Master Thesis

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

# CheckIt: publication tool for gestors of Semantic vocabulary of terms - frontend

Filip Kopecký

Supervisor: Ing. Michal Med, Ph.D.
Supervisor–specialist: Ing. Martin Ledvinka, Ph.D.
Field of study: Open Informatics
Subfield: Software Engineering
May 2023

# ZADÁNÍ DIPLOMOVÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

| | | | |
|---|---|---|---|
| Příjmení: | **Kopecký** | Jméno: **Filip** | Osobní číslo: **483821** |
| Fakulta/ústav: | **Fakulta elektrotechnická** | | |
| Zadávající katedra/ústav: | **Katedra počítačů** | | |
| Studijní program: | **Otevřená informatika** | | |
| Specializace: | **Softwarové inženýrství** | | |

## II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

**CheckIt: publikační nástroj pro správce Sémantického slovníku pojmů - frontend**

Název diplomové práce anglicky:

**CheckIt: publication tool for gestors of Semantic vocabulary of terms - frontend**

Pokyny pro vypracování:

1) Become familiar with existing tools of the Assembly line for creation of Semantic vocabulary of Terms (SSP), designed and created in the project Quality Open Data and Infrastructure (KODI) on the Ministry of Interior of the Czech Republic.
2) Analyze processes in the Assembly line for creation, editing and publication of semantic vocabularies.
3) Design a tool for publication of the new versions of the semantic vocabularies created or edited by the tools of Assembly line. Focus on the frontend part of the tool.
4) Implement design created in the previous point. Consider involvement of the tool into the Assembly line environment.
5) Evaluate functionality and user friendliness of a tool by the user testing (e.g. System Usability Scale or similar) for the given scenarios, focused on communication and conflict resolution.

Seznam doporučené literatury:

Křemen, P.; Nečaský, M., Improving discoverability of Open Government Data with rich metadata descriptions using Semantic Government Vocabulary, Journal of Web Semantics. 2019, 55 1-20. ISSN 1570-8268.
Křemen P., Pojmové znalostní grafy ve veřejné správě, available from:
https://data.gov.cz/%C4%8Dl%C3%A1nky/pojmov%C3%A9-znalostn%C3%AD-grafy-ve-ve%C5%99ejn%C3%A9-spr%C3%A1v%C4%9B
Křemen P.; Med M.;Nečaský M.;Domanská R., Metodika tvorby a údržby sémantického slovníku pojmů veřejné správy, 2022
Křemen P.; Med M.;Nečaský M.;Domanská R., Koncepce sémantického slovníku pojmů pro potřeby konceptuálního datového modelování agend, 2022
Fowler, M: Patterns of Enterprise Application Architecture, Addison-Wesley Professional, 2002.

Jméno a pracoviště vedoucí(ho) diplomové práce:

**Ing. Michal Med, Ph.D. katedra počítačů FEL**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **26.01.2023** Termín odevzdání diplomové práce: **26.05.2023**

Platnost zadání diplomové práce: **22.09.2024**

| | | |
|---|---|---|
| Ing. Michal Med, Ph.D.<br>podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | prof. Mgr. Petr Páta, Ph.D.<br>podpis děkana(ky) |

## III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

| | |
|---|---|
| . | |
| Datum převzetí zadání | Podpis studenta |

## Acknowledgements

I would like to thank my supervisor Ing. Michal Med, Ph.D., for the opportunity to work on this topic and for his guidance. Also, I would like to thank Ing. Martin Ledvinka, Ph.D., for his help and relevant comments. Lastly, I want to thank my mother for her continuous support during my university studies.

## Declaration

## Abstract

The motivation for this thesis was to analyze the current review process of the Assembly line, identify its shortcomings, and propose a new review process that will be available to users through a new Assembly line application. The thesis explains the technologies of the Semantic Web used by the Assembly line and describes its features and tools in depth. The review process and its execution in the Assembly line are revised, and a solution based on the identified flaws in the current implementation is proposed. The thesis then focuses on the software analysis of the new solution, starting with defining the application's scope using functional and non-functional requirements. Then the technologies used for building web applications are introduced and compared, resulting in a selection of technologies suitable for the newly defined review process and the Assembly line's technological stack. The thesis then explains how the new web application was implemented and what strategies were used to ensure a good user experience. Lastly, the implementation was tested by users to evaluate the application's usability.

**Keywords:** web application, RDF, Assembly line, SGoV, Semantic Web

**Supervisor:** Ing. Michal Med, Ph.D.

## Abstrakt

Motivací této diplomové práce byla analýza současného procesu revize změn ve Výrobní lince, identifikace jeho nedostatků a navržení nového procesu který bude uživatelům zpřístupněn skrze nově vytvořenou aplikaci ve Výrobní lince. Práce vysvětluje technologie Sémantického webu, které Výrobní linka používá a přibližuje její vlastnosti a nástroje v ní obsažené. Zabývá se rozborem současné implementace revizního procesu a navrhuje řešení, které by odstraňovalo její nedostatky. Poté se práce soustředí na softwarovou analýzu nového řešení, počínajíc definicí rozsahu aplikace skrze funkční a nefunkční požadavky. Práce posléze porovnává technologie používané pro vývoj webových aplikací. Z popsaných technologií je vybráno řešení, které nejvíce vyhovuje rozsahu aplikace a které půjde nejlépe začlenit do ekosystému Výrobní linky. Následuje popis implementace nové webové aplikace, společně s popisem technik, které byly použity ke zlepšení uživatelské přívětivosti systému. Uskutečněná implementace byla závěrem podrobena uživatelskému testování, které ověřovalo celkovou použitelnost systému.

**Klíčová slova:** webová aplikace, RDF, Výrobní linka, SSP, Semantický web

**Překlad názvu:** CheckIt: publikační nástroj pro správce Sémantického slovníku pojmů - frontend

# Contents

**1 Introduction** 1

**2 Data model description** 5

2.1 RDF 5

2.2 Triples 6

2.2.1 IRI 7

2.2.2 Literal 7

2.2.3 Blank node 8

2.3 Turtle 9

2.4 RDF Vocabularies 11

2.4.1 SKOS 12

2.4.2 DCMI 13

2.4.3 OWL 15

**3 Assembly line** 17

3.1 Overview and basic concepts 17

3.1.1 Term 18

3.1.2 Vocabulary structure 18

3.2 Tools 18

3.2.1 Mission Control 18

3.2.2 TermIt 19

3.2.3 OntoGrapher 20

3.3 Technical overview 20

3.3.1 Front-end applications 20

3.3.2 Servers 21

3.3.3 Security 21

3.3.4 Database 22

**4 Process analysis** 23

4.1 Review process 23

4.1.1 Current platform - GitHub 24

4.1.2 New platform - CheckIt 27

4.1.3 Conclusion 28

4.2 Process entities 29

4.2.1 Change 29

4.2.2 Vocabulary changes 29

4.2.3 Publication 29

4.3 Process participants 30

4.3.1 User 30

4.3.2 Vocabulary gestor 30

4.3.3 Administrator 31

4.4 Processes 32

4.4.1 Requesting a gestor role 32

4.4.2 Publication review 32

4.5 Summary 37

**5 Design of the system** 39

5.1 Functional requirements 39

5.2 Non-functional requirements 43

**6 Technical analysis** 45

6.1 Front-end library/framework 45

6.1.1 React 45

6.1.2 Angular 46

6.1.3 Vue.js . . . . . . . . . . . . . . . . . . . . 47
6.1.4 Conclusion . . . . . . . . . . . . . . . . 47
6.2 Component library . . . . . . . . . . . . 47
6.3 JavaScript Build Process . . . . . . 48
6.3.1 Transpilation . . . . . . . . . . . . . . 48
6.3.2 Bundling . . . . . . . . . . . . . . . . . . 49
6.4 Build tool . . . . . . . . . . . . . . . . . . . 49
6.4.1 Create React App . . . . . . . . . . 50
6.4.2 Vite . . . . . . . . . . . . . . . . . . . . . . 50
6.4.3 Conclusion . . . . . . . . . . . . . . . . 51
6.5 Back end . . . . . . . . . . . . . . . . . . . . 52

**7 Implementation 53**

7.1 Technological stack . . . . . . . . . . . 53
7.2 Authentication . . . . . . . . . . . . . . . 54
7.3 User experience . . . . . . . . . . . . . . 54
7.3.1 Rendering of large datasets . . 55
7.3.2 Optimistic updates . . . . . . . . . 56
7.4 Overview of implemented features 57
7.4.1 User-friendly data visualization 57
7.4.2 Communication . . . . . . . . . . . . 58
7.4.3 Defined user access . . . . . . . . . 59
7.4.4 Collaboration . . . . . . . . . . . . . . 59

**8 Testing 63**

8.1 Unit testing . . . . . . . . . . . . . . . . . . 64
8.2 Scenario testing . . . . . . . . . . . . . . 64
8.2.1 System usability scale . . . . . . 64
8.3 User testing . . . . . . . . . . . . . . . . . . 65
8.3.1 Identified shortcomings . . . . . 66

**9 Conclusion 69**

9.1 Evaluation . . . . . . . . . . . . . . . . . . . 70
9.2 Future work . . . . . . . . . . . . . . . . . . 70
9.2.1 Responsivity improvements . . 70
9.2.2 Extension of known predicate IRIs . . . . . . . . . . . . . . . . . . . . . . . . . 70
9.2.3 Editing of comments . . . . . . . 71
9.2.4 Diagram changes . . . . . . . . . . . 71
9.2.5 Full incorporation into the AL 71

**Bibliography 73**

**A Testing scenarios 79**

A.1 Logging in . . . . . . . . . . . . . . . . . . . 79
A.2 Requesting a gestor role . . . . . . . 79
A.3 Positive review . . . . . . . . . . . . . . . 79
A.4 Approving publication . . . . . . . . 80
A.5 Negative review . . . . . . . . . . . . . . 80

**B Screenshots of the application 81**

B.1 Home page . . . . . . . . . . . . . . . . . . 81
B.2 Administrator panel . . . . . . . . . . 82
B.3 Publications . . . . . . . . . . . . . . . . . 84
B.4 Others . . . . . . . . . . . . . . . . . . . . . . 86

**C Content of electronic attachment 87**

# Figures

2.1 Triple visualization using nodes connected by an arc . . . 6

2.2 Using blank node as a resource . . 8

3.1 Visualization of project separation inside the graph database . . . 19

4.1 Vocabulary manipulation workflow - current state . . . 25

4.2 Vocabulary manipulation workflow - new state . . . 28

4.3 Publication containing modifications to two vocabularies . 30

4.4 User hierarchy of the new system 31

4.5 Requesting a gestor role . . . 34

4.6 New review process overview . . . 35

4.7 Change evaluation subprocess . . 36

7.1 Different rendering approaches of large datasets . . . 55

7.2 Virtualization rendering techniques on a large dataset . . . 56

7.3 Single triple change visualization 57

7.4 Relationship visualization . . . 58

7.5 Change Turtle output . . . 58

7.6 Comment in a discussion thread 58

7.7 Unauthorized access message . . . 59

7.8 Rejection comment on a change 60

7.9 Updated change that was already approved . . . 60

7.10 Approved vocabulary in a publication . . . 60

7.11 Notifications . . . 61

B.1 Home page for non-administrators . . . 81

B.2 Home page for administrators . . 82

B.3 Administrator panel options . . . 82

B.4 Administrator panel vocabularies list . . . 82

B.5 Administrator panel pending gestoring requests . . . 83

B.6 Administrator panel pending gestoring requests expanded . . . 83

B.7 Removing gestors from a vocabulary . . . 84

B.8 Publication overview . . . 84

B.9 Review of changes . . . 85

B.10 Limited review access . . . 85

B.11 List of available vocabularies for users . . . 86

B.12 Landing page . . . 86

# Tables

4.1 Comparison of GitHub and CheckIt as review platforms . . . . . . 37

6.1 Comparison of Create React App and Vite . . . . . . . . . . . . . . . . . . . . . . . 51

8.1 Perceived difficulty of finishing the test scenarios . . . . . . . . . . . . . . . . . . . 66

8.2 SUS scores . . . . . . . . . . . . . . . . . . . 66

# Chapter 1

# Introduction

The Open Data Directive[1] mandates the release of public sector data (Open Government Data) in free and open formats. Some of the benefits of publishing Open Government Data are the possible improvements of efficiency in public administrations, improved transparency of administrative organs, or economic growth [1].

However, only publishing data in an open format may not be sufficient enough to make the data useful. The quality of such data is not guaranteed, and the interoperability with other datasets is also in jeopardy. To address this problem, the European Union provided funding for a project titled *"Developing data policies to improve the quality and interoperability of public administration data"* [2].

The project consists of five main objectives, each working towards solving a particular issue regarding open data [3]. This thesis will focus on the fifth objective, *"Increasing the interoperability of public administration information systems and datasets recorded by them."* One of the goals of this objective is to develop a platform called **Assembly line** (AL) which provides a way for creating and developing ontologies in a collaborative fashion. Despite the platform's general character - allowing it to be used to model any domain, Czech public administration domain specialists mainly utilize it to model their particular areas of interest. Therefore, the AL's primary focus is to provide domain and ontology engineers with a set of tools that would allow them to capture the desired domain entities precisely. Their work (in AL tools) creates the semantic government vocabulary (set of interlinked semantic vocabularies) that describes and connects meanings of individual administration domain entities to other domains or legislative concepts. Each public administration can manage its own semantic vocabulary, describing its area of focus and its important concepts, while AL ensures the semantic integrity of the semantic

[1]DIRECTIVE (EU) 2019/1024 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL

government vocabulary (SGoV) [4].

Each semantic vocabulary in the SGoV is made up of two parts: a glossary of terms and an ontological model. Glossary is represented as a SKOS (Simple Knowledge Organization System) [5] thesaurus of terms that describes their annotation properties (e.g., definitions and labels) and places them in a hierarchical structure. The ontological model further describes the formal relations amongst terms using Web Ontology Language (OWL) [6]. Building SGoV in this way allows for progressive harmonization of data's meaning (semantics) maintained in public administration information systems, which helps to achieve better semantic data interoperability between them [7].

The AL offers tools that allow users to modify both parts of the semantic vocabularies and subsequently publish those modifications. Users of AL modify the semantic vocabularies by using three separate applications, Mission Control[2] (vocabulary manager), TermIt[3] (glossary manager) [8], and OntoGrapher[4] (ontological conceptual modeling tool) [9]. Once users have completed all their desired vocabulary modifications, the altered vocabulary data can be published to SGoV. However, a review of proposed changes must be done before the modifications are incorporated into SGoV. The review process is the only part of the vocabulary manipulation process that is not handled by a tool developed by the AL's authors.

The AL platform uses GitHub (GH) for storing SGoV in Terse RDF Triple Language (Turtle) [10]. Each published set of changes is propagated as a pull request to the SGoV GitHub repository[5]. Users overseeing the repository must manually evaluate it and subsequently merge or reject the changes. The evaluation is done by comparing the original .ttl files (Turtle file format) with the modified versions using the GH text difference checker.

Approaching the evaluation of changes in this way is very problematic for a number of reasons.

The modified files are generated by a server that loads vocabulary data from a database and serializes them in an unstable manner into a file. That can cause a random reordering of the file's content, which would appear as changes in the text difference checker. Because of this, it is particularly challenging to distinguish between the actual modifications made by users and the arbitrary reordering brought on by serialization.

Only GH repository maintainers can approve updates to SGoV, which is the next problematic aspect. It is not maintainable long-term to rely only on repository maintainers because the number of changes is expected only to increase. Also, the project under which AL is developed has a specific

[2]`https://github.com/datagov-cz/mission-control` - Accessed: 14-Jan-2023
[3]`https://github.com/datagov-cz/termit-ui` - Accessed: 14-Jan-2023
[4]`https://github.com/datagov-cz/ontoGrapher` - Accessed: 14-Jan-2023
[5]`https://github.com/datagov-cz/ssp` - Accessed: 14-Jan-2023

deadline, after which the project's future is still being determined. That also imposes an issue because the GH repository's maintainers are the project's developers.

Therefore, the need for a solution that would allow more people to review and accept changes is apparent. However, only adding more people as maintainers of the GH repository does not solve the issue. They would need to be familiar with the GH user interface and know the Turtle language's syntax.

The AL provides a user-friendly way of manipulating the semantic vocabularies' content until the last step, reviewing newly proposed changes. The reviews can be complex and labor-intensive, requiring knowledge about the modeled domain and certain technical expertise. The need for a system and process which would allow more people to review changes in a user-friendly manner is evident. Without the implementation of such a system, the AL's review step could be a severe obstacle in the broader adaptation of the AL by the end-users.

This thesis describes the AL ecosystem and the design, development, and testing of a new system that solves the data publication review step.

# Chapter 2

# Data model description

The Semantic Web is an extension of the existing World Wide Web, enabling computers and people to work together by giving well-defined meaning to data and information [11]. The meaning provided enables computers to understand the data on the Web and analyze it in a similar way to humans. The AL utilizes Semantic Web technologies to make the data published by public agencies machine-readable, sharable, and reusable across applications. This chapter introduces some key technologies that empower the Semantic Web.

## 2.1 RDF

The Resource Description Framework (RDF) is a framework for describing information about resources [12]. The resources are anything from physical objects, people, and documents to even abstract concepts. The resources are identified by their International Resource Identifier (IRI) [13]. The main goal of RDF is to publish such data and interlink them on the Web.

