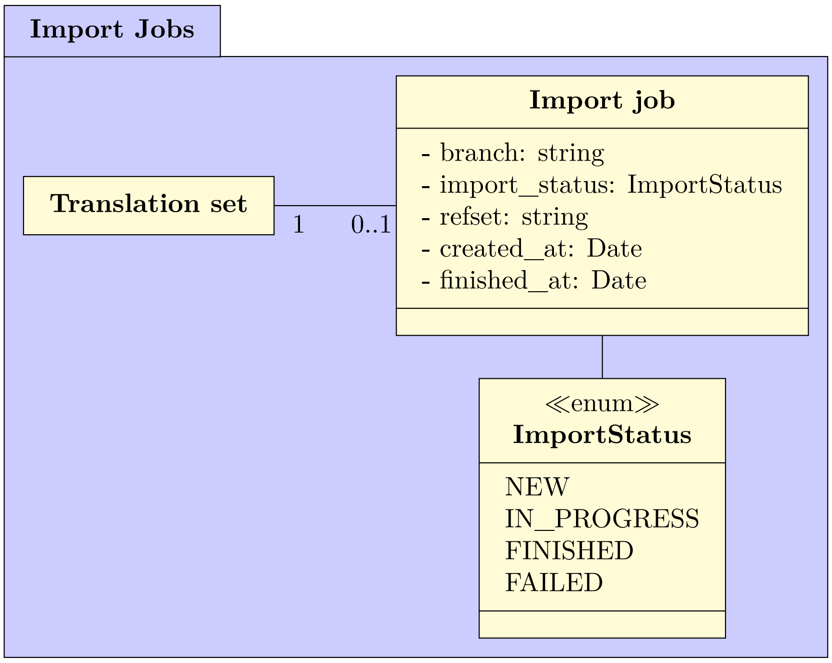


Figure 4.6: Domain model of the import job and translation set relation

From Figures 4.4, 4.5 and 4.6 we can build a complete domain model that encompasses all the essentials of the platform. This model is shown in Figure 4.7. All fields on all the entities are mandatory with the exception of: finished_at on Import Job which only gets filled after a job finished; definition on Term only if found; and flags on Translation (FSN, synonym and preferred) are optional and only get set in the review process.

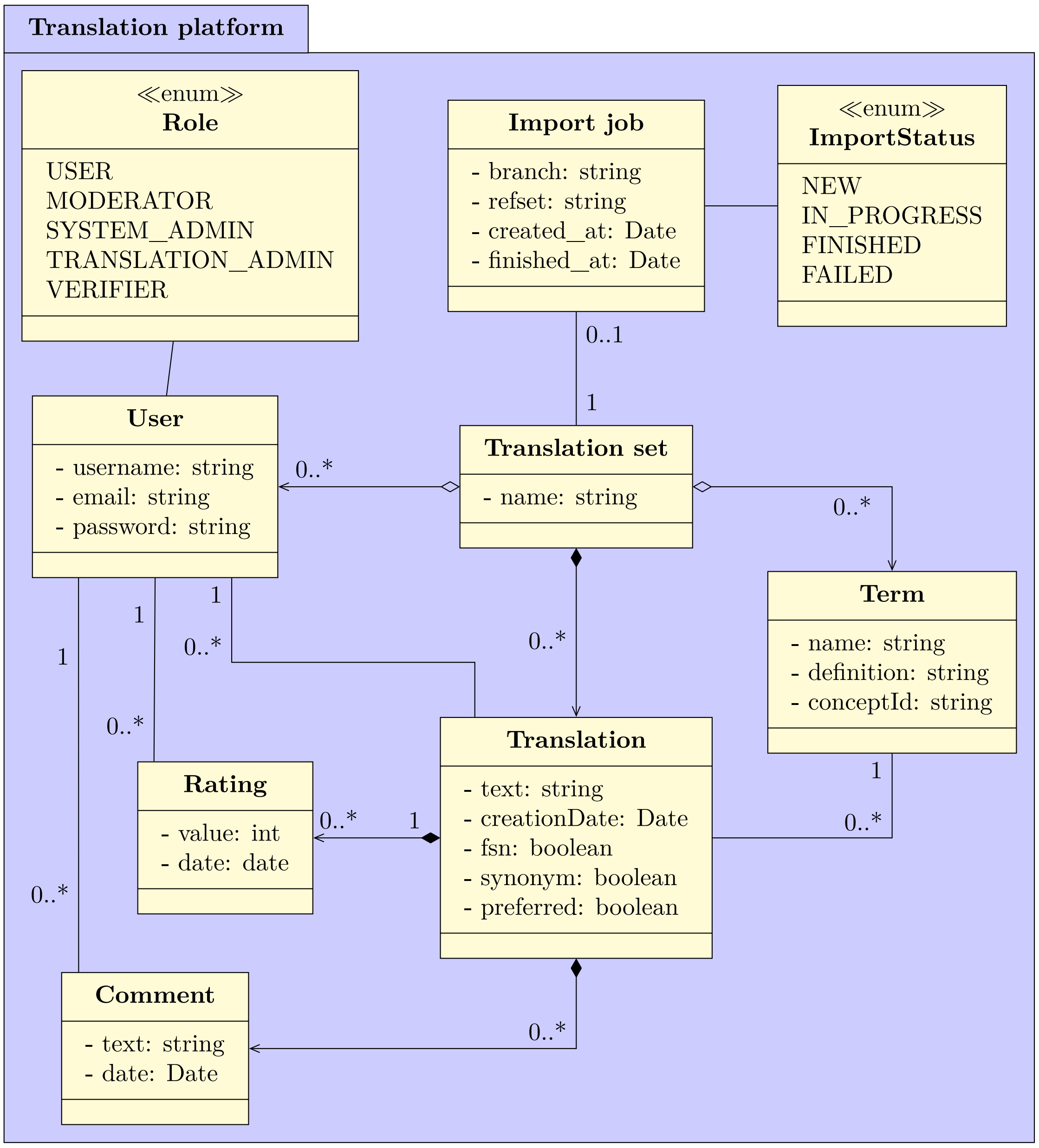


Figure 4.7: Domain model of the translation platform

4.5 States

To allow tracking of the entities as they move through the workflow, a set of states for each entity is used. In some cases, the states are linked between the entities.

The diagram in Figure 4.8 defines the possible states of a set being translated.