To describe information about a resource, RDF uses statements that follow a simple structure containing three elements in the following order: subject, predicate, and object.

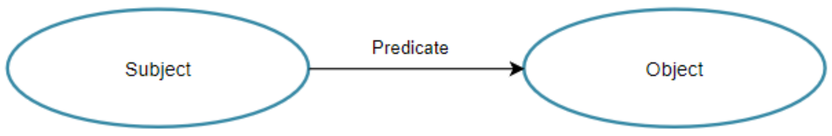


**Figure 2.1:** Triple visualization using nodes connected by an arc

The subject and object are resources that are connected by a predicate. The predicate explains the nature of that connection. In other words, it explains the relationship between these two resources.

Since statements always have three elements, they are called triples. A triple which would say that *“Sky has a blue color”* could informally look like this:

```
sky has-color blue
```

A set of triples can be visualized as a connected graph, where subjects and objects are nodes, while predicates are the arcs (figure: 2.1). The graphs can be stored in various formats, for example: N-Triples[1], Turtle[2], TriG[3], N-Quads[4], JSON-LD[5]. They differ in syntax but result in the same triples for the same graphs. Later we discuss Turtle (section: 2.3), a format that is used in the AL for storing RDF data.

## 2.2 Triples

As we already mentioned, the triples consist of three parts. Each of these parts can be a different kind of element:

- Subject: IRI or blank node
- Predicate: IRI
- Object: IRI, blank node, or literal

In the following subsections we introduce IRIs, literals and blank nodes.

[1]`https://www.w3.org/TR/n-triples/` - Accessed: 14-Jan-2023
[2]`https://www.w3.org/TR/turtle/` - Accessed: 14-Jan-2023
[3]`https://www.w3.org/TR/trig/` - Accessed: 14-Jan-2023
[4]`https://www.w3.org/TR/n-quads/` - Accessed: 14-Jan-2023
[5]`https://www.w3.org/TR/json-ld11/` - Accessed: 14-Jan-2023

### 2.2.1 IRI

Resources are identified by their IRI, which is an abbreviation for International Resource Identifier [14]. They are used as a global identifier that other people can reuse. IRIs are a special form of a Uniform Resource Identifier (URI) [15]. The main difference from URI is the ability to write resource identifiers with non-ASCII characters[6], which is not allowed in URIs, where only US-ASCII[7] characters are permitted. That is a limitation because resource identifiers are not only used by computers but by humans as well. Humans tend to use mnemonic names that might originate from their native language. Therefore, using Unicode characters in the resource identifier is convenient for international users. An example of a resource IRI from Public Sector vocabulary representing Legal Subject in SGoV looks like this:

`https://slovník.gov.cz/veřejný-sektor/pojem/subjekt-práva`

### 2.2.2 Literal

Literals allow us to add values to statements, such as strings, numbers, or dates. They are only allowed in the object part of the triple. The literal consists of two or three elements [16].

- Lexical form
- Datatype IRI
- The language tag when using language string datatype[8]

We will present some simple examples to demonstrate how different types of literals look like in Turtle (more about Turtle in: 2.3). Let us start with typed literals. We separate the lexical form from the datatype by typing ^^between those two. So, the date might look like this:

```
"1970-01-01"^^<http://www.w3.org/2001/XMLSchema#date>
```

String values have many forms of how they can be represented. Many RDF syntaxes support simple literals, which do not require the data type[9] to be stated for string values. This makes it possible to write string values as shown in the following example.

[6]Unicode/ISO 10646

[7]List of permitted characters is available at: `http://www.columbia.edu/kermit/ascii.html` - Accessed: 14-Jan-2023

[8]http://www.w3.org/1999/02/22-rdf-syntax-ns#langString

[9]http://www.w3.org/2001/XMLSchema#string

```
"Text-value"
```

Lastly, we mentioned the language tag for language strings. The syntax of writing down a language string follows the same principle as typed literal, with the difference of appending @ followed by the language tag after the type.

```
"Building"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>@en
```

However, Turtle can shorten it similarly to plain string (simple literal), omitting the data type and leaving just the lexical form and the language tag.

```
"Building"@en
```

### 2.2.3 Blank node

Although literals and IRIs are good for making statements about resources, it is sometimes convenient to talk about resources without using a global identifier (IRI). We can define resources without giving them a global identifier, and such resources still interlink to other resources. However, blank nodes cannot be used everywhere and are limited only to subjects or objects of triples [17]. We will show the possible use of blank nodes in the following examples.

The first example of using blank nodes can be the following. In an e-commerce system, users can place orders from various devices. We want to store these devices as resources in the system, but we are not able to assign a specific IRI to them. Therefore we will use blank nodes to represent them instead.

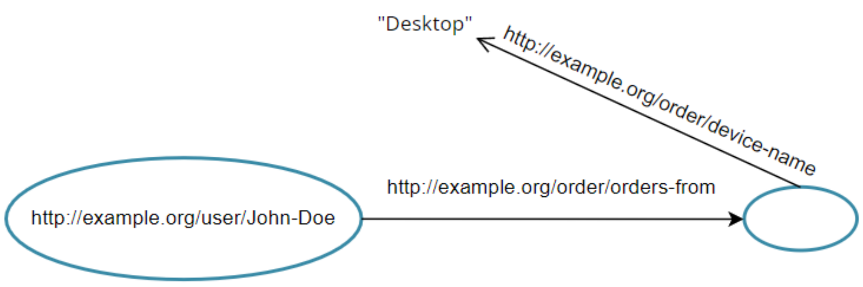

"Desktop"
http://example.org/order/device-name
http://example.org/order/orders-from
http://example.org/user/John-Doe

**Figure 2.2:** Using blank node as a resource

As a second example, let us imagine a scenario where we keep basic information about users of our imaginary system. We have several triples describing a user's age, name, and gender. We want to keep a user's home address in one triple (*user lives-at address*). Because address consists of multiple fields like state, city, street name, and ZIP code, we need to encapsulate all this information in one resource. We could create a unique resource with this combination of fields and assign a global identifier to it. However, that would result in a huge number of new globally identifiable resources which may not be needed. So we use blank nodes to encapsulate all the fields in the address without overblowing the system with irrelevant IRIs.

## 2.3 Turtle

Terse RDF Triple Language (Turtle) is a concrete syntax for RDF [10]. It is an extension of N-Triples [18] syntax which provides a line-based way of writing down graphs. However, Turtle allows for a much more compact textual view of the data.

To fully showcase the advantages of using Turtle, let us start with a small graph written in N-Triples that describes a book and its author. The N-Triples is a line-based format that uses only absolute IRIs enclosed in angle brackets. The triple's subject, predicate, and object are separated by whitespace, and a dot symbol marks the triple's end.

```
<http://example.org/book/book1>
↪  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
↪  <http://www.w3.org/2000/01/rdf-schema#Resource> .
<http://example.org/book/book1>
↪  <http://purl.org/dc/elements/1.1/title> "The Great Gatsby"
↪  .
<http://example.org/book/book1>
↪  <http://purl.org/dc/elements/1.1/title/creator>
↪  <http://example.org/author/author1> .
<http://example.org/book/book1>
↪  <http://purl.org/dc/elements/1.1/type>
↪  <http://example.org/genre/novel> .
<http://example.org/book/book1>
↪  <http://purl.org/dc/elements/1.1/type>
↪  <http://example.org/genre/tragedy> .

<http://example.org/author/author1>
↪  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
↪  <http://xmlns.com/foaf/0.1/Person> .
<http://example.org/author/author1>
↪  <http://xmlns.com/foaf/0.1/name> "F. Scott Fitzgerald" .
```

The book shown in the example is defined as a resource, with the title *"The Great Gatsby."* We also linked the book with the novel and tragedy genre and its author. The author is defined as a person with the name *"F. Scott Fitzgerald."*

The main advantage of Turtle is a more compact and readable way of writing down graphs. There are three main approaches how to achieve a shorter textual representation of RDF data.

1. Predicate lists
2. Object lists
3. Prefixes

It is pretty common that we define multiple triples concerning the same subject. Predicate lists solve this issue by appending multiple predicates and objects separated by semicolons after the object.

```
<http://example.org/author/author1>
↪  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
↪  <http://xmlns.com/foaf/0.1/Person> ;
<http://xmlns.com/foaf/0.1/name> "F. Scott Fitzgerald" .
```

It is also common to define multiple objects per one subject-predicate pair. Object lists allow objects to be written in sequence, separated by a comma.

```
<http://example.org/book/book1>
↪  <http://purl.org/dc/elements/1.1/type>
↪  <http://example.org/genre/novel> ,
↪  <http://example.org/genre/tragedy> .
```

Another way of making the triples more readable is to use prefixes. Prefixes are used to shorten long IRIs by defining prefix labels for the beginning part of an IRI. Let us consider IRI **http://example.org/genre/novel** from the previous example. If we define a prefix label for **http://example.org/genre/** as **gen**, we can refer to the IRI by **gen:novel**. When using prefixes, enclosing brackets are not used.

```
@prefix gen: <http://example.org/genre/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix bk: <http://example.org/book/> .

bk:book1 dc:type gen:tragedy .
```

The initial example can, in its shortest form, look like the following Turtle example. Please note the fact that we are using 'a' for **<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>**. It is a rule in Turtle that bounds the 'a' to that IRI [14].

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix bk: <http://example.org/book/> .
@prefix gen: <http://example.org/genre/> .
@prefix au: <http://example.org/author/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .


bk:book1 a rdf:Resource ;
    dc:title "The Great Gatsby" ;
    dc:creator au:author1 ;
    dc:type gen:novel , gen:tragedy .

au:author1 a foaf:Person ;
    foaf:name "F. Scott Fitzgerald" .
```

## 2.4 RDF Vocabularies

As mentioned in the previous section, the RDF data model allows us to make statements about resources. The model is very flexible and does not prescribe any specific meaning for resources. However, the meaning can be provided by particular vocabularies or conventions that provide semantic information about the resources [14].

Vocabularies describe certain areas of interest, classifying terms used in that specific domain, possibly explaining a relationship between them, and defining possible constraints on the usage of those terms [19].

To support the creation of such vocabularies, RDF provides RDF Schema language (RDFS), which allows the definition of the semantic characteristics of the data [20]. RDFS is not the only vocabulary that can provide semantic meaning to the data. In fact, many vocabularies have become widely used to help describe the data. Some of the relevant vocabularies for the AL are RDFS, SKOS[10], OWL[11], and DCMI[12]. In the following sections, we inspect

[10]`https://www.w3.org/TR/skos-reference/` Accessed: 20-Jan-2023
[11]`https://www.w3.org/TR/owl-features/` Accessed: 20-Jan-2023
[12]`https://www.dublincore.org/specifications/dublin-core/dcmi-terms/` Accessed: 20-Jan-2023

some of them and provide examples based on the actual SGoV data, which is why they contain the Czech language.

### 2.4.1 SKOS

SKOS, which stands for Simple Knowledge Organization System, is an RDF vocabulary developed by the World Wide Web Consortium[13] (W3C) for the representation of KOS (Knowledge Organization System) [21]. It provides a way how to create and manage KOS in a standardized manner by using concepts (identified by IRI) and creating relationships amongst them. To better illustrate the meaning behind SKOS, we will be showing individual examples of SKOS usage on the data originating from SGoV using Turtle notation.

SKOS is built around elements called concepts. These concepts can represent units of thought, ideas, meanings, or objects and events which exist in the knowledge organization system [5]. Concepts are characterized by their labels, which reference them in natural language. SKOS offers three types of labels: *skos:prefLabel*, *skos:altLabel* and *skos:hiddenLabel*. First mentioned is used to provide the preferred label for the concept, while *skos:altLabel* is used to provide alternative labels. For searching purposes, the *skos:hiddenLabel* is used. The hidden labels are usually invisible to users and are only used to help them interact with a KOS via text search functions.

```
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix sgov:
↪  <https://slovník.gov.cz/legislativní/sbírka/361/2000/pojem/>

sgov:stavba skos:prefLabel "Budova"@cs .
sgov:stavba skos:altLabel "Stavení"@cs .
sgov:stavba skos:hiddenLabel "Bduova"@cs .
```

Concepts get their meaning not only from labels and definitions (*skos:definition*) but also from their relationships to other concepts. SKOS provides two types of semantic relations: **hierarchical** and **associative**.

Hierarchical relations indicate whether a concept is in some way more general than other (*skos:broader*). Similarly, the inverse can be expressed by *skos:narrower*, indicating that the concept's meaning is more specific than the other [22].

[13]https://www.w3.org/ Accessed: 9-April-2023

```
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix sgov:
↪   <https://slovník.gov.cz/legislativní/sbírka/361/2000/pojem/>
↪   .
sgov:motorové-vozidlo skos:broader sgov:vozidlo .
```

Associative relations indicate that linked concepts are inherently somehow related, but not in a way that would indicate that one concept is more or less general. We can mention *skos:related* as an example.

Even though concepts can be created and used as stand-alone entities, they are usually grouped together in carefully compiled vocabularies (e.g., thesauri or classification schemes). These vocabularies can be created by creating resources of the *skos:ConceptScheme* class. Concepts are then linked to the concept scheme resource via *skos:inScheme*.

Following example shows concepts which describe various traffic entities. All of them are grouped in one glossary called *Slovník zákona č. 361/2000 Sb. o provozu na pozemních komunikacích a o změnách některých zákonů - glosář*

```
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix sgov:
↪   <https://slovník.gov.cz/legislativní/sbírka/361/2000/pojem/>
↪   .
sgov:motorové-vozidlo skos:inScheme
↪   <https://slovník.gov.cz/legislativní/sbírka/361/2000/glosář>
↪   .
sgov:autobus skos:inScheme
↪   <https://slovník.gov.cz/legislativní/sbírka/361/2000/glosář>
↪   .
sgov:cyklista skos:inScheme
↪   <https://slovník.gov.cz/legislativní/sbírka/361/2000/glosář>
↪   .
```

### 2.4.2 DCMI

The Dublin Core Metadata Initiative (DCMI) is an organization that supports the development and advancement of metadata design and best practices within the metadata space [23].

DCMI created a set of 15 core elements used for resource description[14]. These fifteen elements consequently became part of a larger set[15] of metadata

[14]The Dublin Core™ Metadata Element Set
[15]DCMI metadata terms

vocabularies and technical specifications also maintained by DCMI [24]. Let us list the core fifteen elements. Note that the elements are written with **dc:** prefix which stands for **http://purl.org/dc/terms/**.

- `dc:title`
- `dc:creator`
- `dc:subject`
- `dc:description`
- `dc:publisher`
- `dc:contributor`
- `dc:date`
- `dc:type`
- `dc:format`
- `dc:identifier`
- `dc:source`
- `dc:language`
- `dc:relation`
- `dc:coverage`
- `dc:rights`

These elements can be used to describe concept scheme resource. Let us describe the glossary mentioned in the SKOS section (2.4.1) with some of the core elements.

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix l-sgov-sbírka-361-2000:
↪  <https://slovník.gov.cz/legislativní/sbírka/361/2000/> .

l-sgov-sbírka-361-2000:glosář a skos:ConceptScheme;
  dcterms:created "2020-07-28";
  dcterms:title "Slovník zákona č. 361/2000 Sb. o provozu na
  ↪  pozemních komunikacích a o změnách některých zákonů -
  ↪  glosář"@cs;
```

### 2.4.3 OWL

OWL, which stands for Web Ontology Language, is a language developed by W3C for the Semantic Web. It is designed to represent complex knowledge about things, groups of things, and their relationships richly and comprehensively. OWL uses statements to capture elementary pieces of knowledge. A statement such as *"every man is mortal"* can be used as an example. A collection of such base pieces of knowledge (axioms) essentially form an ontology that asserts that these axioms are true [6].

To better illustrate how OWL can express knowledge, we will use simple statement examples based upon two people named "John" and "Alice." Statements typically refer to objects and their description. For example, they can be created by placing objects in categories (*"John is a man"*) or stating something about their relationships (*"John and Alice are married"*). All the atomic parts of the statements: objects (*"John, "Alice"*), category (*"man"*), and relations (*"married"*) are called entities. In the current version of OWL, the objects are denoted as **individuals**, categories as **classes**, and relations as **properties**.