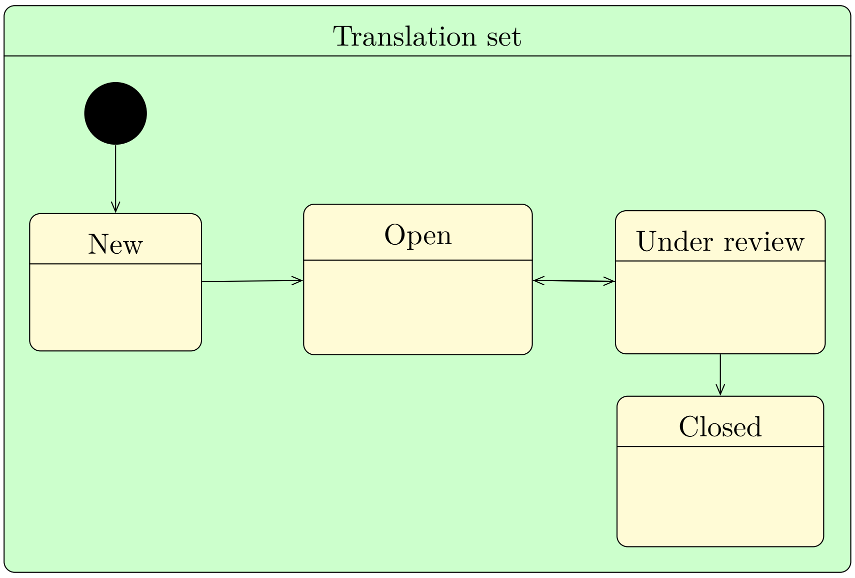


Figure 4.8: States of a translation set

The *New* state is only used while the import job is not in the *finished* state. The *Open* state allows for new comments to be added and translations suggested. While in *Under review*, translations can no longer be added, only review flags can be added. The transition between *Open* and *Under review* can be made at any time. The *Closed* state sets all the information in the translation set into read-only mode, only to be viewed.

Figure 4.9 presents the states of an Import Job. Every job starts as *New*, when the process starts it becomes *In progress*, there are two options from there: a successful completion and therefore state *Finished*, or an error resulting in the *Failed* state.

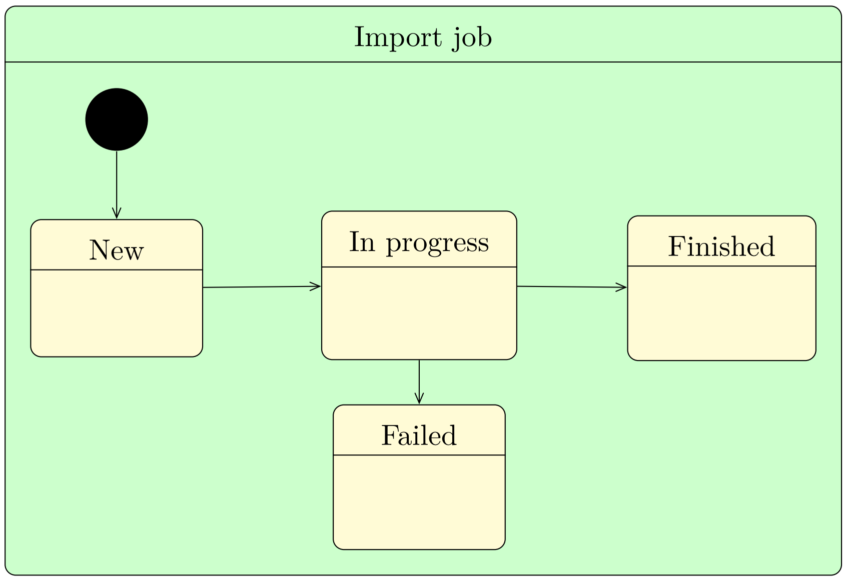


Figure 4.9: States of an import job

Chapter 5

Design

5.1 Architecture

The presentation of the architecture of the app is based on the C4 model [14]. Provides 4 levels of abstraction to separate concerns and information based on the intended audience. Starting with **System context**, it shows the big picture of the users and systems that exist within the scope of the application.

Decreasing the abstraction one level is the container diagram, depicting what the system consists of and what the technologies running those containers. It helps to show the inner working structure of a system.

Breaking down the containers are component diagrams, those present what is inside the container and describe the inner workings and control flows inside each container.

The final level down is platform dependent and is used to describe the detailed implementation of the components described one level above.

The context of the application is described in Figure 5.1.

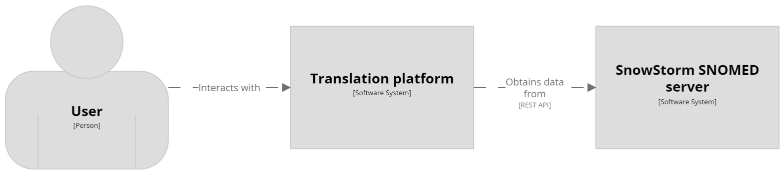


Figure 5.1: System context

As described in Figure 5.2 the platform as a whole has a multi-tier architecture. The React front end presents the data to the user and obtains user inputs. Those are then sent to the Java Spring back end, which processes the inputs and presents the data back. When needed, it communicates with MariaDB to manipulate and persist the data. Also, to allow term import, it communicates with a SnowStorm SNOMED server.

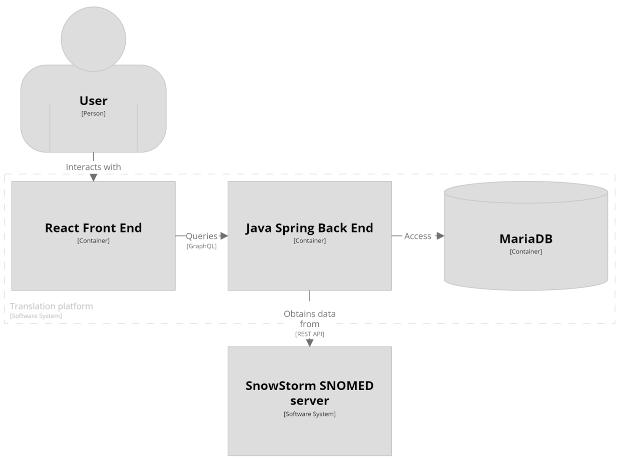


Figure 5.2: Tiers of the platform

Each user of the application falls into one of the areas of expertise, as shown in Figure 2.1. Then those areas are broken down into roles in Figure 5.3.

The translator is a member of the medical community with knowledge of the source and target language. Within the application itself, this role is referred to as *User*.

The health agency contains three roles: the moderator, who can delete defacing comments; translation manager responsible for a translation set and managing its users, in the application referred to as *Translation manager* on front end, *Translation admin* inside the code itself; verifier is responsible for checking the translation and marking it with appropriate flags.

The system administrator role is a representation of the IT professional expertise, with full access to the system.

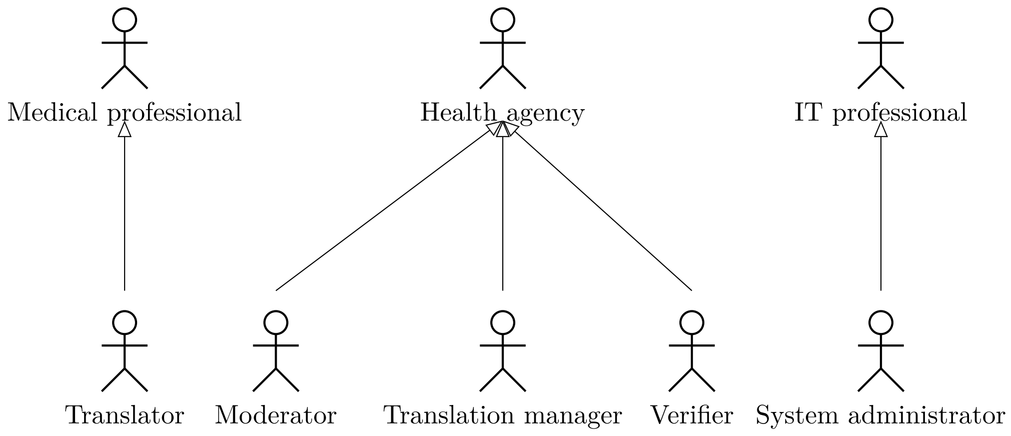


Figure 5.3: Expertise to roles

5.2 Technologies used

To implement the application based on the described architecture, four key technologies are needed: front end, API, back end, and a database. The following sections describe the technology for each part: ReactJS for the front end, GraphQL for API, Spring Boot for back end, and MariaDB for the database.