In the following example, we create OWL classes representing men and women. We further define them as *subClasses* of class **Person** (every man or woman is also a person). Then we create our individuals, **John** and **Alice**, each assigning their gender. Then we define the object property *hasWife*. We declare that if **A** *hasWife* **B**, **A** must be a man (*rdfs:domain*) and **B** must be a woman (*rdfs:range*). Also, we add the fact that property *hasWife* can also be expressed as an inverse of property *hasHuband*. Then we make Alice John's wife. Also, we expect John and Alice to have kids in the future, so we create class **Parent**, defined as anyone connected to class **Person** via *hasChild* property.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix ex: <http://example.com/> .

ex:Person a owl:Class .
ex:Man a owl:Class ;
    rdfs:subClassOf ex:Person .
ex:Woman a owl:Class ;
    rdfs:subClassOf ex:Person .

ex:John a ex:Man .
ex:Alice a ex:Woman .

ex:hasWife a owl:ObjectProperty ;
    rdfs:domain ex:Man ;
    rdfs:range ex:Woman ;
    owl:inverseOf ex:hasHusband .

ex:John ex:hasWife ex:Alice .

ex:Parent  owl:equivalentClass  [
   a owl:Restriction ;
   owl:onProperty ex:hasChild ;
   owl:someValuesFrom ex:Person
 ] .
```

The example presented is rather simple and generic. Its sole objective is to showcase some of the OWL's capabilities. While departing from the data from SGoV-based examples, illustrating OWL in "real-life" situations may help to explain its possibilities better.

# Chapter 3

# Assembly line

This chapter details the key concepts used in AL, describes the tools offered to end users, and mentions the other technologies present in AL that make the tools work.

## 3.1 Overview and basic concepts

The AL was created to provide a set of tools that allow the collaborative creation of semantic vocabularies. These vocabularies are used to gradually harmonize the meaning of the data stored in the information systems of the Czech public administration. Together they form the SGoV, which can be perceived as a dictionary of terms used in the public sector, including their definitions, their connection to the legislation, and their semantic relations to each other. Creating SGoV allows the meaning of terms used by public administrations to be shared but also brings other benefits, which we list below.

- Harmonization of the meaning amongst data in the public agency information systems.
- Enhanced interoperability of the data in the public agency information systems.
- Improved discoverability of the open datasets
- Increased quality of descriptive metadata of the open datasets.

### 3.1.1 Term

A term is the basic building block for all the vocabularies in SGoV. It represents an entity or a concept present in the domain that is captured by the vocabulary. The meaning of a term is defined by its label, definition, synonyms, and relationships to other terms, to name a few. Let us consider the concept of a *building* as an example. According to the Energy Management Act of the Czech Republic, a *building* is defined as a heating-equipped structure that is both above and below ground. The Land Registry Act defines a *building* as a construction above ground with solid foundations. We can see that each domain understands the concept of building differently. We can solve this issue by creating different semantic vocabularies (for each domain) and placing precisely defined terms into them. That allows the terms to coexist in the SGoV and preserve their distinct meaning.

### 3.1.2 Vocabulary structure

Each semantic vocabulary inside of SGoV is split into two parts - glossary and conceptual model.

Glossary is a list of terms present in the vocabulary, with their label, definition, and hierarchical relations to other terms. The glossaries are made by domain experts who utilize the SKOS organizational scheme to express this knowledge about terms.

The conceptual model expands the vocabulary further by adding semantic relations between terms defined in the glossaries. The relationships and the definition of their constraints and specifics allow ontology engineers to capture the semantics of the domain more precisely. AL internally uses OWL to represent the expressive conceptual model.

## 3.2 Tools

The tools that AL users have at their disposal are examined in more detail in the following sections.

### 3.2.1 Mission Control

Mission control is considered to be an entry point into the whole AL. It is a tool that allows users to manage their vocabulary modifications. Users create projects to which they import vocabularies they would like to modify. Projects are not limited only to already existing vocabularies. New vocabularies can be created and imported into the projects as well. Projects allow for the collaboration of multiple users on the same series of changes. Separating the work into projects also ensures that changes of one user do not interfere with changes of another user (in case they do not work on the same project). The separation is achieved by creating a new copy of the vocabulary inside the project.

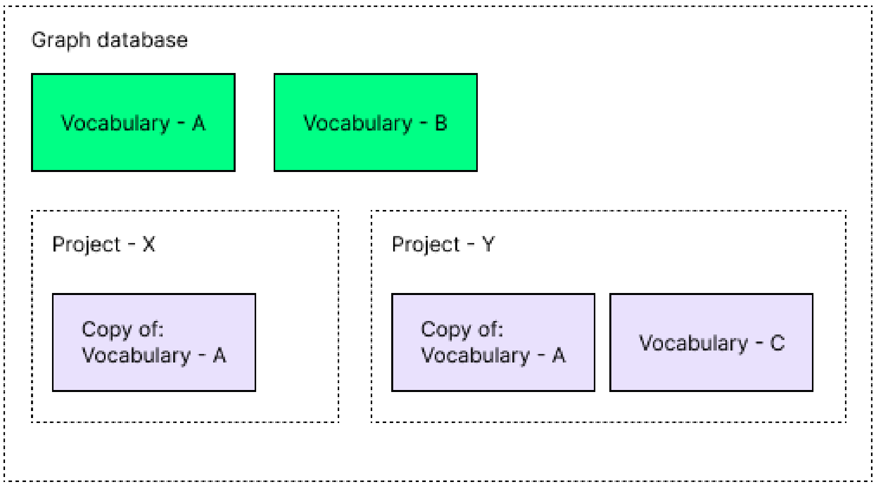


**Figure 3.1:** Visualization of project separation inside the graph database

The project's contributors can publish the project to the SGoV once they have made all the desired modifications. A pull request containing all the changes from the project is created in the GH SGoV repository, where the repository maintainers review the changes.

### 3.2.2 TermIt

TermIt is a glossary editor with an intuitive user interface that lets users add new SKOS concepts (terms) and modify or remove already-existing ones. In TermIt, users can modify the terms' labels, definitions, sources of definitions, synonyms, search strings, notations, scope notes, and text examples.

TermIt allows users to express both hierarchical and associative SKOS relationships between terms. The hierarchical structure of SKOS is the result of defining the parent terms (*skos:broader*) of terms.

The associative relationships are used in the following ways. Suppose a term has the exact same meaning as a term from a different vocabulary. In that case, it is possible to record this relation between them (*skos:exactMatch*) by connecting them via the exact match property. However, this property should be used cautiously due to the transitive nature of the property. If term **A** is the exact match of term **B** and term **B** is the exact match of term **C**, then term **A** is the exact match of term **C**. If terms do not share their exact meaning but are somehow related, they can be connected by utilizing the related property (*skos:related*) of terms.

Finally, TermIt also allows vocabulary modifications, such as changing the label (*dc:title*) or definition of a vocabulary (*dc:description*).

### 3.2.3 OntoGrapher

OG is a conceptual modeling tool that allows users to modify the ontological model part of a vocabulary. The fundamental benefit of OG is its capacity to accept and generate machine-readable outputs in the form of SKOS thesauri and OWL ontologies without the need for the users to understand any of these standards. Users create diagrams in which they visually model the semantic relationships between terms defined in the glossary. OG allows users to model the cardinalities of a relationship, its labels, and synonyms. Also, the modification of the term's intrinsic tropes, stereotypes, labels, and synonyms is possible. Regarding stereotypes, users can define up to two stereotypes, although at least one is needed for proper model validation (depending on the stereotype). First are type stereotypes that define the ontological nature of a term. We can mention Object Type and Event Type as examples. Second are the OntoUML stereotypes [25] used for validation of the model. We can mention Kind, Role, Phase, and Category as examples. Users, if they desire, can also see a list of all relationships in which the term is involved. All of these features allow for the creation of complex models that are also automatically validated for their consistency during modeling.

## 3.3 Technical overview

Now that the outline of how each primary AL tool works has been established, let us take a closer look at the rest of the AL architecture.

### 3.3.1 Front-end applications

All the AL tools with which users interact (TermIt, OG, MC) are web applications built with React[1]. They serve the roles of clients in a client-server architecture. The client-server architecture is a computing model in which a server provides clients with resources and services. The architecture follows a request-response messaging pattern where a client sends a request,

[1]`https://react.dev/` Accessed: 19-April-2023

to which the server returns a response [26]. The servers that are present in the AL are described in the next section.

### 3.3.2 Servers

The AL contains several servers that perform different tasks required for the AL to function.

- SGoV server[2]
  - Provides a common Application Programming Interface (API) for creating new vocabularies, managing projects, publishing changes, and mirroring the data between GitHub and the deployed databases.
- TermIt server[3]
  - Provides API for all actions that are TermIt related (e.g., creation and modification of terms and modifications of vocabularies).
- SGoV validator[4]
  - Checks the consistency and compliance of glossaries and models according to a predefined set of rules.
- Authentication server
  - More in section 3.3.3.

### 3.3.3 Security

Since AL consists of several tools used by numerous users, centralized user administration and authentication solution is employed. The AL utilizes Keycloak[5] as an administration, authentication, and authorization solution. The AL tools communicate with the authentication server using the OpenID Connect protocol[6] (OIDC) that allows third-party applications to verify the identity of the end user and to obtain basic user profile information. Keycloak offers a Single-Sign-On (SSO) service that allows users to use the same credentials across multiple applications. Since AL consists of many tools, the use of such an authentication mechanism is practical for the AL users. Once users log into one AL tool, they are not required to re-login to access a different one.

[2]`https://github.com/datagov-cz/sgov` Accessed: 2-May-2023
[3]`https://github.com/datagov-cz/termit` Accessed: 2-May-2023
[4]`https://github.com/datagov-cz/sgov-validator` Accessed: 2-May-2023
[5]`https://www.keycloak.org/` Accessed: 2-May-2023
[6]`https://openid.net/connect/` Accessed: 2-May-2023

### 3.3.4 Database

The AL uses GraphDB[7] and Virtuoso[8] as RDF triplestores, each having its own deployed instance in the AL. The Virtuoso instance is a public database that stores the approved version of SGoV. The GraphDB instance is a copy of the public database where users make changes.

[7] `https://graphdb.ontotext.com/documentation/10.2/` Accessed: 2-May-2023
[8] `https://virtuoso.openlinksw.com/` Accessed: 2-May-2023

# Chapter 4

# Process analysis

In this chapter, we focus on the current state of the AL from a user perspective, the already defined processes, and the resulting shortcomings of the current implementation. We define a new set of processes on top of which a new system called CheckIt was developed. A system that allows reviewing and processing the latest increments to SGoV.

## 4.1 Review process

The review process is essential to the vocabulary (ontology) manipulation workflow. It ensures that changes applied to vocabularies are adequate and correct. The changes must align with the modified vocabulary's overall structure and conventions. Without such revision, logical inconsistencies might be introduced into the ontology, impacting the ontology's usability and comprehensibility. Therefore, any changes that violate the vocabulary's structure or standards must be identified and fixed.

A platform that supports such a review process must meet specific criteria in order to ensure the process's efficacy and accuracy. We identified a couple of key requirements for the platform, which are listed below, alongside a brief rationale for each requirement.

- User-friendly data visualization
    - Ontologies can get fairly complex, making it challenging for users unfamiliar with the Semantic Web technologies to understand the meaning of data. User-friendly data visualization that simplifies the underlying ontology structure is crucial for easier identification of changes.

- Communication
  - The platform should provide a tool allowing reviewers to communicate their feedback regarding the changes clearly and effectively, so the author of changes can make necessary modifications.
  - When a reviewer is unsure about the meaning of a change, the platform should provide them with a means to communicate with the change's author to clarify any ambiguities.
- Collaboration
  - The platform should allow multiple reviewers to collaborate on the review process. Allowing multiple reviewers to participate in a review increases the efficiency of the whole process.
- Defined user access
  - Only qualified users should have access to a specific portion of changes in a review process. Defining a set of users who can access specific parts of the review is essential for ensuring that reviewers have the expertise and knowledge needed to identify issues in the proposed changes. Those reviewers are eligible to make informed decisions regarding the approval of changes and can provide valuable feedback to the creators of changes.

In the following section, we inspect how GH, the currently used platform for the review process in the AL, fulfills the mentioned requirements.

### 4.1.1 Current platform - GitHub

The AL uses the GH platform for two reasons: persisting the SGoV and reviewing the modifications to SGoV. The review process that is present in the current version of AL was defined around the capabilities of the platform rather than the actual review process needs.

The current review process works in the following manner. After users publish their modified vocabularies, a pull request in GH is created. Then it is up to the repository maintainers to check each changed line of the resulting .ttl file and decide whether such changes make sense to the domain. If they do, the pull request (PR) is approved, and changes become incorporated into SGoV. If not, the pull request is rejected, and the reviewer may send an email to the author of the changes explaining why.

We can already see that the current review process is not optimal. However, let us take a closer look at the GH platform to see why using it in the AL review process is suboptimal.

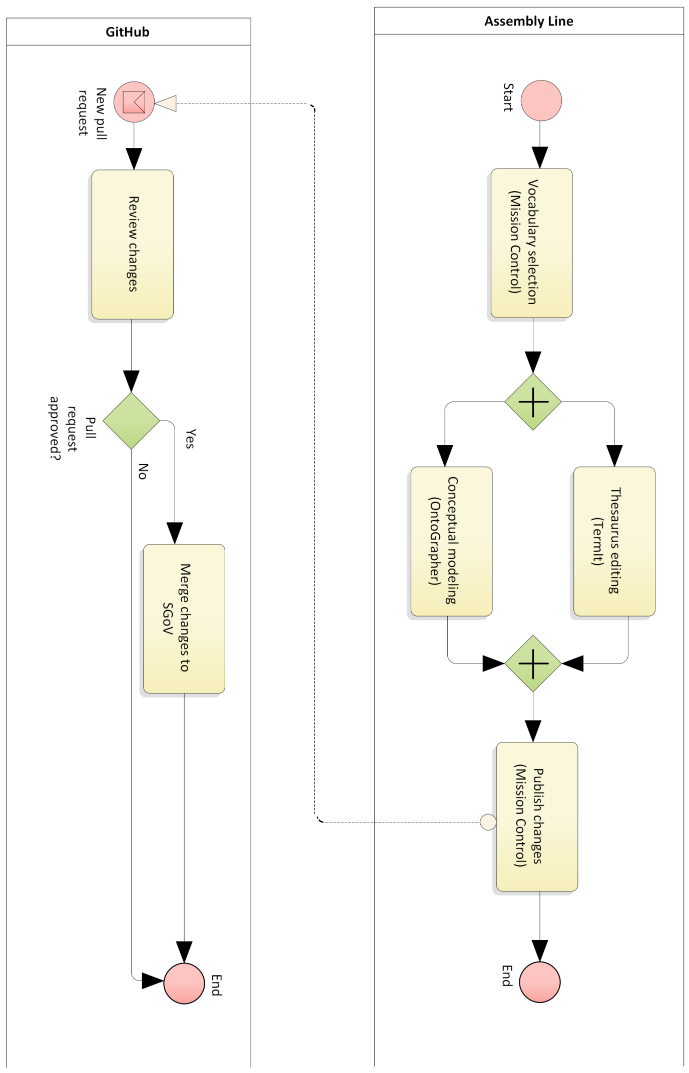

GitHub
New pull request
Review changes
Pull request approved?
Yes
No
Merge changes to SGoV
End
Assembly Line
Start
Vocabulary selection (Mission Control)
Conceptual modeling (OntoGrapher)
Thesaurus editing (TermIt)
Publish changes (Mission Control)
End

**Figure 4.1:** Vocabulary manipulation workflow - current state

The first issue comes from the different means of authentication used by GH and AL. GH uses its own authentication mechanism that is not in any way connected to the one of AL. So if AL users want to communicate with the reviewers directly on the pull requests, they need to have a GH account.

Despite the need for multiple accounts management, there is still the problem of understanding the meaning and syntax of published data. Users of AL do not directly modify the triples; instead, they are provided with a user interface that does not require them to know Turtle or any of the used RDF vocabularies (e.g., SKOS) to manipulate the RDF data. Consider changing the term definition as an example. Users only change the field *term definition* (in TermIt), possibly not knowing that the resulting triple in Turtle might look like the following example.

```
<IRI> skos:definition "term definition"@en .
```

However, the situation worsens when the changed data comes from OG. The amount of changed triples quickly gets into tens. The triples may contain blank nodes and may be scattered all over the final .ttl file due to multiple IRIs being present in the relationships.

Another problematic aspect regarding the .ttl file is the possible random reorganization of its content. Whenever a new pull request is created, the SGoV server loads the changed vocabulary from the database and serializes it into a file. However, this serialization is unstable, making it possible for random reordering of the file to happen. The GH interface would show such reordering as deletion of some triples in one section of the file and their creation in another. For a reviewer, it means remembering which triples were just reorganized and which were actually modified/created.

Showing pure triples (Turtle syntax) creates a disparity between what users of AL are used to seeing and what is presented in GH. TermIt encapsulates hierarchy and properties under a singular link or text value. OG shows the relationships visually, making it easy to understand for users familiar with UML. The anticipated end-users of OG are ontology engineers from whom an understanding of UML is expected.

The collaborative aspect among reviewers is limited due to the fact that the PR review is bound to only one GH account. Meaning one reviewer cannot approve one part of the pull request while others would approve the rest.

Lastly, there is no way how to divide access further granularly over the GH repository. Meaning GH does not allow granting write access to only some subset of files in the repository. That is problematic since anyone with write access can change the entire content of SGoV. Therefore, the write access should be given only to the administrators of the entire AL.

To summarize, following shortcoming of the AL review process were identified.

- Communication with the reviewer (need of an GH account).
- Required knowledge of the semantic web (e.g., meaning of SKOS, Turtle structure).
- Data visualisation disparity.
- Unclear distinctions between real changes and changes caused by random reordering.
- Inability to grant write access only to some vocabularies in SGoV.
- Review is bound to only one approving user (limited possibility of collaboration on a review).

### 4.1.2 New platform - CheckIt

The identified issues of the current implementation should be resolved by creating a new tool called CheckIt which would replace the current GH user interface used for the review process. Therefore, GH would become irrelevant to the end users of the AL. It still would be present in the AL technical workflow (for persisting the SGoV), but users would not have to interact with it in order to make changes to the SGoV repository.

The new solution should provide users with an intuitive interface in which they can compare and assess modifications applied to SGoV. The changes should be displayed in an AL-like manner, replacing the underlying triple data representation with a property-value format or with a visual graph representation (nodes connected by arcs).

Displaying the changes in an AL-like manner allows more people to understand the meaning of changes and improves the communication efficiency between reviewers and creators of changes.

The new platform should display only the actual changes made by AL users, not the ones caused by the unstable serialization. By showing only the real changes, reviewers can evaluate publications more effectively.

Making the platform part of the AL also enables all users of the AL to be involved in the review process. They can access the platform under the same credentials they use for any other tool present in the AL. That means that any AL user can be given the right to review changes to any SGoV vocabulary. That makes the review a collaborative activity, not bound to only one reviewer per publication, further improving the process efficacy.

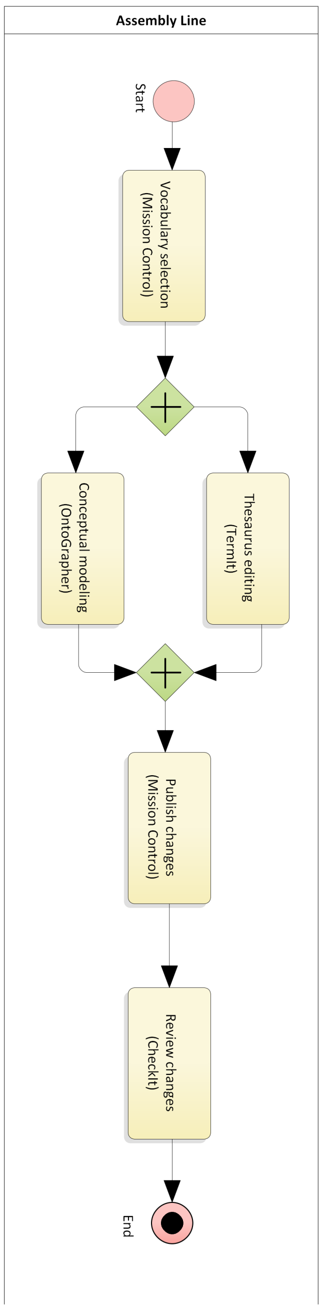

Assembly Line
Start
Vocabulary selection (Mission Control)
Conceptual modeling (OntoGrapher)
Thesaurus editing (TermIt)
Publish changes (Mission Control)
Review changes (CheckIt)
End

**Figure 4.2:** Vocabulary manipulation workflow - new state

### 4.1.3 Conclusion

It is evident that the currently deployed solution using GH as a tool for comparing the vocabulary changes has many flaws. Defining a new review process in the AL and integrating a new tool built around that process into the AL would not only expand the AL's capabilities but could increase the overall approachability of the platform by end-users.

## 4.2 Process entities

In order to better illustrate the new review process, it is necessary to define the three entities that are essential to it (figure: 4.3).

### 4.2.1 Change

Change is the basic building block of the new review process. It usually represents a single changed, created, or deleted triple. For example, changing a term's label or definition is considered a change that must be reviewed. The changes usually originate from statements that users create while interacting with the AL tools. Most of the AL tools' operations result in a single statement being created or modified. However, that is not true for OntoGrapher, where users can create custom relationships between terms. These relationships are usually expressed by multiple statements. We encapsulate all of those statements into a single change. For example, the relationship *"person drives a car"* is considered a singular change in the new review process, despite being made from multiple triples. The increased number of triples is caused by the expressiveness of OWL, which allows specifying the cardinalities of the relationships and other related attributes.

### 4.2.2 Vocabulary changes

Changes that affect the same vocabulary are grouped together to form a logical unit. In the new review process, we want the vocabularies to have a determined set of users who can review the changes applied to them. Therefore having changes grouped by the vocabulary they affect allows for a more straightforward distinction of whether a user is eligible to review such changes or not.

### 4.2.3 Publication

When users publish their changes, a publication is created, which starts off the new review process. Publications consist of one or more vocabulary changes, each containing changes to their respective vocabulary. That makes a publication a wrapper entity for all the other mentioned entities. Publications are considered indivisible, meaning it is impossible to persist only certain parts of a publication. Therefore only the entire publication might be accepted and persisted into SGoV or nothing.

To account for the collaborative aspect of the review process, we define a publication as an entity that can change over time. For example, if a reviewer demands some modifications to the vocabulary changes, users can update those changes without causing the reviewers to lose their overall progress on the review.

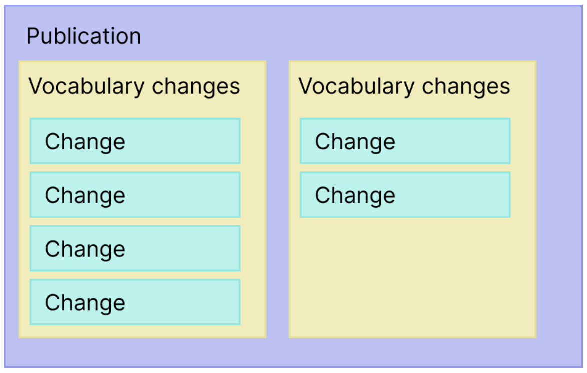


**Figure 4.3:** Publication containing modifications to two vocabularies

## 4.3 Process participants

To overcome the current implementation's shortcomings and support newly created processes, we need to define three additional hierarchical roles for the AL. The hierarchy ensures that roles at higher levels are given all the privileges of roles at lower levels.

### 4.3.1 User

By default, all users of the CheckIt tool have the *user* role. This role grants them read access and comment access to all published changes.

### 4.3.2 Vocabulary gestor

We define the gestor's role as an overseeing authority over a particular vocabulary. Every *gestor* of a specific vocabulary is responsible for the changes applied to it and therefore decides which changes or new data will be incorporated into the persisted vocabulary.

Any user can become a *gestor* of one or multiple vocabularies. It is the administrator's job to decide whether a user is eligible to become a gestor of a vocabulary or not.

### 4.3.3 Administrator

Administrators are the users with the highest possible privileges in the new system. They possess the gestor rights to all vocabularies in SGoV.

Their main objective is to manage who has the gestor rights to which vocabularies. Meaning they can assign or remove the vocabulary-gestor role to or from any user. Administrators can also elevate other users to the administrator level to make the system more independent, eliminating the possible lack of administrators in case of work indisposition or vacation. It is expected that the current set of SGoV repository maintainers will be assigned the role of administrators of the new system.

The hierarchy of the users, along with their allowed actions, is shown in the following diagram (figure: 4.4).

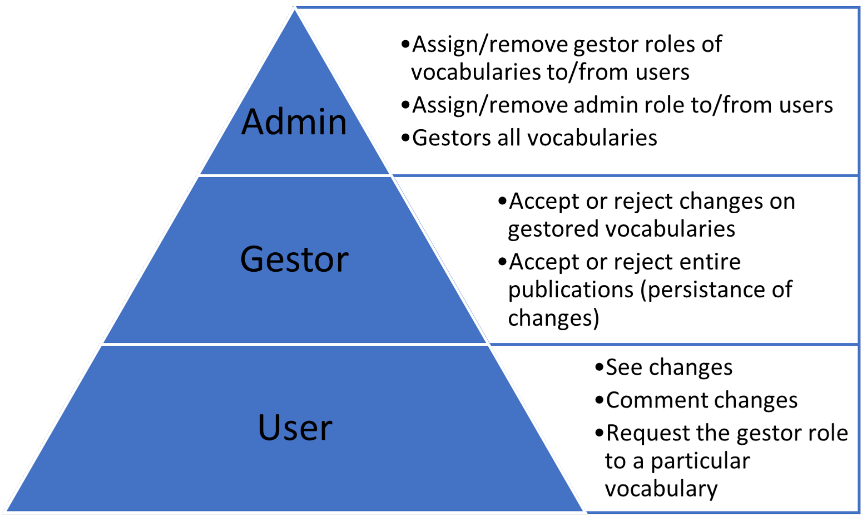

Admin
•Assign/remove gestor roles of vocabularies to/from users
•Assign/remove admin role to/from users
•Gestors all vocabularies
Gestor
•Accept or reject changes on gestored vocabularies
•Accept or reject entire publications (persistance of changes)
User
•See changes
•Comment changes
•Request the gestor role to a particular vocabulary

**Figure 4.4:** User hierarchy of the new system

## 4.4 Processes

This section presents the new processes needed for the new AL review workflow. The new processes are tightly bound to the roles we introduced in the previous section.

### 4.4.1 Requesting a gestor role

Anyone who wants to become a gestor of a particular vocabulary can do so by creating a *gestoring request*, which is then evaluated by any of the administrators of CheckIt. Any vocabulary that already exists in SGoV can have gestoring requests made for it.

Administrators should decide whether a particular user is eligible to become a gestor of requested vocabulary by knowing the requested vocabulary and the user. Such knowledge is expected from an administrator of the system.

New vocabularies (not persisted in SGoV yet) are gestored only by the administrators, and none can request to become their gestor until they are persisted in SGoV. The rationale for this restriction is that administrators are in charge of the entire SGoV content and should know which vocabularies are in the system.

The next figure (4.5) depicts the requesting gestor role process in a diagram.

### 4.4.2 Publication review

The review process starts with the creation of a publication. Publications are created when authors of changes decide that their changes are complete and ready for assessment from gestors.

Based on the publication's content, relevant gestors are notified about a new publication that requires their expertise needed for a proper evaluation of the changes.

Publications contain vocabulary changes that can be reviewed only by users who have the gestor right over a vocabulary that the changes affect. Users that do not have gestor control over the modified vocabulary can only see the changes to that vocabulary with the option to provide some insight regarding the meaning of the change (clarification to the gestor) in an attached comment thread.

The review of each change can have two possible outcomes. First is the approval of the change. Change should be approved if it does not cause logical inconsistencies in the vocabulary and complies with its structure.

If the meaning of the change is unclear to a gestor or some modifications are needed for the change to be approvable, discussion over the change needs to happen. Gestors should ask questions when something is unclear to them, and anyone (preferably the change's author) should respond.

The discussion should result in a state where the gestor has enough information to decide whether the change can be approved. If the change in its current state cannot be approved, the gestor must explain to the change's author what needs to be modified in order for the change to be accepted. The author can respond with follow-up questions if necessary. The gestor rejects the change in its current (non-approvable) state and writes a rejection comment summarising the requested modifications.

When a single gestor approves all changes that affect a particular vocabulary (vocabulary changes), then the vocabulary in the publication is considered to have an approving review. If a publication contains at least one approving review per each affected vocabulary, the changes in the publication can be integrated into SGoV.

Any gestor of an involved vocabulary in the publication can confirm the merge of the publication into the SGoV repository (approve publication), which ends the review process.

However, having an approving review associated with each vocabulary present in the publication does not guarantee the merge into the SGoV repository. Gestors might require some additional changes to be added to the publication. In that case, they reject the publication and provide a rejection message which contains a summary of what changes need to be added.

If gestors do not require any additional changes to be added, they approve the publication, and the changes get persisted into the SGoV. Hence the changes are applied to their respective vocabularies and become a part of the publicly available SGoV repository.

To better showcase the new review process, we present two diagrams. First (4.6) is the overview of the whole process. It demonstrates the collaboration between the gestor and the author of the publication. The second diagram (4.7) is a close-up showing the individual change review process.

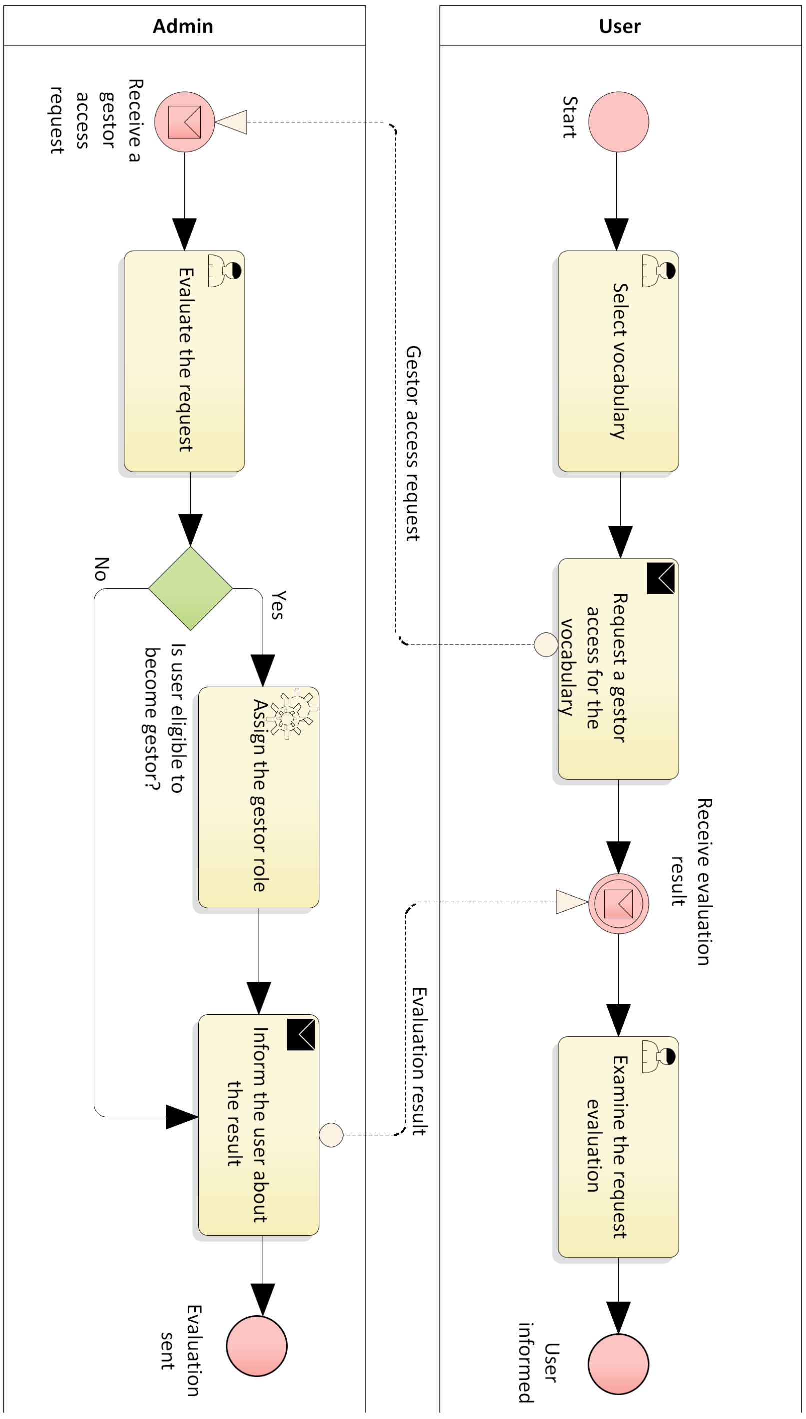

Admin
User
Start
Select vocabulary
Request a gestor access for the vocabulary
Gestor access request
Receive a gestor access request
Evaluate the request
Is user eligible to become gestor?
Yes
No
Assign the gestor role
Inform the user about the result
Evaluation result
Evaluation sent
Receive evaluation result
Examine the request evaluation
User informed

Figure 4.5: Requesting a gestor role

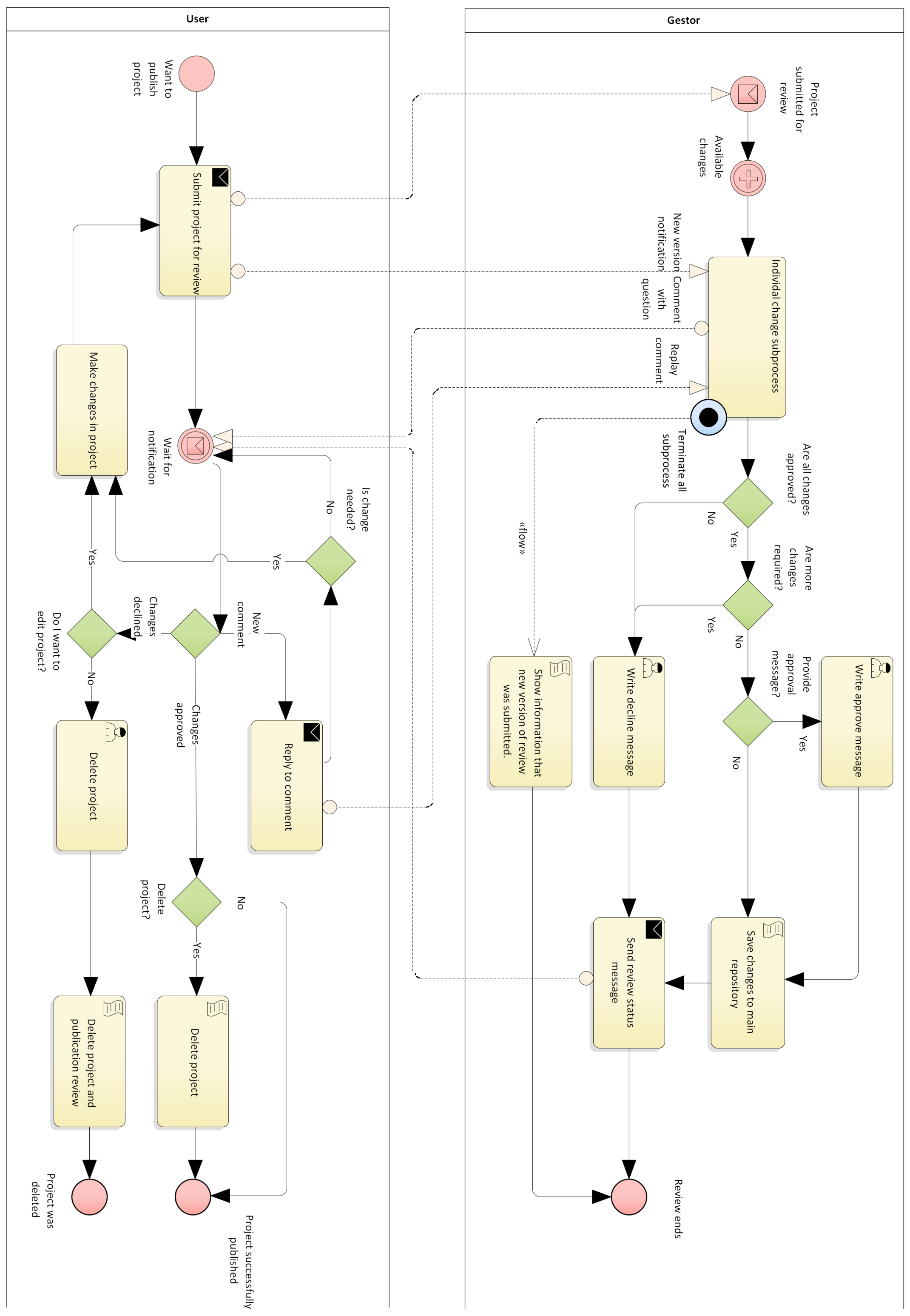


**Figure 4.6:** New review process overview

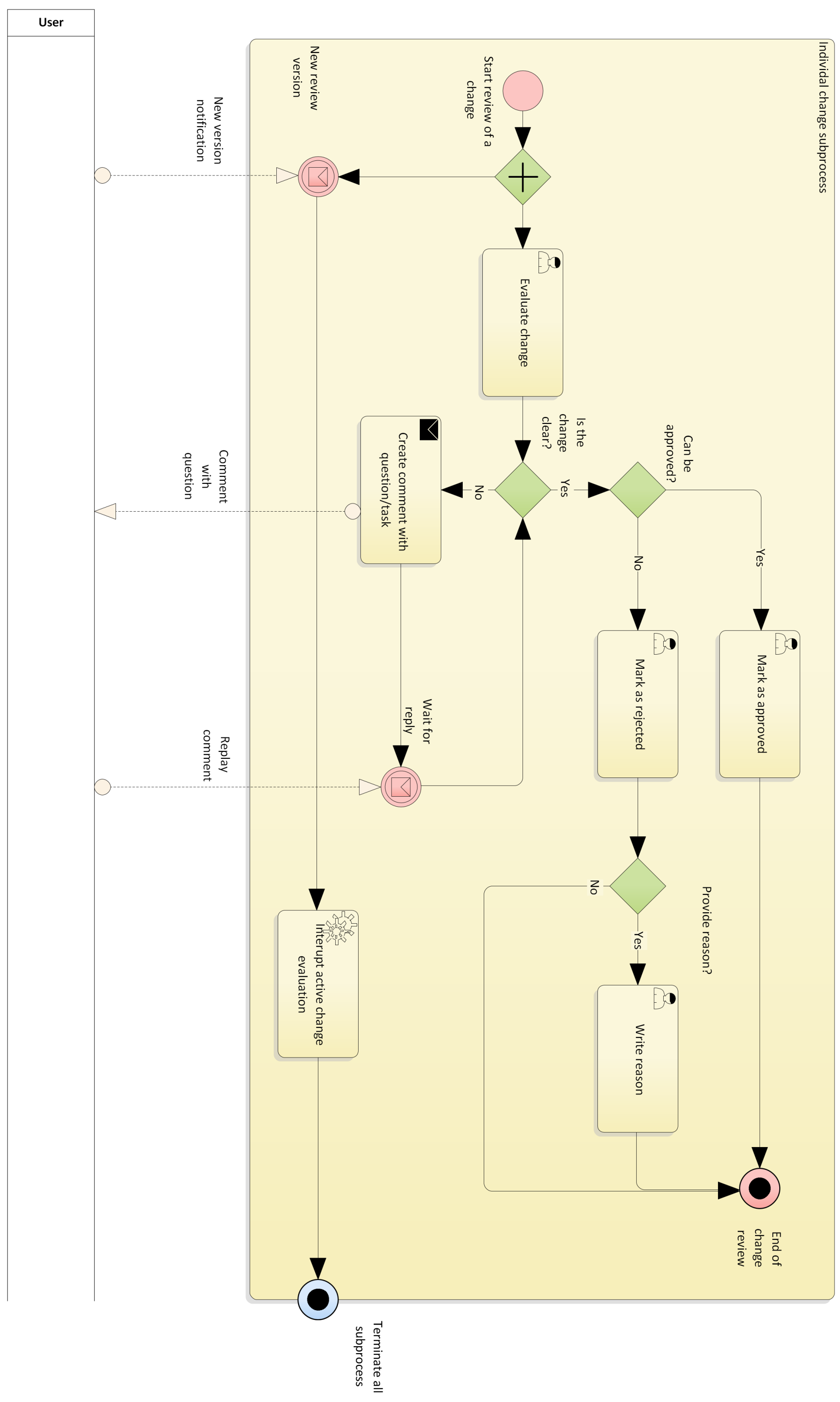


**Figure 4.7:** Change evaluation subprocess

## 4.5 Summary

In this chapter, we defined a set of requirements that should be met by a platform used for the AL review process. We examined how these needs are fulfilled in the present implementation and offered a new solution that would address the shortcomings of the current one. The comparison of the two platforms is shown in the table below.

| | Current platform (GH) | New platform (CheckIt) |
|---|---|---|
| In Assembly line | No | Yes |
| Authentication | GitHub | Assembly line |
| Triple visualisation | Turtle notation | Property-value format, graphs for relationships, Turtle notation |
| Review granularity | Per pull request | Per changed triple or set of triples (relationships) |
| Write authority | Over whole SGoV | Per vocabulary |
| Discussion over changes | Per lines of changed .ttl file | Per changed triple or set of triples (relationships) |