5.2.1 ReactJS

The React [15] framework is used as a base for the component-based User Interface (UI) design. To handle routing to correct pages, React Router is used. The Apollo Client [16] provider allows easy GraphQL requests. State management is handled with the included useState hooks; for states relating to the GraphQL queries, the Apollo Client handles those. Along with the states, it also keeps a cache of the results.

ReactJS was chosen based on two factors, it was suggested by the thesis supervisor and the author had the most experience with it from existing frameworks, despite that the experience is still limited.

5.2.2 GraphQL

The key component is a Schema consisting of Types. [17] The example in Listing 5.1 defines a Person with 3 attributes: name which is of type string and must be not null (denoted by !), age which is an optional integer, and addresses which is an array that will always be an array (denoted by the outer !) but could be of size 0 and all items in the array will be of type Address (denoted by the inner !).

```
type Person {
    name: String!
    age: Int
    addresses: [Address!]!
}
```

Listing 5.1: Example type

Example inspired by official documentaion. [17]

Special types are Query and Mutation. Those "*define the entry point of the GraphQL query*"[18]. A Query is used to obtain the data and Mutation is used to manipulate the data.

In traditional REST APIs it is always necessary to send a complete Data Transfer Object (DTO), therefore often a need arises to have several DTOs for the same entity based on the use. In a GraphQL query, the client can declare what exactly is needed, making implementation simpler and faster.

A Mutation uses a special type called *input* which is used to input complex data structures. Every mutation also has a return type, which is essentially the same as a query of such a type.

A GraphQL API call is made as a POST request with the body being the desired query.

```
    query {
        translationSets {
            id
            name
            status
            refsetName
            refsetId
            termCount
            translation_manager {
                id
                name
            }
        }
    }
```

Listing 5.2: Query for translation sets

The query in the previous Listing 5.2 sent using the Apollo Client results in a response in the following Listing 5.3.

```
{"data": {
        "translationSets": [
            {
                "id": 38,
                "name": "GP Test",
                "status": "OPEN",
                "refsetName": "General Practice / Family ...",
                "refsetId": "450970008",
                "termCount": 4500,
                "translation_manager": {
                    "id": 52,
                    "name": "Test Translation Admin",
                    "__typename": "User"
                },
                "__typename": "TranslationSet"
            }
        ]
}}
```

Listing 5.3: Result for query from listing 5.2

Text in refsetName shortened to fit on the page.

GraphQL API was chosen because of extensive experience working with it by the author.

5.2.3 Spring Boot

Spring Boot [19] itself provides a strong base that can be extended with the necessary functions. Spring version 3.2.4 on Java 21 built with Maven is used.

Additional components that were key in development are Lombok and MapStruct. Both allow for a smaller codebase and more streamlined development.

Lombok [20] enables a large reduction in the amount of boilerplate code. It uses annotation to provide functions that otherwise would be essentially the same in most classes.

MapStruct [21] allows for mappers not to have to be implemented, only defined as interfaces and annotated to ensure correct mappings. For simple data types it can generate the mapping, for custom types it tries to find an applicable method based on the source and target data types, for this it can also look for a method matching this signature in other classes, those are defined in the `uses` section.

The example in Listing 5.4 shows a typical use case. The `@Mapper` defines the interface as a MapStruct mapper. The `componentModel = "spring"` specifies to use the spring component model for generated code, therefore making it a Bean that can be Autowired in spring. The `uses = {ImportStatusMapper.class})` adds other classes where to look for a signature mapping, in this case it is the ImportStatusMapper class. If the name of the source and target fields is the same, they do not have to be defined, but when it is different, the `@Mapping` is used, which specifies the source and target fields.

```
@Mapper(componentModel = "spring",
        uses = {ImportStatusMapper.class})
public interface ImportJobMapper {

    @Mapping(target = "status", source = "importStatus")
    @Mapping(target = "created", source = "createdAt")
    ImportJob toDto(ImportJobEntity importJobEntity);
}
```

Listing 5.4: MapStruct interface

To facilitate the use of GraphQL in Spring, the DGS Framework created by Netflix was used. "*The framework provides an easy-to-use annotation-based programming model,...*" [22].

The cornerstone of DGS is a *datafetcher* which is similar in use to a controller in a REST based application. It handles calls to queries with the annotation `@DgsQuery`, mutations with `@DgsMutation`, and calls to a type when used as a child in another type as, for example, `@DgsData(parentType = "Translation", field = "ratings")` this being a handle for request on field `ratings` from within type `Translation`.

Another benefit provided by the DGS Framewoek is the automatic generation of DTOs used by the datafetchers from the schema.

Spring Boot was chosen based on extensive experience of the author.

5.2.4 MariaDB

MariaDB [23] was chosen mostly for convenience because the author already had an instance running and available. There were no issues encountered with the use and implementation of the data model within the scope of this thesis.

MariaDB was chosen purely based on the convenience of the author already having an instance running.

5.3 Database

The database structure is defined by the Hibernate [24] entities on the back end using the code-first approach.

The code-first approach builds the database model from the entities defined in the code of the back end application. The resulting database schema is shown in Figure 5.4.

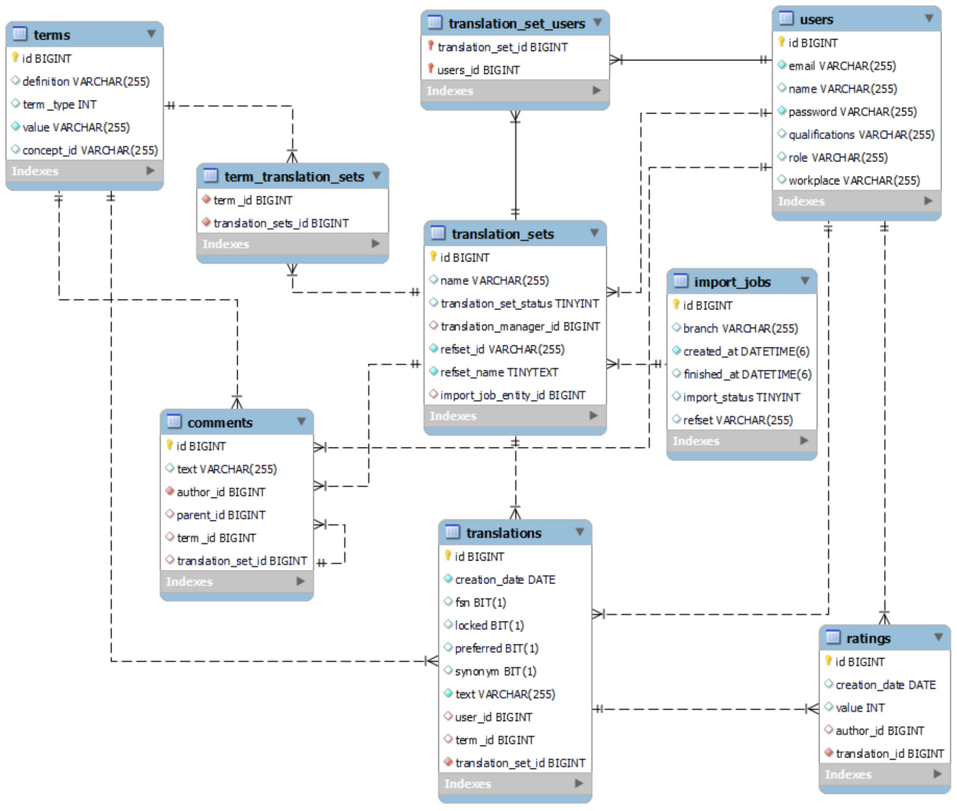


Figure 5.4: Database diagram

The code-first approach has its benefits and drawbacks. The major benefit being that the database is always matching the entities defined in the code and requires no work to keep up with changes in the database model. Despite such a benefit, it has its drawbacks.

A complication that comes with this approach is the handling of enumerations. Enumerations defined in the code do not get stored in the database and will be stored directly in the row as either the numerical representation of the enumeration or as the textual value. The textual representation makes it harder to update when the enumeration values change. On the other hand, the numerical representation makes the raw data in the database lack information.

5.4 Back end

Back end takes on a key role, especially when using the code-first approach described in the previous section. Thanks to that, it makes the database interchangeable without having to define it using a separate Data Definition Language (DDL) SQL file.

Following from the Code-first approach, the key aspect is the model package, which describes all the entities that exist within the application. For communicating with the database, a set of Spring Data repositories [25] is used. Those add a level of abstraction to allow for use of functions in a descriptive manner, removing the need to implement them, and minimising the size of the code base.

The back-end API layer uses GraphQL. The benefits of its usage and how it is implemented are discussed in Sections 5.2.2 and 5.2.3

The diagram in Figure 5.5 shows the interaction of components inside the back end. The back end entry point are the Data Fetchers which handle GraphQL queries. To be able to present the data according to the GraphQL schema, a set of Data Transfer Objects (DTO) is needed. To always keep DTOs up to date with the schema, DGS allows the DTOs to be generated from the schema. To allow for efficient mapping between the model entities and these DTOs, MapStruct interfaces are used. MapStruct uses annotations on interfaces to generate implementation at compile time.

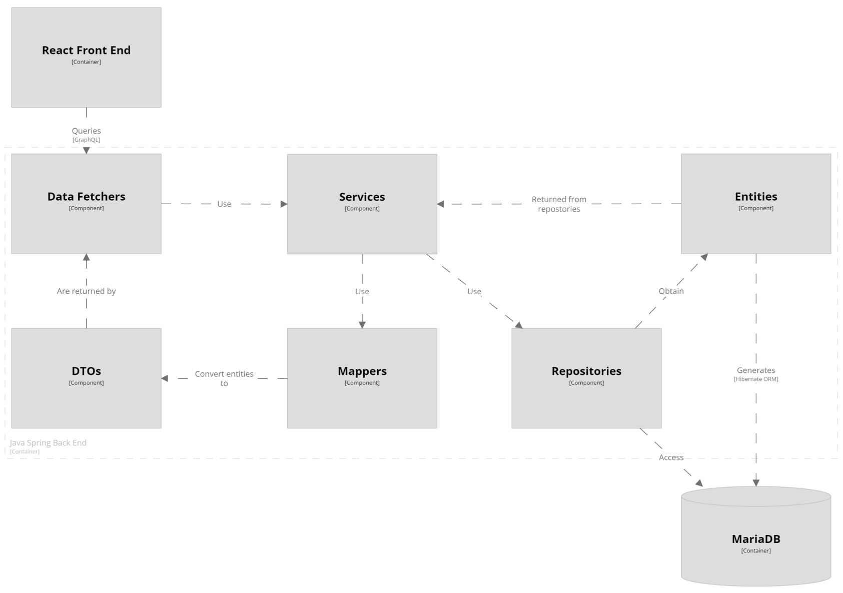


Figure 5.5: Components of the back end

The back end has 8 services, each taking care of the actions in relation to the respective entity after which the service is named.

- CommentService
- ImportJobService
- RatingService
- TermService
- TranslationService
- TranslationSetService
- UserService
- RefsetImportService

All services except for `RefsetImportService` operate synchronously. The `RefsetImportService` utilises the built-in Spring Boot `@Async` [26] annotation. This enables asynchronous execution of methods annotated with it. This service handles import of concepts in a given refset from the SnowStorm server to the database of the platform. As such import can take extended amounts of time and does not have to be complete before the creation of a translation set.

Error handling uses the Spring built-in behaviour. An abstract class `BaseException` which extends the `RuntimeException` is used to introduce two pieces of information: GraphQL `ErrorType` [27] and custom `TranslationErrorCodes` which are passed as additional information to the exception handler and passed as a response.

5.5 Front end

Front end providing the presentation of the data is built on the React framework. It is composed of a few pages that use reusable components to minimise the code base size. For components that are the foundation of all, like text boxes, buttons, check boxes, etc. Material UI's component library [28] has been used.

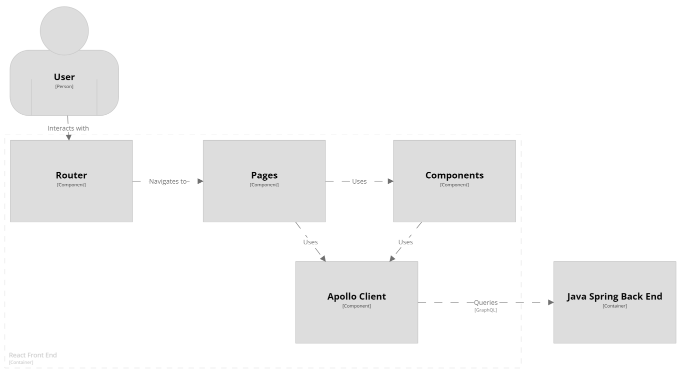
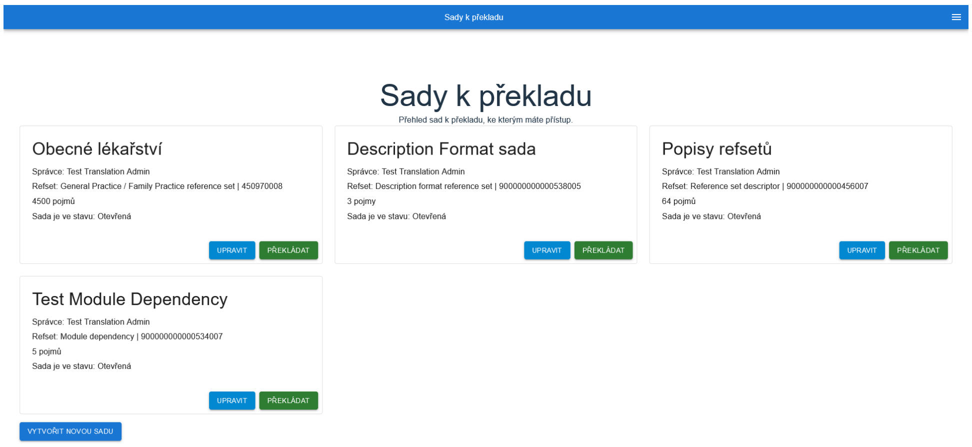


Figure 5.6: FE Components

Apollo Client [16] shown in Figure 5.6 is the key to interaction with the back end.