| UI language | English | Czech, English |

**Table 4.1:** Comparison of GitHub and CheckIt as review platforms

It is evident that the new platform offers a much more accessible and efficient review process than the current solution. By creating CheckIt and using the newly defined review process, the SGoV could be modified and expanded more accurately and quickly.

# Chapter 5

# Design of the system

This chapter focuses on the design of the new system. We define the functional and nonfunctional requirements of the system.

## 5.1 Functional requirements

The functional requirements define what the system must accomplish or must be able to do [27]. The requirements are derived from a thorough analysis of the needs and preferences of the application's users and stakeholders. The definition of a functional requirement must be precise, granular, and concise, not allowing any misinterpretation of the requirement to happen. The requirements must not contradict each other, and their fulfillment in the final product is verifiable [28]. By clearly defining the functional requirements, we establish the scope of the system and set the criteria for its successful implementation.

The complete list of identified functional requirements for the new system is provided below.

- FR: 1. The system must allow all users to log in under their AL credentials.
- FR: 2. The system must allow all users to log out.
- FR: 3. The system must allow all users to change the language of the UI.
- FR: 4. The system must allow all users to see all of their notifications.
- FR: 5. The system must redirect all users to the relevant part of the application after interacting with a notification.

- FR: 6. The system must show all users their number of unread notifications.
- FR: 7. The system must allow all users to mark all their notifications as read.
- FR: 8. The system must allow administrators to see all unresolved gestoring requests.
- FR: 9. The system must allow administrators to approve gestoring requests.
- FR: 10. The system must allow administrators to reject gestoring requests.
- FR: 11. The system must allow administrators to filter out the vocabularies with at least one gestor assigned.
- FR: 12. The system must allow administrators to see how many vocabularies have assigned at least one gestor.
- FR: 13. The system must allow administrators to assign any user as a gestor to any vocabulary.
- FR: 14. The system must allow administrators to remove any gestor from any vocabulary.
- FR: 15. The system must allow administrators to filter the list of users by their first name and last name.
- FR: 16. The system must allow administrators to add any user as an administrator.
- FR: 17. The system must allow administrators to remove any administrator from the system that is not them.
- FR: 18. The system must allow all users to see all vocabularies persisted in SGoV.
- FR: 19. The system must allow filtering a list of vocabularies by their label.
- FR: 20. The system must allow all users to see what vocabularies they are gestoring.
- FR: 21. The system must allow all users to create a gestoring request to any vocabulary they do not already gestor.
- FR: 22. The system must allow all users to see all their pending gestoring requests.
- FR: 23. The system must allow all users to see a list of unresolved publications.

- FR: 24. The system must allow all users to see a list of resolved publications.
- FR: 25. The system must allow all users to see the state of a publication alongside its closing comment (if present).
- FR: 26. The system must allow all users to see what vocabularies are in the publication.
- FR: 27. The system must allow users to see who are the gestors of a vocabulary.
- FR: 28. The system must allow all users to see which users have approved a vocabulary in a publication.
- FR: 29. The system must allow all users to see all vocabulary changes in a publication.
- FR: 30. The system must allow all users to see simple (single triple) changes in a property-value manner.
- FR: 31. The system must allow all users to see changes regarding relationships in a graph-like manner (subject and object = nodes, relationship = arc)
- FR: 32. The system must allow all users to see all changes in a Turtle syntax.
- FR: 33. The system must allow all users to see comments regarding a change.
- FR: 34. The system must allow all users to create comments on a change.
- FR: 35. The system must allow all users to see the number of comments in the change's comment thread.
- FR: 36. The system must allow all users to see the IRIs of subjects present in changes.
- FR: 37. The system must allow all users to see the IRIs of predicates present in changes.
- FR: 38. The system must allow all users to see the rejection comments of other users. The comment must include the author's name and the time elapsed from the comment's creation.
- FR: 39. The system must allow all users to view their present location inside the publication.
- FR: 40. The system must allow gestors of vocabularies that are present in the publication to accept changes regarding their gestored vocabulary.

- FR: 41. The system must allow gestors of vocabularies that are present in the publication to reject changes regarding their gestored vocabulary.
- FR: 42. The system must allow the gestor who rejected a change to create a rejection comment which must be at least ten characters long.
- FR: 43. The system must allow gestors that made a decision (approve or reject) on a change to undo their decision.
- FR: 44. The system must allow gestors to see their decision (approval or rejection) on a change.
- FR: 45. The system must show an altered change message for changes that were already processed by a particular gestor but got revised later.
- FR: 46. The system must allow gestors to dismiss the altered change message.
- FR: 47. The system must always show gestors the first unrevised change when starting or resuming a review of a vocabulary.
- FR: 48. After providing a decision to a change, the system must show the gestor the next undecided change in the vocabulary.
- FR: 49. The system must allow gestors reviewing a vocabulary to see how many changes they have yet to decide in that vocabulary.
- FR: 50. The system must allow a gestor of a vocabulary present in the publication to reject the whole publication. The gestor must provide a reason that is at least ten characters long.
- FR: 51. The system must allow a gestor of a vocabulary present in the publication to approve the whole publication when the publication contains at least one approving review per each vocabulary.

## 5.2 Non-functional requirements

The nonfunctional requirements specify the system's operational capabilities and constraints. They can be referred to as quality attributes of the system that define its performance, availability, usability, and reliability [29]. Unlike functional requirements, the focus is not on individual features but on the system's overall behavior.

- NFR: 1. The application shall be localized in Czech and English.
- NFR: 2. The application shall load in under 5 seconds on slow networks (download 4Mbps).
- NFR: 3. The application shall have a responsive layout.
- NFR: 4. The application shall be fully operational on the versions of the following browsers:
  - Chrome >= 87
  - Firefox >= 78
  - Safari >= 14
  - Edge >= 88
- NFR: 5. The application shall be operational under Docker.

# Chapter 6

# Technical analysis

The technologies that may be utilized to create a front-end application are compared in this chapter. The best option (in the context of the AL) is chosen after the comparison is complete. The selected technical solution must be completely functional under the AL technological stack.

## 6.1 Front-end library/framework

There are a lot of frameworks or libraries which can be used to build a web application. In our comparison, we will focus only on the most popular ones. According to the Stack Overflow survey[1] in 2022, the most popular ones are as follows:

- React
- Angular[2]
- Vue.js[3]

### 6.1.1 React

React is an open-source JavaScript library used for building user interfaces. It is being maintained by Meta and the open-source community [30]. The main idea behind React is the use of components which are reusable pieces of code that represent a specific part of the user interface. These components manage

[1]`https://survey.stackoverflow.co/2022/#most-popular-technologies-webframe` Accessed: 19-April-2023

[2]`https://angular.io/` Accessed: 19-April-2023

[3]`https://vuejs.org/` Accessed: 19-April-2023

their state and can be nested into each other, resulting in a complex user interface. Components are written in JSX[4], a JavaScript extension allowing the writing of XML-like syntax inside JavaScript code. One of the critical features of React is its use of virtual DOM[5], which allows the mapping of in-memory representation DOM to the real DOM. The main advantage lies in the ability to render only a specific part of the DOM when updates happen, in contrast to rendering the whole page again. The virtual DOM makes the re-rendering of changed parts of the website faster, resulting in a smoother user experience.

Unlike the other technologies mentioned below, React is a library, which means that it is entirely upon the developer to implement all the application functionalities. Luckily many frequently occurring problems (e.g., routing, data fetching, form handling) can be solved by using publicly available third-party React libraries. Let us take routing as an example, which can be implemented by using react-router[6] or wouter[7] (to name a few). Some developers prefer this freedom of choice, but it can be overwhelming and unnecessary for others.

As a solution to this problem, React-powered frameworks emerged on the market, offering out-of-the-box support for some repeatedly occurring problems (e.g., routing). React documentation recommends using one of the following frameworks:

- Next.js[8]
- Remix[9]
- Gatsby[10]

Each offers a built-in set of functionalities that the application might need.

### 6.1.2 Angular

Angular is a framework for building web applications using extended HTML syntax and TypeScript [31]. Google develops it and offers comprehensive tools to build complex, scalable, and maintainable applications.

[4]`https://facebook.github.io/jsx/` Accessed: 19-April-2023

[5]Document Object Model - `https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model` Accessed: 13-May-2023

[6]`https://reactrouter.com/en/main` Accessed: 13-May-2023

[7]`https://github.com/molefrog/wouter` Accessed: 13-May-2023

[8]`https://nextjs.org/` Accessed: 19-April-2023

[9]`https://remix.run/` Accessed: 19-April-2023

[10]`https://www.gatsbyjs.com/` Accessed: 19-April-2023

The basic building blocks of Angular applications are components. Components serve as declarative representations of views, composed of sets of screen elements that can be dynamically selected and modified by Angular. Each component defines a class containing application data and logic and is associated with an HTML template.

Angular components utilize services that offer specialized functionality that is not directly tied to views. The service providers can be injected into components as dependencies, resulting in reusable, modular code. Alongside dependency injection, Angular also offers two-way data binding. The learning curve is relatively steep due to the robustness of the framework.

### 6.1.3 Vue.js

Vue is a JavaScript framework for building user interfaces [32]. It was released in 2014 by Evan You, who, alongside the GitHub community, develops the framework to this day. Vue offers a declarative and component-based programming model that uses HTML, CSS, and JavaScript. Vue utilizes virtual DOM for quicker rendering, analogous to React. Dependency injection and two-way data binding are supported, unlike React. As mentioned in the introduction to this section, the community is the smallest out of the three frameworks/libraries presented, which could be a possible drawback.

### 6.1.4 Conclusion

The choice of technology was rather straightforward, given that all of the AL's front-end applications are created using React. It would not make sense to depart from the front-end stack because we anticipate incorporating the new application into the AL. Because of the program's scope and the lack of prior knowledge of mentioned frameworks, creating the application in Vue.js or Angular would not be practical.

We chose not to employ any React-based frameworks after giving them some thought. We put great importance on being able to manage the project entirely, and we would not be able to make considerable use of the offered frameworks' abilities.

## 6.2 Component library

Creating UI components from scratch can be a tedious and often redundant process. Therefore, developers utilize component libraries that allow them to use common UI components like tables or modals in their applications without having to build them from the ground up. The prevalent UI component libraries in AL front-end applications are Material-UI[11] and React Bootstrap[12]. We chose Material-UI as the preferred component library over React Bootstrap primarily due to our prior experience with the library and the substantial community support it receives.

## 6.3 JavaScript Build Process

In the following sections, we introduce two concepts that play an important role in developing and deploying JavaScript applications. The concepts are transpilation and source code bundling.

### 6.3.1 Transpilation

JavaScript is a fast-developing language that follows the ECMA script specification [33]. Major updates of this specification have been released annually every June since 2015. That might be problematic for developers because they do not have the guarantee that all users of their web application are using it on the latest web browsers. To make matters worse, not even the latest web browsers necessarily support the latest JavaScript features. To overcome this issue, developers utilize a process called transpilation. Transpilation takes a source code that is written in one programming language and produces equivalent source code in another [34].

JavaScript transpilers are primarily focused on transforming the newer versions of JavaScript into older, more compatible versions of it. They are also utilized for transforming JSX into standard JavaScript. That is essential for React applications because they are written using JSX.

Notable mentions of JavaScripts transpilers are listed below.

- Babel[13]
    - It can transpile ECMAScript 2015+ code into a backward-compatible version of JavaScript. It can also transform TypeScript and JSX into standard JavaScript.
- esbuild[14]

[11] `https://mui.com/` Accessed 8-May-2023
[12] `https://react-bootstrap.github.io/` Accessed 8-May-2023
[13] `https://babeljs.io/` Accessed 13-May-2023
[14] `https://esbuild.github.io/` Accessed 13-May-2023

  - Considered more as a bundler, but it has transpiler features built in. Esbuild offers the same transpilation capabilities as Babel.
- Typescript[15]