Figures 5.7 and 5.8 show the differences between the system administrator and a user. This overview shows all the sets that a user can take action on; for administrator, it is all the sets, for user only those they are assigned to. The administrator also has additional options for managing the set: set name, set users who can access, and change the state of the set; and also an option to make a new set.



Sady k překladu

Přehled sad k překladu, ke kterým máte přístup.

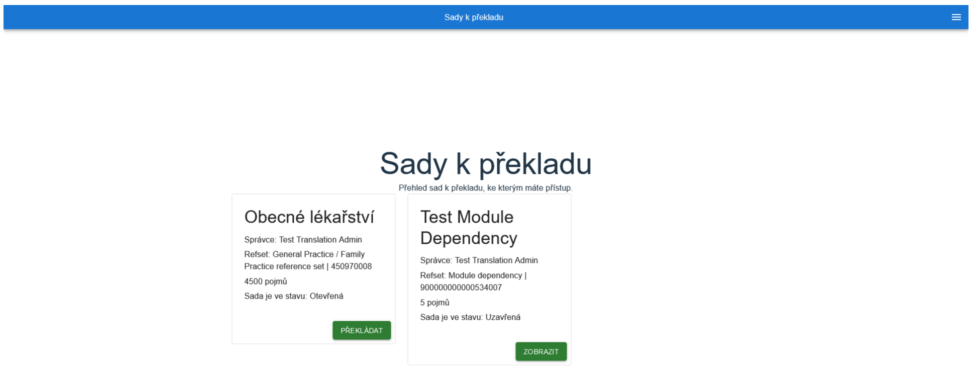
Obecné lékařství
Správce: Test Translation Admin
Refset: General Practice / Family Practice reference set | 450970008
4500 pojmů
Sada je ve stavu: Otevřená

Description Format sada
Správce: Test Translation Admin
Refset: Description format reference set | 900000000000538005
3 pojmy
Sada je ve stavu: Otevřená

Popisy refsetů
Správce: Test Translation Admin
Refset: Reference set descriptor | 900000000000456007
64 pojmů
Sada je ve stavu: Otevřená

Test Module Dependency
Správce: Test Translation Admin
Refset: Module dependency | 900000000000534007
5 pojmů
Sada je ve stavu: Otevřená

Figure 5.7: Administrator view of sets



Sady k překladu

Přehled sad k překladu, ke kterým máte přístup.

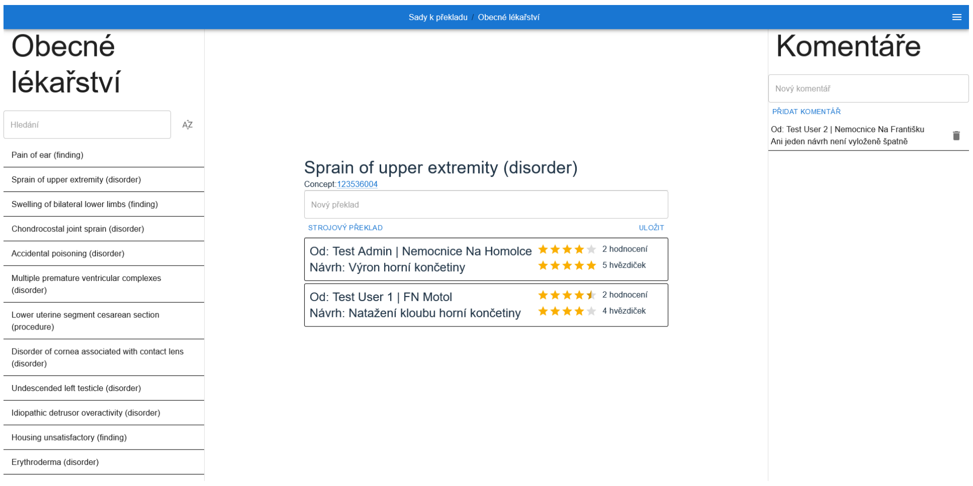
Obecné lékařství
Správce: Test Translation Admin
Refset: General Practice / Family Practice reference set | 450970008
4500 pojmů
Sada je ve stavu: Otevřená

Test Module Dependency
Správce: Test Translation Admin
Refset: Module dependency | 900000000000534007
5 pojmů
Sada je ve stavu: Uzavřená

Figure 5.8: User view of sets

The following figure 5.9 shows the view inside a set open for translation. The page consists of three key sections: left, right and middle. The left section shows a list of all terms in the set that require translations. The list of terms can be searched thru and ordered alphabetically. The right section creates a space for discussion relating the translations of the given term.

The middle section shows the textual value of the concept; below is a link to the SNOMED-CT browser to the concept to easier find the relating concepts. The text box is followed by two buttons, the left one gives the option to obtain a translation from a translation engine. The right one saves the translation; after that, it will appear in the list of translations. Each translation shows the author, their workplace, and the suggested translation. The top star scale on the right of each translation shows current rating and how many ratings are present. The lower scan gives the option of rating the translation using the stars on a scale of 1 to 5.



Sady k překladu | Obecné lékařství

Obecné lékařství

Hledání

Pain of ear (finding)
Sprain of upper extremity (disorder)
Swelling of bilateral lower limbs (finding)
Chondrocostal joint sprain (disorder)
Accidental poisoning (disorder)
Multiple premature ventricular complexes (disorder)
Lower uterine segment cesarean section (procedure)
Disorder of cornea associated with contact lens (disorder)
Undescended left testicle (disorder)
Idiopathic detrusor overactivity (disorder)
Housing unsatisfactory (finding)
Erythroderma (disorder)

Sprain of upper extremity (disorder)
Concept: 123536004

Nový překlad

STROJOVÝ PŘEKLAD ULOŽIT

Od: Test Admin | Nemocnice Na Homolce ★★★★☆ 2 hodnocení
Návrh: Výron horní končetiny ★★★★★ 5 hvězdiček

Od: Test User 1 | FN Motol ★★★★½ 2 hodnocení
Návrh: Natažení kloubu horní končetiny ★★★★☆ 4 hvězdiček

Komentáře

Nový komentář

PŘIDAT KOMENTÁŘ

Od: Test User 2 | Nemocnice Na Františku
Ani jeden návrh není vyložené špatně

Figure 5.9: View inside the set

5.6 Security

To keep track of the identities of users and allow different accesses to different users, a username password authentication system is used. Each user has a predefined role based on which authorisation takes place.

To keep passwords safe, the BCrypt [29, sec. 5] algorithm is used. To provide authority back to the front end, a JSON Web Token (JWT) [30] is used to communicate a successful login and also to send the role of the user.

Role-Based Access Control (RBAC) [31] model is used to determine the access of a given user.

Chapter 6

Testing

6.1 Methodology

The planned scope of testing consists of three levels of function tests:

- Unit tests
- Integration tests
- User testing

Unit tests cover the smallest function units in the code. Therefore, a unit test covers one function. A set of unit tests covers a whole class.

Integration tests validate the correctness of a flow from start to finish. An integration will cover the correctness of a behaviour flow intended for such a process.

User testing is a manual process that checks the accuracy when a user interacts with the system. Those tests can also provide feedback on User Experience (UX).

An additional testing scope that has been identified as useful are performance tests, specifically of the refset terms import.

To judge the usability of the system, an evaluation using the System Usability Scale [32] (SUS), can provide a rough idea of how usable the system is. For a representative score, the testers must be from the group of intended users, otherwise the score does not bring any benefit.

6.2 Realisation

Automated tests are divided between unit and integration tests. Unit tests are built using the JUnit and Mockito frameworks. On analysis of the services, since the interaction between them is minimal, integration tests would essentially cover the same as Unit tests, therefore, they were not realised, however, since the services all return mapped DTOs, they also test mappers, which are generated therefore are not really testable.

For each method that was a target of the unit tests, a test was created for a successful flow, if such flow has a branch that leads to another, but still positive outcome, another test was added to cover that branch. After the positive outcomes, all possible negative outcomes were considered and a flow to each possible negative outcome was tested to ensure the expected behaviour. Some of these tests helped to clear out incorrect exceptions used for a given negative event.

For user testing, five scenarios were designed. Each with a specified target users and steps to undertake such a scenario. For each of the tests, three metrics were collected, two open-ended questions: *"Were any actions unclear?"* and *"Was there a step too confusing?"*; and 1 metric measured on the scale of 1-10, the *"Usefulness of the scenario"*.

The usefulness rating is the basis for a quantitative rating of the test scenarios to ensure that they were designed to match the needs of the users. If the scenario is found to be useful, then the two open questions provided a qualitative analysis of the User Experience (UX). The scenarios are defined in Appendix C.

To evaluate the usability of the system itself SUS scores will be gathered using the questionnaire in Appendix D. The resulting score is calculated as follows: for questions 1,3,5,7,9 $score = scale - 1$; for questions 2,4,6,8,10 $score = 5 - scale$; then sum the score from each question and multiply by 2.5.

To measure the time taken to import terms, a timestamp is stored at the beginning and end of each ImportJob. The time taken is correlated to the amount of terms, and whether it was the first import of the refset or if the terms were already in the database.

6.3 Results

Results are presented per type of the test, for unit and performance testing the data from them are presented. For user testing a discussion about the results, implementation of remedies, and reevaluation of the cases.

6.3.1 Unit testing

Unit testing was focused exclusively on services in coverage, but even with that due to the use of entities, DTOs, mappers, exceptions, and other accompanying classes. The overall coverage is shown in the following table.

Table 6.1: Overall unit test coverage

Class	Method	Branch	Line
85.7 %	70.4 %	52.2 %	67.3 %

Table 6.1 shows that the services form a major part of the system, as no other part was targeted and the coverage is still relatively high.

Table 6.2 shows the coverage of the unit tests only inside the services package and breaks it down per service.

Table 6.2: Unit test coverage per service

Service	Method	Branch	Line
CommentService	100% (13/13)		100% (29/29)
ImportJobService	100% (5/5)		100% (27/27)
RatingService	100% (6/6)		100% (12/12)
RefsetImportService	36.4% (4/11)	0% (0/6)	10.8% (10/93)
TermService	100% (5/5)		100% (5/5)
TranslationService	100% (14/14)	100% (6/6)	100% (29/29)
TranslationSetService	100% (30/30)	100% (18/18)	100% (70/70)
UserService	100% (12/12)	100% (2/2)	100% (29/29)
Total	92.7 %	81.2 %	71.8 %

Coverage is shown in percentages, followed by the true number of covered/-total cases. Blank entries mean no branches present to test.

The breakdown of coverage in Table 6.2 shows one outlier that is `RefsetImportService`, which is due to the special nature of this service. It interacts with an external API, making it difficult to mock. The testing uncovered the need to handle cases of broken imports by user input. This resulted in modifications being made to the front end of the application. Furthermore, an error was discovered when reaching the offset of 10000, which is needed for some refsets larger than that. This limitation is part of the SnowStorm API used by the application to obtain terms in a refset with the message "*Maximum unsorted offset + page size is 10,000.*".

6.3.2 Import performance

For each refset the first run was measured and 5 runs after that. First run imports the term text into the databse, while consecutive runs only map the terms to the set.

General Practice / Family Practice reference set first run took 298.93 seconds, while the next five took on average 220 seconds with a standard deviation of 18.43; 4500 terms were imported, resulting in approximately 20.45 terms per second.

Anatomy structure and part association reference set first run took 11.45 seconds, while the next five took on average 12.34 seconds with a standard deviation of 0.48; 325 terms were imported, resulting in approximately 26.33 terms per second.

PARTIALLY EQUIVALENT TO association reference set first run took 89.75 seconds, while the next five took on average 94.17 seconds with a standard deviation of 2.89; 2225 terms were imported, resulting in approximately 23.63 terms per second.

On the largest set evaluated, the first run was the longest and all consecutive were faster, but in terms of the speed of import, it is the slowest. From the evaluated sets, we can conclude that the smaller the set, the faster the import. However, for smaller sets, the first import is the fastest.

6.3.3 User testing

The limited time frame and delayed development mean that user testing has been limited to a small group of testers, consisting of 3 people working in IT. The author gave them a quick introduction into the field of intended use cases.

For an initial round of tests each tester had 4 scenarios to complete with specified targets, after completing those a period of freeroam testing; access to the site with the target to break it, while being supervised by the author.

The scenarios that yielded implementation issues being discovered are C.5 and C.6.

Scenario C.5 identified an error in the front end implementation, when unchecked boxes were treated as undefined values, therefore breaking the store request and persisting information.

Scenario C.6 identified a major issue in user registration, while the back end uses BCrypt as described in Section 5.6, the registration process was overlooked and the password was stored directly in plain text, therefore the login broke, as the plain text password is not in BCrypt format. Correction was made to encode the password correctly.

Scenario C.4 provided feedback on the UX as it was not clear to the tester which stars are meant to give the rating. A discussion of the scenario with the tester proved that it was not clear, especially since the test account used already rated the translation. When tested again on another translation where a rating was not yet given, it was clearer to the tester.

Freeroam testing mostly revealed UI and possible UX issues. The major of which is the slow response of the UI to some user actions. Investigation of the issue uncovered no clear cause or way to remedy this.

Alongside those, an issue with the profile page was also discovered. Where the change of the email directly broke the application. The back end logs directed towards issues with authentication. Since the JWT uses email as an identifier when the email was changed, the email in the JWT is no longer found in the database and therefore cannot authenticate the user. This issue has been remedied by updating the JWT on a change of email.

An extended discussion with another tester concerning their UI/UX feedback, resulted in minor changes to the visual side to make segments of the app easier to navigate and understand. Those changes mostly consisted of clearer styling to keep consistency. A larger and function-orientated improvement was more and clear options how to cancel an action or navigate out of an action screen.

The second round of testing after improvements were made based on the feedback of the testers verified that the improvements rectified the issues the testers identified.

The SUS was not used because the testers were not the intended users, therefore the final score would not provide a reasonable result.