  - TypeScript is a superset of JavaScript that adds optional static typing to the language. However, the code must be transpiled into regular JavaScript before execution because static typing is not supported in standard JavaScript. TypeScript offers this transpilation.

### 6.3.2 Bundling

A JavaScript bundler is a tool that combines multiple JavaScript files and their dependencies into a single file [35]. Creating a single file that contains all the necessary code for the application to function is essential. Without bundling, developers would need to manually keep track of all the dependencies in their code, which is almost impossible with a large application. By utilizing bundling, developers can write their code in separate files, which is vital for code maintainability, and still be sure that the final product will be operational without worrying about the complicated dependency resolution.

One of the setbacks of bundling is the resulting file's potential size. For a large application, creating a single bundle would not be efficient. Browsers would have to download the entire application before using it, making the website's load time unnecessarily long. We can solve this by creating more smaller bundles, which are downloaded only when they are needed. As a result, browsers send out more network queries, but the website loads much faster.

Examples of popular bundlers are listed below:

- Webpack[16]
- Rollup[17]
- esbuild

## 6.4 Build tool

Even though the decision to create a React application was already established, it is still necessary to determine which build tool will be used for transpiling

[15]`https://www.typescriptlang.org/` Accessed 13-May-2023
[16]`https://webpack.js.org/` Accessed 13-May-2023
[17]`https://rollupjs.org/` Accessed 13-May-2023

and bundling production code and serving the local development server. Although it is technically possible to not use any of the available build tools, the official React documentation and the community agree that using them is preferable. In the following section, we introduce two build tools, **Create React App** and **Vite**, between which the decision was made.

### 6.4.1 Create React App

Create React App[18] (CRA) is a build tool that helps developers create new React applications. The tool creates a project folder structure with essential files, sets up the development server, and provides bundling and transpiling of the final code.

CRA was the way to start a new React project for a long time. The React team even recommended it in their official React documentation. However, CRA suffers from performance drops in development when a project grows in size. The reason is that CRA uses webpack, which bundles the entire application before it is locally served. This bundling creates a longer start-up time for the development server and a longer time to reflect code changes.

React applications in AL use this approach, which is understandable considering the time when they were created. However, the React team no longer recommends that developers start a new React project with CRA. The CRA approach was recommended in the old React documentation [36], but with the release of the new version, all references to CRA were deleted, favoring the use of Vite or React-based frameworks instead [30].

### 6.4.2 Vite

Vite[19] is a build tool that can be used to create React, Vue.js, Svelte[20], Preact[21], or Lit[22] applications. It offers the same functionality as CRA while offering developers a much faster development server than CRA's solution [37]. The server start-up time and updates are faster thanks to the utilization of the build processes in the following manner. Vite first splits the modules in the application into two categories: **dependencies** and **source code**.

Dependencies are pieces of code that are not expected to change frequently during development. (e.g., npm packages) These dependencies are pre-bundled

[18]`https://create-react-app.dev/` Accessed: 5-April-2023
[19]`https://vitejs.dev/` Accessed 5-April-2023
[20]`https://svelte.dev/` Accessed 5-April-2023
[21]`https://preactjs.com/` Accessed 5-April-2023
[22]`https://lit.dev/` Accessed 5-April-2023

with esbuild, which offers 10-100x faster bundle time than JavaScript-based bundlers. [38] [39]

Source code is a category containing code that not only contains pure JavaScript but has some parts that need transforming, JSX, for example. It is expected that code in this category will be modified often. This code is served over native ECMAScript modules (ESM) [40], and it is the browser's responsibility to handle the job of the bundler. Vite only transforms and serves the requested code by the browser.

The important thing to note is that this approach is used only during the application's development phase. When building for production, the native ESM approach is not used. The whole code is bundled and sent to the client when building the application for production. One potential downside of using Vite is the smaller community since it is quite a new approach to creating React applications (released in 2020).

### 6.4.3 Conclusion

Even though AL uses CRA in all front-end applications, we decided to use Vite instead. It serves a better developer experience without any significant downsides. One potential pitfall is the default naming convention of environment variables. CRA uses the *'REACT_APP_'* prefix, while Vite uses *'VITE_.'* This difference in prefixes becomes problematic when using a shared assembly line package[23]. Nevertheless, we decided not to use the shared assembly line package in this project. The reason behind this decision is explained in the implementation section (7.2).

| | CRA | Vite |
|---|---|---|
| Released | 2016 | 2020 |
| Dev server | Bundle based | Native ESM |
| Prod. bundler | Webpack | Rollup |
| JS standard default support | >= ECMAScript 2015 | >= ECMAScript 2015 |
| Typescript support | Yes | Yes |
| Recommended by React team | No | Yes |

**Table 6.1:** Comparison of Create React App and Vite

[23] `https://github.com/datagov-cz/assembly-line-shared` Accessed 9-May-2023

## 6.5 Back end

Our front-end application requires a server that would handle the calculation of changed triples, management of the review process data in a database, and persistence of the approved publications in SGoV. Such a server was developed by Bc. Michal Švagr and its source code can be found on GitHub[24]. The communication between the two is achieved by using the server's publicly available REST API [41].

[24]`https://github.com/mighantos/checkit-server` Accessed: 15-May-2023

# Chapter 7

# Implementation

This chapter provides an overview of the technologies utilized in the project and outlines the steps we took to enhance the user experience. We also clarify why we handled authentication without using the shared AL package.

## 7.1 Technological stack

We built the application with React v18 and used the functional component approach for the component declaration [42]. As mentioned in the analysis (section: 6.1.4), we are not using any react-based framework. Therefore, we had to leverage publicly available libraries to solve frequently occurring problems (e.g., routing). The list below states a problem alongside a library that has been utilized to help solve that problem.

- Routing
  - React Router[1]
- Language localization
  - React Intl[2]
- Global state
  - Redux[3]
- Network requests, Optimistic UI updates

[1]`https://reactrouter.com/en/main` Accessed 9-May-2023
[2]`https://formatjs.io/docs/react-intl/` Accessed 9-May-2023
[3]`https://redux-toolkit.js.org/` Accessed 9-May-2023

    - RTK query[4]

- Authentication
    - OIDC-client[5]
- Efficient rendering of large data sets
    - React Virtuoso[6]

## 7.2 Authentication

The AL tools employ a shared React component that provides them with an authentification mechanism and URL addresses of the other tools. The React component is published as an NPM[7] package that the tools download and use as a top-level component. The main purpose of this component is to reduce the amount of code needed for providing the authentification mechanism, which is the same for all the front-end applications of AL. However, the component was built with the technologies of the front-end tools in mind. For example, the component relies on all the tools being built using CRA, which requires the use of a specific prefix for environment variables. These variables are then parsed by the component and returned back to the tools. That makes the package almost unusable for any other building tool. Apart from having a strong connection to CRA (which we do not use), the component uses an outdated npm package (oidc-client[8] in version 1.11.5) for OIDC standard communication.

These reasons led us to develop the authentification mechanism again, utilizing the newest version of the package used in the shared component.

## 7.3 User experience

To provide an enjoyable user experience, we needed to ensure that pages were fast to load and quick to respond to user interaction.

[4]`https://redux-toolkit.js.org/rtk-query/overview` Accessed 9-May-2023
[5]`https://authts.github.io/oidc-client-ts/` Accessed 9-May-2023
[6]`https://virtuoso.dev/` Accessed 9-May-2023
[7]`https://www.npmjs.com/` Accessed 9-May-2023
[8]`https://github.com/IdentityModel/oidc-client-js` Accessed 20-May-2023

### 7.3.1 Rendering of large datasets

It is critical to consider how the dataset's rendering strategy may affect the application's performance when presenting a sizeable data collection. Degraded performance can hurt the system's usability, making its users less efficient. We can show how three different rendering strategies affect the system's performance. The figure below (7.1) shows the approaches visually, with a blue rectangle indicating the viewport, a black rectangle marking that the element is present in DOM, and gray demonstrating the opposite.

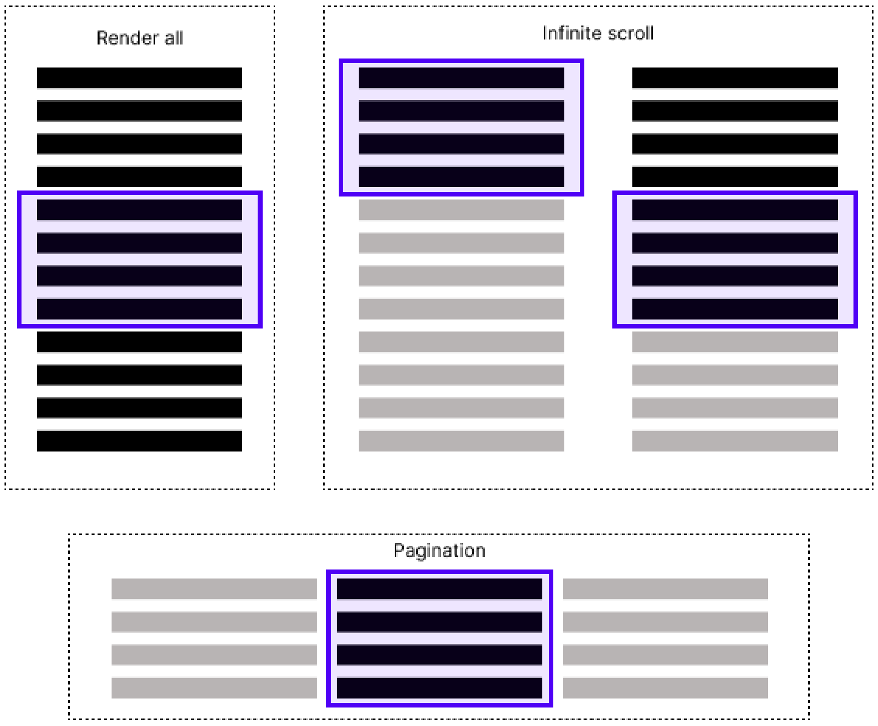


**Figure 7.1:** Different rendering approaches of large datasets

- Render all
  - It renders all elements on the initial render. Causing long page loads and unnecessarily large DOM size
- Infinite scroll
  - It renders only a limited subset of elements on the initial render. The rest of the elements is shown upon request (scroll). However, the already rendered elements stay in the DOM, also causing its size to be increased.
- Pagination

- It renders only a limited subset of elements on the initial render. The DOM does not get larger because previously rendered elements get replaced by new ones. However, the look of pagination might only be suitable for some applications.

We can see that all of the three mentioned techniques have their shortcomings. A rendering concept called virtualization was created to overcome those problems. Virtualization provides capabilities of infinite scroll without keeping the nonvisible (previously rendered) elements in the DOM, making the rendering technique highly efficient.

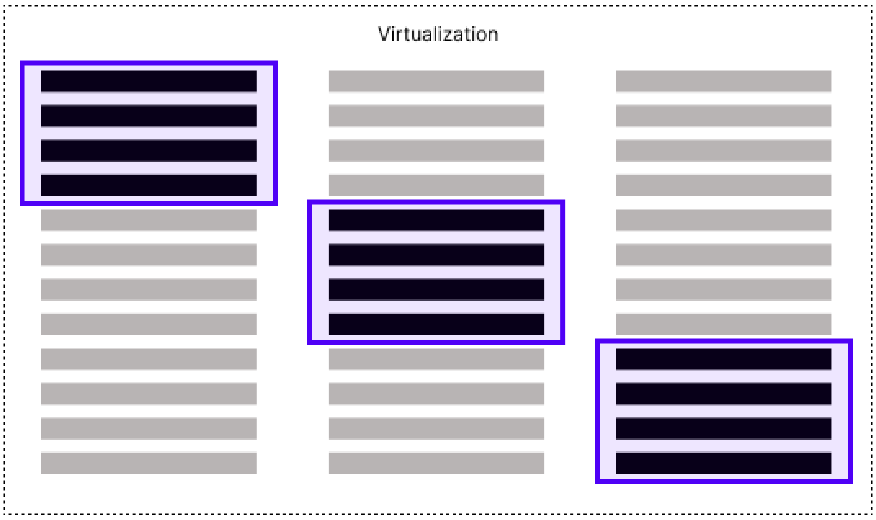


**Figure 7.2:** Virtualization rendering techniques on a large dataset

Because the pagination look is unsuitable for our application, we had to use the virtualization approach to ensure a good user experience.

### 7.3.2 Optimistic updates

We used the optimistic UI update design pattern to make the application feel quicker and more responsive [43]. The idea behind this pattern is to make the UI behave as if a change (mutation of data) made by the user was successful prior to getting confirmation from the server that it had been carried out. If an error happens and the server does not return a confirmation of the action, the UI must return to its original state.

Optimistic UI update pattern is only applicable to use cases where the developer is aware of the altered resource's state after the mutation has occurred in advance. To put it another way, a developer does not rely on the information contained in the confirmation message. For the UI, the mere confirmation that a resource was updated is sufficient. For example, adding *likes* on social media posts is usually done with an optimistic UI update approach. Users do not have to wait until a confirmation from a server is returned to let users know they added a *like* to a particular post. The *like* (mutation of data) is added instantaneously, making the application feel faster than it actually is.

We impose this strategy on various users' actions. For example, the approval or rejection of a change is done instantaneously, not holding the user back from reviewing the remaining changes in the publication. Another example is the action of requesting a gestor role. Users see a badge indicating the request was created immediately after interacting with the request button, eliminating the need for blocking loading spinners that would slow down the user.

## 7.4 Overview of implemented features

In the process analysis (section 4.1), we identified several requirements that a review process platform should meet in order to ensure the process's efficacy and accuracy. Section 5.1 described the system's functional requirements, which were implemented to allow the new process to function in the new CheckIt tool. Let us inspect how the platform requirements are implemented in the application.

### 7.4.1 User-friendly data visualization

We visualize the data as close to the rest of the AL tools as possible. For a single triple change, we show the data in a property-value manner (similar to TermIt, figure: 7.3). Thus for a predefined set of predicates, we replace their IRIs with the language-typed labels used in TermIt or OG. For the predicates that do not have labels shown in TermIt or OG, we offer CheckIt's custom labels or show the predicates' full IRIs.

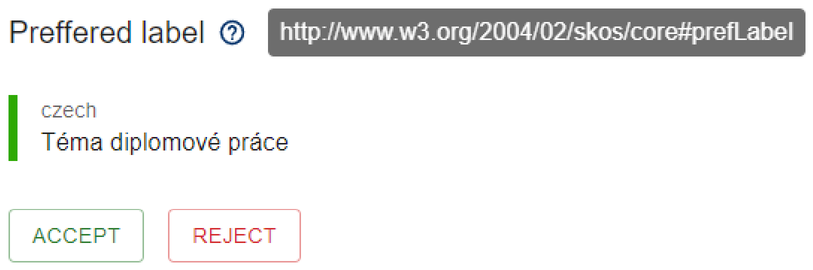


**Figure 7.3:** Single triple change visualization

Relationships are shown visually using nodes connected by arcs (similar to OG, figure: 7.4). For users that can make use of the Turtle syntax (similar to GH, figure: 7.5), we provide the option to see the changes in their raw Turtle form.

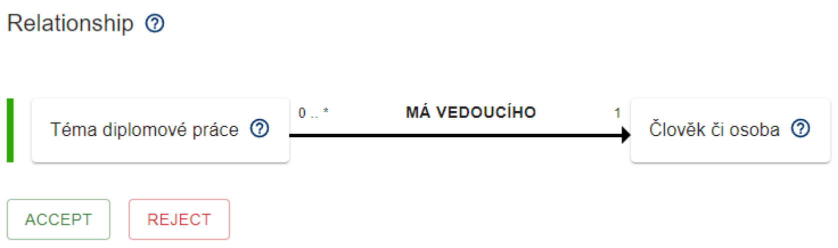
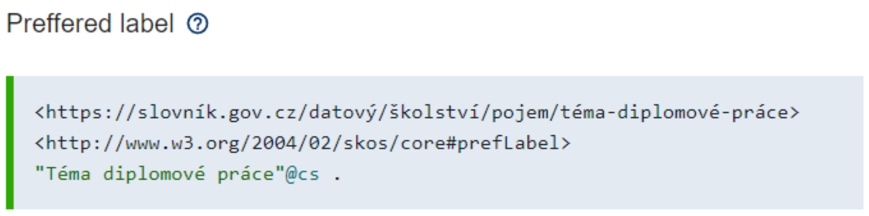


**Figure 7.4:** Relationship visualization



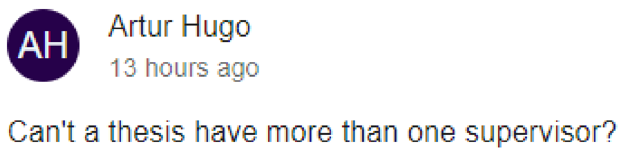
```
<https://slovník.gov.cz/datový/školství/pojem/téma-diplomové-práce>
<http://www.w3.org/2004/02/skos/core#prefLabel>
"Téma diplomové práce"@cs .
```

**Figure 7.5:** Change Turtle output

All visualizations contain a color indicator that tells the user whether the triples are newly created (green), modified (orange), or deleted (red). When a triple object is modified, we show the object's previous value next to the new one so the reviewers better understand what exactly was modified.

### 7.4.2 Communication

Communication among reviewers and other system users is possible in the discussion thread attached to each change. Having a separate thread per each change makes it easier for everyone to understand which change is being discussed. In the discussion thread, gestors can request modifications to the discussed change or debate the meaning of the change with other system users.