Chapter 7
Conclusion

The primary objective of this thesis was to build a platform that can be modified and adapted to the needs of the specific agency.

The main goal that has been achieved successfully is a complete chain from taking a refset from SNOMED-CT into the platform and allowing selected users to suggest translations, rate other translations, and have a discussion in the comments about the translations, all the way to adding review flags: if the translation is to be used as FSN, synonym and whether it is preferred.

The main focus has been on a robust back end that is extensible and usable for any agency. The front end has been built as a Proof of Concept that it can work and be used by people. However, due to the time constraints of the thesis, the moderating functions were omitted. Export functionality has been investigated, due to the complex nature of the file format to transfer data and especially the lack of documentation of the official tooling for such use, resulting in the omission of it.

The completed work is a solid foundation on which to build a working solution to translate medical taxonomies.

7.1 Testing

Despite the issue initially identified by the testers, they rated the application as usable with difficulties. After remedies were made to functional problems and a set of UI improvements implemented, the final rating was usable, but with minor issues. Those minor issues that remain are the result of slow response to user actions.

7.2 Retrospective

After completion of the work, a few issues became clear.

The code-first approach to database design is great if the application can do a full CRUD cycle for every entity; if it does not, and access directly to the database is needed for clean-up or some level of alterations, especially on enumerated fields, it adds unnecessary difficulty.

Becoming familiar with ReactJS took the author a significant time, but especially attempts to optimise and speed up the responsiveness have taken a lot of time, while yielding little to no measurable difference.

Spring Boot has again proven to be a helpful and great resource. The DGS Framework was very helpful in the completion of the work, but it also brought an issue when the behaviour of the implementation of it differs between the Gradle and Maven versions. This is most likely based on the fact that the Gradle version is official and the Maven version is community-made. As the author had experience with the Maven version but prefers the Gradle build tool, initially the application was meant to be developed using Gradle, but upon encountering the difference in behaviour, a switch was made to Maven.

The process itself brought a lot of great experience in the process of designing an application in a field the author is not overly familiar. Key feedback for future projects is to do a deeper dive prior to starting development, as it makes it easier to understand and plan for the needed functionalities and especially for edge cases.

Acronyms

CAP	College of American Pathologists. 1
DDL	Data Definition Language. 27
DTO	Data Transfer Object. 23, 25, 27, 33
EHR	Electronic health records. 1
FSN	Fully Specified Name. 7, 8, 16, 17, 37
IHTSDO	International Health Terminology Standards Development Organisation. 1, 2, 8
JWT	JSON Web Token. 31, 36
NHS	National Health Service of the United Kingdom. 1
RBAC	Role-Based Access Control. 31
SNOMED	Systemized Nomenclature of Medicine. 1, 5, 21
SNOP	Systemized Nomenclature of Pathology. 1
SUS	System Usability Scale [32]. 33, 34, 36
UI	User Interface. 8, 23, 36, 37
UX	User Experience. 33, 34, 36

Bibliography

1. *Purpose of the GP electronic health record* [online]. 2023. [visited on 2023-12-18]. Available from: `https://www.england.nhs.uk/long-read/purpose-of-the-gp-electronic-health-record/`.
2. *SNOMED CT NHS* [online]. [visited on 2024-01-03]. Available from: `https://digital.nhs.uk/services/terminology-and-classifications/snomed-ct`.
3. BENSON, Tim. *Principles of Health Interoperability HL7 and SNOMED*. 2nd. Springer-Verlag London, 2012. ISBN 978-1-4471-2800-7.
4. *Snomed Members* [online]. [visited on 2023-11-27]. Available from: `https://www.snomed.org/members`.
5. *SnowStorm* [online]. [visited on 2023-11-19]. Available from: `https://github.com/IHTSDO/snowstorm`.
6. *SNOMED-CT Wikipedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [visited on 2023-11-19]. Available from: `https://en.wikipedia.org/wiki/SNOMED_CT`.
7. *5-Step briefing* [online]. [visited on 2023-11-19]. Available from: `https://www.snomed.org/five-step-briefing`.
8. *SNOMED CT Browser* [online]. [visited on 2023-12-03]. Available from: `https://browser.ihtsdotools.org/`.
9. *Fully Specified Name* [online]. [visited on 2023-11-19]. Available from: `https://confluence.ihtsdotools.org/display/DOCGLOSS/fully+specified+name`.
10. *Guidelines for Translation of SNOMED CT* [online]. 2022. [visited on 2023-11-17]. Available from: `https://confluence.ihtsdotools.org/display/DOCTRANSLATE/Guidelines+for+Translation+of+SNOMED+CT`.
11. *Translation tools* [online]. [visited on 2024-05-20]. Available from: `https://www.implementation.snomed.org/translation-tools`.
12. *SNOMED CT Concept Editor* [online]. [visited on 2024-05-20]. Available from: `https://docs.b2ihealthcare.com/snow-owl-authoring-platform/editing-and-authoring/snomed-ct-concept-editor`.

13. *Snow Owl®Authoring Platform* [online]. [visited on 2024-05-21]. Available from: `https://b2ihealthcare.com/products/%5C#authoring`.
14. SIMON, BROWN. *The C4 model for visualising software architecture* [online]. [visited on 2024-05-13]. Available from: `https://c4model.com/`.
15. *React* [online]. [visited on 2024-05-21]. Available from: `https://react.dev/`.
16. *Introduction to Apollo Client* [online]. 2024. [visited on 2024-05-12]. Available from: `https://www.apollographql.com/docs/react/`.
17. *Object types and fields* [online]. 2024. [visited on 2024-05-09]. Available from: `https://graphql.org/learn/schema/%5C#object-types-and-fields`.
18. *The Query and Mutation types* [online]. 2024. [visited on 2024-05-09]. Available from: `https://graphql.org/learn/schema/#the-query-and-mutation-types`.
19. *Spring Boot* [online]. [visited on 2024-05-21]. Available from: `https://spring.io/projects/spring-boot`.
20. *Project Lombok* [online]. [visited on 2024-05-21]. Available from: `https://projectlombok.org/`.
21. *MapStruct* [online]. [visited on 2024-05-21]. Available from: `https://mapstruct.org/`.
22. *DGS Framework* [online]. 2020. [visited on 2024-05-04]. Available from: `https://netflix.github.io/dgs/`.
23. *MariaDB* [online]. [visited on 2024-05-21]. Available from: `https://mariadb.org/`.
24. *Hibernate ORM* [online]. [visited on 2024-05-21]. Available from: `https://hibernate.org/orm/`.
25. *Working with Spring Data Repositories* [online]. [visited on 2024-05-21]. Available from: `https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html`.
26. *The @Async annotation* [online]. [visited on 2024-05-21]. Available from: `https://docs.spring.io/spring-framework/reference/integration/scheduling.html%5C#scheduling-annotation-support-async`.
27. *Enum ErrorType* [online]. [visited on 2024-05-13]. Available from: `https://javadoc.io/doc/com.netflix.graphql.dgs/graphql-error-types/3.5.2/index.html`.
28. *Material UI* [online]. [visited on 2024-05-21]. Available from: `https://mui.com/material-ui/`.