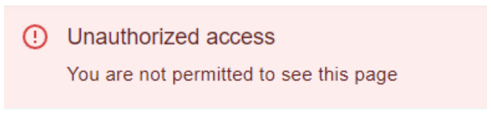
Can't a thesis have more than one supervisor?

**Figure 7.6:** Comment in a discussion thread

### 7.4.3 Defined user access

In section 4.3 we presented three roles that are needed for the new set of processes. These roles define what the users are eligible to do in the new process and therefore define what actions are offered to them in the CheckIt application. Let us point out specific examples where the UI changes accordingly to the user role.

The administrator panel is a section of the application accessible only to administrators as it offers management of the gestor and administrator roles. To prevent unauthorized access to this section, CheckIt hides all the navigational elements leading to the panel from non-administrators (figure: B.1). Also, when a user who is not an administrator requests the administrator panel through its URL, CheckIt displays a "Forbidden Access" message.


Unauthorized access

You are not permitted to see this page

**Figure 7.7:** Unauthorized access message

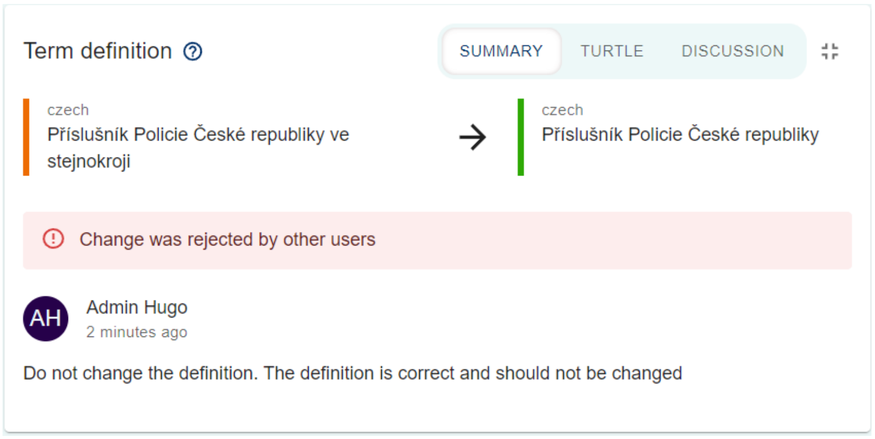
The list of changes in a publication also changes to reflect the users' role properly. Gestors are offered options for processing the changes, while the rest of the users can access only the changes' content and attached discussion threads (figure: B.10).

The last notable mention is the approval or rejection of entire publications. Only eligible gestors are provided with the options to process the publication (figure: B.8).

### 7.4.4 Collaboration

We defined the new review process to allow multiple users to participate in a single publication review. How users collaborate on reviews is thoroughly described in section 4.4.2 and the primary UI features used to do so were mentioned in the previous sections (defined user access and communication). However, CheckIt still offers some additional functionalities that help users in their collaborative efforts.

First, we can mention the rejection comments on changes. These comments are shown right next to the changed data so every user can see them.



Term definition

SUMMARY TURTLE DISCUSSION

czech
Příslušník Policie České republiky ve stejnokroji

czech
Příslušník Policie České republiky

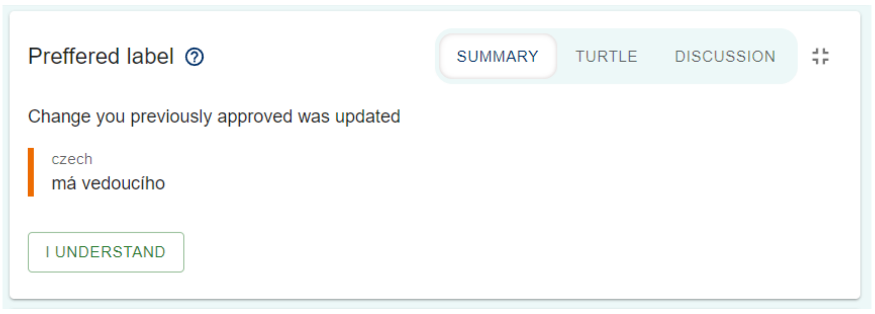
Change was rejected by other users

AH Admin Hugo
2 minutes ago

Do not change the definition. The definition is correct and should not be changed

**Figure 7.8:** Rejection comment on a change

Gestors who processed a change that was later revised must be made aware of it. We notify the gestors by displaying a special message next to the change that has to be dismissed by the gestor.



Preffered label

SUMMARY TURTLE DISCUSSION

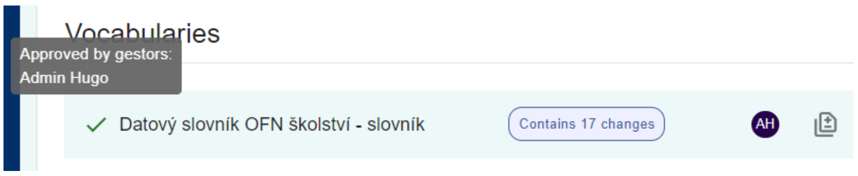
Change you previously approved was updated

czech
má vedoucího

I UNDERSTAND

**Figure 7.9:** Updated change that was already approved

Another important feature that increases the efficiency of the new review process is showing who already approved a vocabulary inside a publication. For gestors, this is crucial because they do not need to go over changes that some other gestor has already approved.



Vocabularies

Approved by gestors:
Admin Hugo

Datový slovník OFN školství - slovník

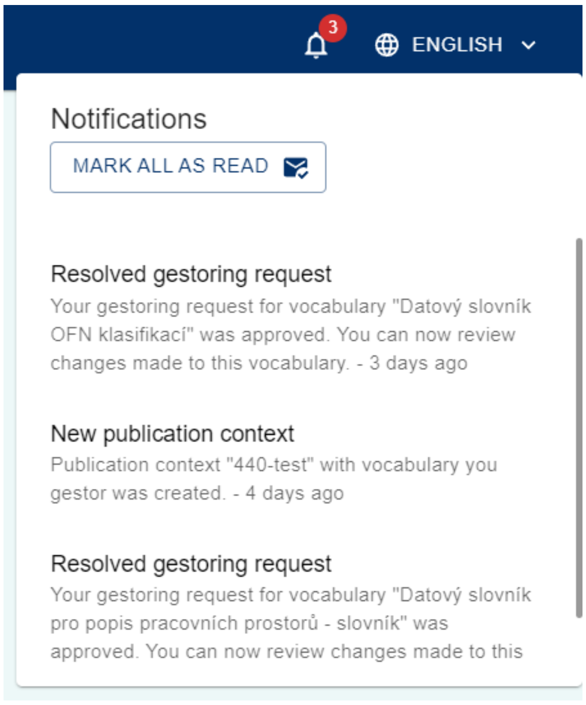
Contains 17 changes

AH

**Figure 7.10:** Approved vocabulary in a publication

Collaboration would not be possible without UI support for requesting and managing the gestor role. We already mentioned the administrator panel used for this exact purpose. Administrators can use the admin panel to assign a gestor role to users and evaluate incoming gestoring requests (figures: B.2). Users create these requests in a section of the application where all the SGoV vocabularies are listed. In the list, users can see which vocabularies are already gestoring and which are awaiting an evaluation of their request from an administrator (figure: B.11).

Lastly, users are notified about the relevant actions of other users by a personalized notification system. The notifications are made to inform relevant users about new or updated publications, created comments, and created or processed gestoring requests.



Notifications

Resolved gestoring request
Your gestoring request for vocabulary "Datový slovník OFN klasifikací" was approved. You can now review changes made to this vocabulary. - 3 days ago

New publication context
Publication context "440-test" with vocabulary you gestor was created. - 4 days ago

Resolved gestoring request
Your gestoring request for vocabulary "Datový slovník pro popis pracovních prostorů - slovník" was approved. You can now review changes made to this

**Figure 7.11:** Notifications

# Chapter 8

# Testing

Software testing plays an important role during the software development process. Incorporating software testing in development is beneficial for various reasons. It increases the software's quality by identifying errors that cause the system not to work as intended. Early issue detection and resolution greatly reduce the cost and time needed for resolving problems later in development or during production. A well-tested software should perform reliably and should lead to higher customer satisfaction [44].

Software testing is divided into two categories, automated and manual testing. Automated testing utilizes the use of specialized software tools or frameworks to execute pre-defined test cases that verify the software functionalities. Having the test automated allows for their fast and repeated execution. That makes them particularly useful for regression testing, which ensures that previously tested software performs the same after it was modified [45].

Manual testing is done by people, making the test cases slower to execute and possibly more expensive in contrast to automated testing. Depending on the chosen testing method, the test cases might be defined precisely, giving the tester step-by-step instructions on what needs to be done or bit ambiguously, allowing the tester to explore the application more freely. While giving testers step-by-step instructions might ensure that the outlined path works as expected, the free exploration approach helps uncover hidden problems that could otherwise remain undetected.

We used both manual and automated approaches to test our application. In the following section, we describe unit testing, our selected approach for automated testing, and the scenario testing technique that was used for manual testing.

## 8.1 Unit testing

Unit testing is a type of software testing that is focused on testing small units of the application in isolation to verify their correctness. The isolation is achieved using test doubles, which provide functionalities of other application parts [46]. Typically, unit tests are automated tests created and executed by the software developers [47].

During the development of CheckIt, we utilized Vitest[1] as a unit test framework. Together with the React-testing library[2] , we tested individual components in the application to ensure their correctness.

## 8.2 Scenario testing

We employed a user testing strategy known as scenario testing for manual testing. Scenario testing is based on creating fictional but credible stories (scenarios) that demonstrate real-world usage of the software. The scenarios do not contain a step-by-step guide of what needs to be done but rather tell a compelling story that makes the tester feel like an end-user of the software with an objective they need to complete [48].

The main advantage of scenario-based testing is the gained understanding of how users interact with the software, making the developer understand how intuitive and usable the software is to its end users. Apart from better understanding the user experience, scenario testing increases the overall test coverage of the system because each scenario typically covers multiple test cases at once.

### 8.2.1 System usability scale

We evaluate the usability of the system with the System Usability Scale (SUS) [49]. The SUS is a widely used questionnaire-based tool designed to measure the perceived usability of systems, applications, and websites [50]. The questionnaire consists of ten statements. Respondents are asked to rate their level of agreement or disagreement with each statement using the Likert Scale. The Likert scale is a five-point scale that ranges from strongly disagree (scale position = 1) to strongly agree (scale position = 5).

The statements are as follows:

[1]`https://vitest.dev/` Accessed 10-May-2023

[2]`https://testing-library.com/docs/react-testing-library/intro` Accessed 10-May-2023

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

The SUS yields a single number (SUS score), which serves as a composite indicator of the system's overall usefulness. To calculate the score, we sum the value of all items in the questionnaire and multiply it by 2.5 to obtain the final result. The value of items is calculated in the following manner. For items 1,3,5,7 and 9, the score contribution is the scale position minus one. For items 2,4,6,8 and 10, the contribution is five minus the scale position. After summing and multiplying the values, the final result (SUS score) ranges between 0 and 100. Research shows that a SUS score that is higher than 68 would be considered above average, and anything lower is below average [51].

## 8.3 User testing

We conducted the user testing on an instance of AL that was already running on the Czech Technical University servers. The AL instance was updated to include the CheckIt application[3] and the CheckIt server.

Users were following testing scenarios defined in the appendix (A). The scenarios were focused on two primary user actions: requesting a gestor role and performing reviews of publications. After each scenario, testers rated the difficulty of achieving the scenario's objective and optionally wrote remarks regarding the task. When testers finished all five scenarios, they rated the application's usability using SUS.

[3]Application available at: `https://onto.fel.cvut.cz/modelujeme/v-n%C3%A1stroji/checkit/`

Our user testing involved a total of six testers. The results of the user testing are displayed below, beginning with the testers' evaluation of how difficult it was to complete a testing scenario. The numbers in the table (8.1) represent how many testers assign the mentioned difficulty in the column to a particular scenario.

| Testing scenario | Without difficulty | Slightly difficult | Very difficult | Could not finish |
|---|---|---|---|---|
| Logging in | 6 | 0 | 0 | 0 |
| Requesting a gestor role | 3 | 3 | 0 | 0 |
| Positive review | 5 | 1 | 0 | 0 |
| Approving publication | 4 | 1 | 1 | 0 |
| Negative review | 5 | 1 | 0 | 0 |

**Table 8.1:** Perceived difficulty of finishing the test scenarios

The SUS evaluation turned out very positive. The lowest score achieved was 75, while the highest reached 95 points. The average and our final score across all testers is 84.5 points, which is above the 68 points threshold, making the system above average in terms of usability.

| | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 | Score |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Tester 1 | 3 | 2 | 3 | 2 | 4 | 1 | 5 | 2 | 3 | 1 | 75 |
| Tester 2 | 5 | 1 | 5 | 1 | 5 | 1 | 4 | 2 | 5 | 1 | 95 |
| Tester 3 | 4 | 1 | 4 | 2 | 5 | 1 | 5 | 1 | 3 | 1 | 82.5 |
| Tester 4 | 5 | 1 | 5 | 1 | 4 | 1 | 4 | 1 | 3 | 1 | 90 |
| Tester 5 | 4 | 2 | 4 | 1 | 4 | 2 | 3 | 4 | 5 | 1 | 75 |
| Tester 6 | 5 | 2 | 5 | 1 | 4 | 2 | 4 | 1 | 5 | 1 | 90 |

**Table 8.2:** SUS scores

### 8.3.1 Identified shortcomings

User testing identified a couple of issues that, if addressed, could lead to greater user satisfaction with the application.

One tester had a serious problem approving a publication after successfully finishing the vocabulary review. The required step back to the publication overview was not clearly communicated, making the tester return to the home page. Ultimately, the tester approved the publication but rated the experience negatively due to the lack of clear instructions after finishing the vocabulary review. We can solve this problem by providing more precise guidance after a review is finished and offering a go-back button redirecting users to the publication overview page.

The user testing also exposed a bug in the application that showed an incorrect number of unread notifications. That resulted in the number of unread notifications decreasing to negative values when users interacted with them. The cause for the bug was not updating the unread counter after a new notification was fetched. However, this bug has been fixed and is no longer present in the application.

Lastly, let us list some possible improvements that were mentioned directly by testers or resulted from the testers' complaints:

- Making the list outputs (changes, publications, vocabularies) to use the whole screen vertically.
    - With the change list, the issue is present only on 1440p resolution monitors and higher.
- Add the ability to cancel a gestoring request.
- Highlighting a list row over which a user has a cursor.
    - The interaction buttons are at the end of the lists' row, which can be problematic on larger monitors without row highlighting.
- Better indicate that the sidebar is expandable.

# Chapter 9

# Conclusion

This thesis introduced readers to the technologies of the Semantic Web, starting with RDF and the structure of triples. It then introduced Turtle, a concrete syntax for RDF used in the AL, and exemplified its syntax with a comparison to N-Triples syntax. The thesis then explained and showcased some of the well-known RDF vocabularies the AL utilizes.

After providing readers with the necessary technological background of the Semantic Web, we introduced the AL and its key concepts. We presented terms, the basic building blocks of all vocabularies in SGoV, and we outlined the internal structure of these vocabularies. Then the tools with which users create these vocabularies and the terms within them were presented. After that, the technical overview of the whole AL was provided.

Having outlined how the AL tools work, we focused on the process of reviewing changes originating from them. We explained the importance of an adequate review process and set the general requirements for a platform that would enable such process. We then looked at how the AL's current review platform met these requirements and identified serious flaws in the review process that are caused by the limitations of the platform. To improve the review process, we outlined the foundation of a new platform, an application called CheckIt, that would meet our stated general requirements and would be designed to work within the AL. We defined a new review process alongside new entities and precisely defined user roles that the new platform would support and therefore solve the identified issues of the current review process.