29. PROVOS, Niels; MAZIERES, David. A future-adaptable password scheme. In: *USENIX Annual Technical Conference, FREENIX Track*. 1999, vol. 1999, pp. 81–91. Available also from: `https://www.usenix.org/legacy/events/usenix99/provos/provos.pdf`.

30. *Introduction to JSON Web Tokens* [online]. [visited on 2024-05-21]. Available from: `https://jwt.io/introduction`.

31. FERRAIOLO, David; CUGINI, Janet; KUHN, D Richard, et al. Role-based access control (RBAC): Features and motivations. In: *Proceedings of 11th annual computer security application conference*. 1995, pp. 241–48.

32. BROOKE, John. SUS – a quick and dirty usability scale. In: 1996, pp. 189–194.

33. CALISTO, Francisco Maria; NASCIMENTO, Jacinto C. *SUS Survey*. ResearchGate, 2018. Available from DOI: `10.13140/rg.2.2.25301.06883`.

34. CALISTO, Francisco Maria. *MIMBCD-UI/sus: v1.0.0-alpha*. 2018. Available from DOI: `10.5281/zenodo.1435044`.

Appendix A

Contents of the attachment

```
/
├── be ...................................... Source code for the back end
│   ├── src
│   │   ├── main
│   │   │   ├── java ............................................ Source files
│   │   │   └── resources
│   │   │       ├── application.example.yaml . Example configuration file
│   │   │       └── schema
│   │   │           └── schema.graphql
│   │   └── test ........................................ Test source files
│   ├── testReport ........................... Report of the test coverage
│   ├── Dockerfile
│   ├── pom.xml
│   └── README.md .................... Includes how to run this module
├── fe ...................................... Source code for the front end
│   ├── src
│   │   ├── modules ................................ Reusable components
│   │   ├── pages ....................................... Pages of the app
│   │   └── main.jsx .................... Entry point, router definitions
│   ├── Dockerfile
│   ├── package.json
│   └── README.md .................... Includes how to run this module
├── testing
│   ├── performance ............... Source data for performance results
│   └── scenarios
│       ├── general ........................... Scenarios as in Appendix C
│       ├── supporting..................... Scenarios as in Appendix B
│       └── templates ................ Templates to use to give to testers
│           └── used.................... Scenarios that were used for tests
└── README.md
```

Only key parts shown and described. Supporting files have been omitted in the listing of the contents.

Appendix B

Supporting scenrios for user testing

Scenarios that themselves do not bring direct benefit, but are necessary for the use of the application.

B.1 Login

Steps:

1. Open provided URL
2. Fill in the provided username
3. Fill in the provided password
4. Click on "Přihlásit se"

URL, username and password are provided to the testers directly based on the scenario to test

B.2 Logout

Steps:

1. Click on the three line icon in top right corner
2. Click on "Odhlásit se"

B.3 Update personal details

Steps:

1. Click on the three line icon in top right corner
2. Click on your name (the first option)
3. Change fields to intended values
4. Click on "Uložit změny"

Appendix C
User testing scenarios

C.1 Create translation set

Intended users:

- System Administrators
- Translation Managers

Preconditions: User is logged in and on /translation-sets page
Task: Create a translation set named *Testing Subset* with translation manager *Test Admin* and refset *General Practice / Family Practice reference set.*
Steps to complete the task:

1. Click on "Vytvořit novou sadu"
2. Fill in the name
3. Set the translation manager for the set
4. Select a refset to translate from available refsets
5. Click on "Vytvořit" to create the set

Evaluation:

- Was the task clear and understandable? *Yes/No (Please describe your issues)*
- On scale from 1 (not useful) to 5 (very useful), how would you rate the usefulness of this task?
- On scale from 1 (hard) to 5 (easy), how easy was it complete this task?

C.2 Create translation

Intended users:

- Translators
- Translation Managers

Preconditions: User is logged in and on in an open set
Task: Add a translation suggestion to any term inside the set
Steps to complete the task:

1. Select a term on the left side
2. Enter a translation suggestion in the middle
3. Click on "Uložit" to save the suggestion

Evaluation:

- Was the task clear and understandable? *Yes/No (Please describe your issues)*
- On scale from 1 (not useful) to 5 (very useful), how would you rate the usefulness of this task?
- On scale from 1 (hard) to 5 (easy), how easy was it complete this task?

C.3 Modify set metadata

Intended users:

- System Administrators
- Translation Managers

Preconditions: User is logged in and on edit site for any set
Task: Change the name of the set to *Běžné lékařství* and add any available user to the set
Steps to complete the task:

1. Modify the options required
   - Change name
   - Modify users that act as translators
2. Click on "Uložit" to save the changes

Evaluation:

- Was the task clear and understandable? *Yes/No (Please describe your issues)*
- On scale from 1 (not useful) to 5 (very useful), how would you rate the usefulness of this task?
- On scale from 1 (hard) to 5 (easy), how easy was it complete this task?

C.4 Rate translation

Intended users:

- Any user with access to the transtlation set

Preconditions: User is logged in and on in an open set
Task: Add a 2 star rating to any translation
Steps to complete the task:

1. Rate a translation using the star system

Evaluation:

- Was the task clear and understandable? *Yes/No (Please describe your issues)*
- On scale from 1 (not useful) to 5 (very useful), how would you rate the usefulness of this task?
- On scale from 1 (hard) to 5 (easy), how easy was it complete this task?

C.5 Review translation

Intended users:

- Translation manager
- System Admin
- Translation verifier

Preconditions: User is logged in and on a set which is under review
Task: Mark a translation as a FSN
Steps to complete the task:

1. Apply FSN flag to a translation
2. Click on "Potrvrdit označení"

Evaluation:

- Was the task clear and understandable? *Yes/No (Please describe your issues)*
- On scale from 1 (not useful) to 5 (very useful), how would you rate the usefulness of this task?
- On scale from 1 (hard) to 5 (easy), how easy was it complete this task?

C.6 Register as a new user

Intended users:

- Any user

Preconditions: User does not already have an account
Task: Create a user account and login
Steps to complete the task:

1. Click on "Nemáte účet? Zaregistrujte se"
2. Fill in email and password
3. Click on "Pokračovat"
4. Fill in full name, workplace and qualifications
5. Click on "Pokračovat"
6. Verify correctness of entered values
7. Click on "Zaregistrovat" to finish registering
8. Enter email and password
9. Click on "Přihlásit se"

Evaluation:

- Was the task clear and understandable? *Yes/No (Please describe your issues)*
- On scale from 1 (not useful) to 5 (very useful), how would you rate the usefulness of this task?
- On scale from 1 (hard) to 5 (easy), how easy was it complete this task?

Appendix D

System Usability Scale questionnaire

		Strongly Disagree				Strongly Agree
1.	I think that I would like to use this system frequently	1	2	3	4	5
2.	I found the system unnecessarily complex	1	2	3	4	5
3.	I thought the system was easy to use	1	2	3	4	5
4.	I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5.	I found the various functions in this system were well integrated	1	2	3	4	5
6.	I thought there was too much inconsistency in this system	1	2	3	4	5
7.	I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8.	I found the system very cumbersome to use	1	2	3	4	5
9.	I felt very confident using the system	1	2	3	4	5
10.	I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

This form has been created using parts of the source code from the MIMBCD-UI project. [33, 34]