We then defined the functional and non-functional requirements for the new solution, ultimately defining the application's scope. Then we analyzed available technologies used for building web applications. The technologies were compared, and their underlying concepts explained, leading to the decision to build a React application and utilize Vite as the build tool. Then we detailed how the new application was implemented and demonstrated how the new solution fulfills the general requirements of a review platform. The last chapter was focused on software testing. We explained how we tested the application using automated and manual testing and summarised the results of the user testing.

## 9.1 Evaluation

The newly created CheckIt application is a solid replacement for the current review process solution that uses GH, as it offers a user-friendly way of evaluating changes applied to SGoV and has well-defined rules for who can evaluate which changes. The user tests, which were very positive in terms of user-friendliness, testify to the overall success of the implementation. By making CheckIt an integral part of the AL, users would benefit from the new, more transparent, and approachable review process in which they could also participate. Also, the vocabulary manipulation workflow would take place entirely within the tools provided by the AL, making the toolset complete and not reliant on any external tools.

## 9.2 Future work

CheckIt's final implementation fulfills every functional requirement described in section 5.1. However, there are still some enhancements, which we outline below, that could further improve the user experience.

### 9.2.1 Responsivity improvements

Even though the application is built with responsivity in mind, some edge cases still exist where the user interface does not reflect the screen dimension properly. For example, navigational elements displayed on a change can sometimes collide with the rest of the change's content. Fixing these minor issues could improve the overall users' satisfaction with the software.

### 9.2.2 Extension of known predicate IRIs

The predefined set of predicate IRIs that gets transformed into a localized label currently contains only the most used IRIs in the AL. This set can always get extended to cover more IRIs that could originate from the AL vocabulary manipulation process. Having more IRIs transformed into labels helps the user-friendly data visualization objective.

### 9.2.3 Editing of comments

Currently, users can only add new comments to a discussion thread without the possibility of editing them. Adding this feature could improve the clarity of a discussion, avoiding unnecessary comments that contain corrected grammar or fixed typos.

### 9.2.4 Diagram changes

Ontology engineers create semantic relationships in OG by placing terms on canvases and linking them together, which results in a diagram. However, CheckIt displays the relationships individually, ignoring where they appear in the diagram and how the overall diagram looks. Displaying the entire diagram with the highlighted publication changes could help gestors better evaluate the publication.

### 9.2.5 Full incorporation into the AL

To fully incorporate CheckIt into the AL vocabulary workflow, a couple of modifications to the MC are required. The publish button, which creates the pull request in GitHub by calling the SGoV server, should be replaced with a solution that calls only the CheckIt server. The server will find the changes and send them to our CheckIt application, where gestors can process them. Also, every time a user ends a project in MC, the CheckIt server must be notified so that any active publication that was created as a result of that project is deleted as well.

# Bibliography

[1] The Publications Office of the European Union. What is open data? [Online; accessed 23-Apr-2023] https://data.europa.eu/en/trening/what-open-data.

[2] European Social Fund. Rozvoj datových politik v oblasti zlepšování kvality a interoperability dat veřejné správy. [Online; accessed 23-Apr-2023] https://www.esfcr.cz/projekty-opz/-/asset_publisher/ODuZumtPTtTa/content/rozvoj-datovych-politik-v-oblasti-zlepsovani-kvality-a-interoperability-dat-verejne-spravy.

[3] KODI. Kodi - rozvoj datových politik v oblasti zlepšování kvality a interoperability dat veřejné správy. [Online; accessed 23-Apr-2023] https://data.gov.cz/kodi/.

[4] Karel Klíma, Petr Křemen, Martin Ledvinka, Michal Med, Alice Binder, Miroslav Blaško, and Martin Nečaský. Assembly line for conceptual models. [Online; accessed 23-Apr-2023] https://www.semantic-web-journal.net/content/assembly-line-conceptual-models-0.

[5] Antoine Isaac and Ed Summers. SKOS simple knowledge organization system primer. W3C note, W3C, August 2009. [Online accessed 9-Apr-2023] https://www.w3.org/TR/2009/NOTE-skos-primer-20090818/.

[6] Bijan Parsia, Markus Krötzsch, Sebastian Rudolph, Peter Patel-Schneider, and Pascal Hitzler. OWL 2 web ontology language primer (second edition). W3C recommendation, W3C, December 2012. [Online accessed 11-Apr-2023] https://www.w3.org/TR/2012/REC-owl2-primer-20121211/.

[7] Rozvoj datových politik v oblasti zlepšování kvality a interoperability dat veřejné správy . C5v2 - koncepce sémantického slovníku pojmů pro potřeby konceptuálního datového modelování agend. [Online; accessed 23-Apr-2023] https://data.gov.cz/kodi/v%C3%BDstupy/C5V2.pdf.

[8] Martin Ledvinka, Petr Křemen, Lama Saeeda, and Miroslav Blaško. Termit: A practical semantic vocabulary manager. Technical report, Czech Technical University in Prague, May 2020. [Online accessed 25-Apr-2023] `https://www.researchgate.net/publication/341541783_TermIt_A_Practical_Semantic_Vocabulary_Manager`.

[9] Alice Binder and Petr Křemen. Ontographer: a web-based tool for ontological conceptual modeling. Technical report, Czech Technical University in Prague, 2021. [Online accessed 25-Apr-2023] `https://ceur-ws.org/Vol-3115/paper2.pdf`.

[10] Gavin Carothers and Eric Prud'hommeaux. RDF 1.1 turtle. W3C recommendation, W3C, February 2014. [Online accessed 20-Jan-2023] `https://www.w3.org/TR/2014/REC-turtle-20140225/`.

[11] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

[12] RDF Core Working Group. Rdf. `https://www.w3.org/RDF/`. [Online; accessed 5-Jan-2023].

[13] M. Duerst and M. Suignard. Internationalized Resource Identifiers (IRIs). RFC 3987, IETF, 2005.

[14] Guus Schreiber and Yves Raimond. RDF 1.1 primer. W3C note, W3C, June 2014. [Online accessed 9-Jan-2023] `https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/`.

[15] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifiers (uri): Generic syntax. RFC 2396, Internet Engineering Task Force, August 1998. [Online Accessed: April 18, 2023] `https://www.rfc-editor.org/info/rfc2396`.

[16] David Wood, Richard Cyganiak, and Markus Lanthaler. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, February 2014. [Online accessed 9-Jan-2023] `https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`.

[17] Patrick Hayes and Peter Patel-Schneider. RDF 1.1 semantics. W3C recommendation, W3C, February 2014. [Online accessed 20-Jan-2023] `https://www.w3.org/TR/2014/REC-rdf11-mt-20140225/`.

[18] Gavin Carothers and Andy Seaborne. RDF 1.1 n-triples. W3C recommendation, W3C, February 2014. [Online Accessed: April 18, 2023] `https://www.w3.org/TR/2014/REC-n-triples-20140225/`.

[19] W3C Working Group. Vocabularies. [Online; accessed 8-Jan-2023] `https://www.w3.org/standards/semanticweb/ontology`.

[20] Ramanathan Guha and Dan Brickley. RDF schema 1.1. W3C recommendation, W3C, February 2014. [Online accessed 20-Jan-2023] `https://www.w3.org/TR/2014/REC-rdf-schema-20140225/`.

[21] Alistair Miles and Sean Bechhofer. SKOS simple knowledge organization system reference. W3C recommendation, W3C, August 2009. [Online accessed 9-Apr-2023] https://www.w3.org/TR/2009/REC-skos-reference-20090818/.

[22] Antoine Isaac and Ed Summers. SKOS simple knowledge organization system primer - concepts. W3C note, W3C, August 2009. [Online accessed 9-Apr-2023] https://www.w3.org/TR/2009/NOTE-skos-primer-20090818/#secconcept.

[23] Dublin Core Metadata Initiative. About dublin core. [Online; accessed 9-Apr-2023] https://www.dublincore.org/about/.

[24] Dublin Core Metadata Initiative. Dublin Core Metadata Element Set, Version 1.1, 2012. [Online; accessed 9-Apr-2023] https://www.dublincore.org/specifications/dublin-core/dcmi-terms/#section-7.

[25] Giancarlo Guizzardi, Claudenir M. Fonseca, Alessander Botti Benevides, João Paulo A. Almeida, Daniele Porello, and Tiago Prince Sales. Endurant types in ontology-driven conceptual modeling: Towards ontouml 2.0. In Juan C. Trujillo, Karen C. Davis, Xiaoyong Du, Zhanhuai Li, Tok Wang Ling, Guoliang Li, and Mong Li Lee, editors, *Conceptual Modeling*, Cham, 2018. Springer International Publishing.

[26] Shakirat Sulyman. Client-server model. *IOSR Journal of Computer Engineering*, 16:57–71, 01 2014.

[27] Ruth Malan, Dana Bredemeyer, et al. Functional requirements and use cases. *Bredemeyer Consulting*, 2001. [Online Accessed: May 10, 2023] https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=aceaa41855c38aebe7c823e60e94b39506a92b99.

[28] Nuclino Team. Functional requirements. Nuclino. [Online Accessed: May 10, 2023] https://www.nuclino.com/articles/functional-requirements.

[29] Andrew Stellman and Jennifer Greene. *Applied Software Project Management*. O'Reilly, Sebastopol, 2006. ISBN: 0-596-00948-8.

[30] React Development Team. React. [Online; accessed 19-Apr-2023] https://react.dev/.

[31] Angular Development Team. What is angular? [Online; accessed 19-Apr-2023] https://angular.io/guide/what-is-angular.

[32] Vue.js Development Team. Introduction - what is vue.js? [Online; accessed 19-Apr-2023] https://vuejs.org/guide/introduction.html#what-is-vue.

[33] ECMA International. Ecmascript® 2023 language specification. [Online Accessed: May 13, 2023] `https://www.ecma-international.org/publications-and-standards/standards/ecma-262/`.

[34] Devopedia. Transpiler. [Online Accessed: May 13, 2023] `https://devopedia.org/transpiler`.

[35] Alberto Gimeno. How javascript bundlers work. [Online Accessed: May 13, 2023] `https://medium.com/@gimenete/how-javascript-bundlers-work-1fc0d0caf2da`.

[36] React Development Team. Create a new react app. [Online Accessed: May 13, 2023] `https://legacy.reactjs.org/docs/create-a-new-react-app.html`.

[37] Nilanth. Use vite for react apps instead of cra. *dev.to*, 2021. [Online Accessed: April 18, 2023] `https://dev.to/nilanth/use-vite-for-react-apps-instead-of-cra-3pkg`.

[38] Evan You and Vite Contributors. Vite.js: Why vite? [Online; accessed April 5, 2023] `https://vitejs.dev/guide/why.html`.

[39] Evan Wallace. Esbuild FAQ: Benchmark Details. [Online; accessed April 5, 2023] `https://esbuild.github.io/faq/#benchmark-details`.

[40] ECMA International. Ecmascript 2015 language specification – ecma-262 6th edition. [Online Accessed: May 13, 2023] `https://262.ecma-international.org/6.0/#sec-modules`.

[41] Mark Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces.* O'Reilly Media, Inc., 2011.

[42] Facebook. React - components and props. [Online Accessed: May 9, 2023] `https://legacy.reactjs.org/docs/components-and-props.html#function-and-class-components`.

[43] Simon Hearne. Optimistic ui patterns. [Online Accessed: May 9, 2023] `https://simonhearne.com/2021/optimistic-ui-patterns/`.

[44] Andreas Spillner. *Software Testing Foundations: A Study Guide for the Certified Tester Exam.* Rocky Nook, Santa Barbara, CA, 4th edition, 2014.

[45] Elfriede Dustin, Jeff Rashka, and John Paul. *Automated software testing: Introduction, management, and performance: Introduction, management, and performance.* Addison-Wesley Professional, 1999.

[46] Martin Fowler. Testdouble. [Online Accessed: May 10, 2023] `https://martinfowler.com/bliki/TestDouble.html`.

[47] Guru99. Unit testing guide. [Online Accessed: May 10, 2023] `https://www.guru99.com/unit-testing-guide.html`.

[48] JD Cem Kaner. An introduction to scenario testing. *Florida Institute of Technology, Melbourne*, pages 1–13, 2013.

[49] John Brooke. Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, 189, 11 1995.

[50] James R Lewis. The system usability scale: past, present, and future. *International Journal of Human–Computer Interaction*, 34(7):577–590, 2018.

[51] Usability.gov. System usability scale. [Online Accessed: May 10, 2023] `https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html`.

# Appendix A

# Testing scenarios

## A.1 Logging in

You must log in to CheckIt to do your job. Use the login details provided to log in.

## A.2 Requesting a gestor role

On your way to the office today, you met a colleague in the corridor who told you that you had to request the right to the vocabulary: Datový slovník OFN aktualit – slovník. The dictionary is expected to be heavily edited, and your supervisor wants only the best employees to evaluate the changes. Since your promotion is within reach, you get to work right away.

## A.3 Positive review

As dictionary editor: Vyhláška č. 268/2009 Sb. o technických požadavcích na stavby, you are responsible for all publications related to this vocabulary. This means you must check all changes that modify the vocabulary and ensure they are correct. Your good friend has informed you by email that he has made a publication that modifies the vocabulary. Since he is a capable modeler, you will approve all the changes.

## A.4 Approving publication

Once all changes to the publication have been approved, it is necessary to approve the entire publication and thus officially incorporate them into SGoV. Without the approval of the whole publication, your work as a reviewer would be wasted. See the detail of Vyhláška č. 268/2009 Sb. o technických požadavcích na stavby publication and approve the entire publication with the following message: "The changes comply with the decree ." If you have failed to complete the task from the previous step, skip this task.

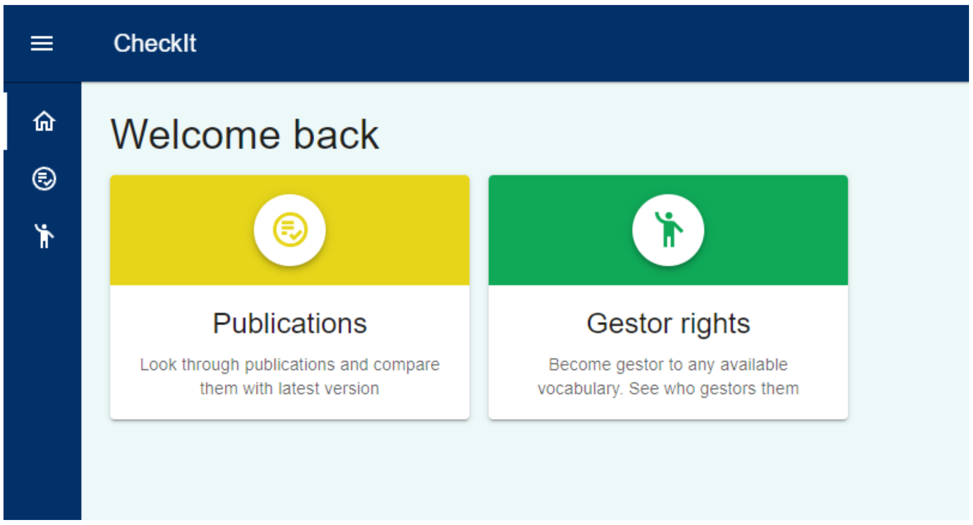
## A.5 Negative review

The last task of the day is to check the publication COUNCIL DIRECTIVE 1999/37/EC on the registration documents for vehicles. The publication contains a vocabulary of the same name, but the vocabulary is already complete and should not be edited. Therefore reject all the changes in this publication. For the change of the alternative title, write the following reason for rejection: "The VIN code is an essential part of the concept. It is not possible to have the concept without it".

# Appendix B

# Screenshots of the application

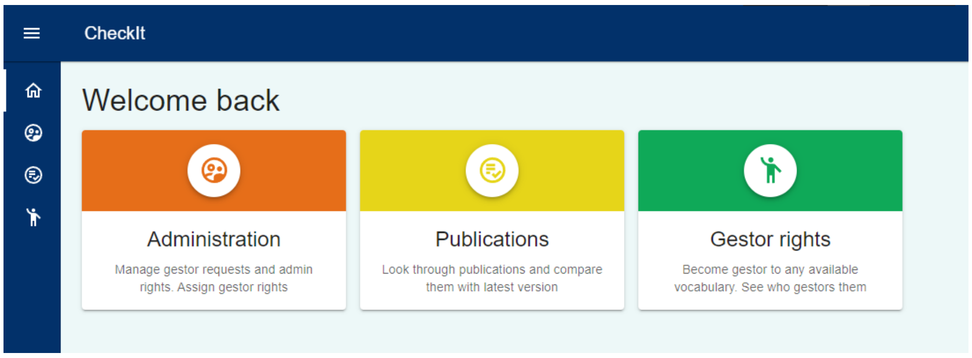
## B.1 Home page



Look through publications and compare them with latest version

Become gestor to any available vocabulary. See who gestors them

**Figure B.1:** Home page for non-administrators


CheckIt

Welcome back

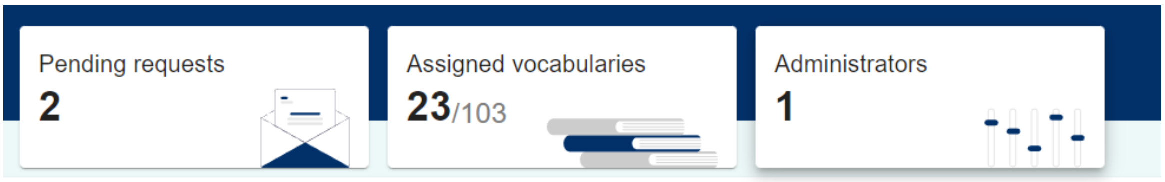
Administration
Manage gestor requests and admin rights. Assign gestor rights

Publications
Look through publications and compare them with latest version

Gestor rights
Become gestor to any available vocabulary. See who gestors them

**Figure B.2:** Home page for administrators

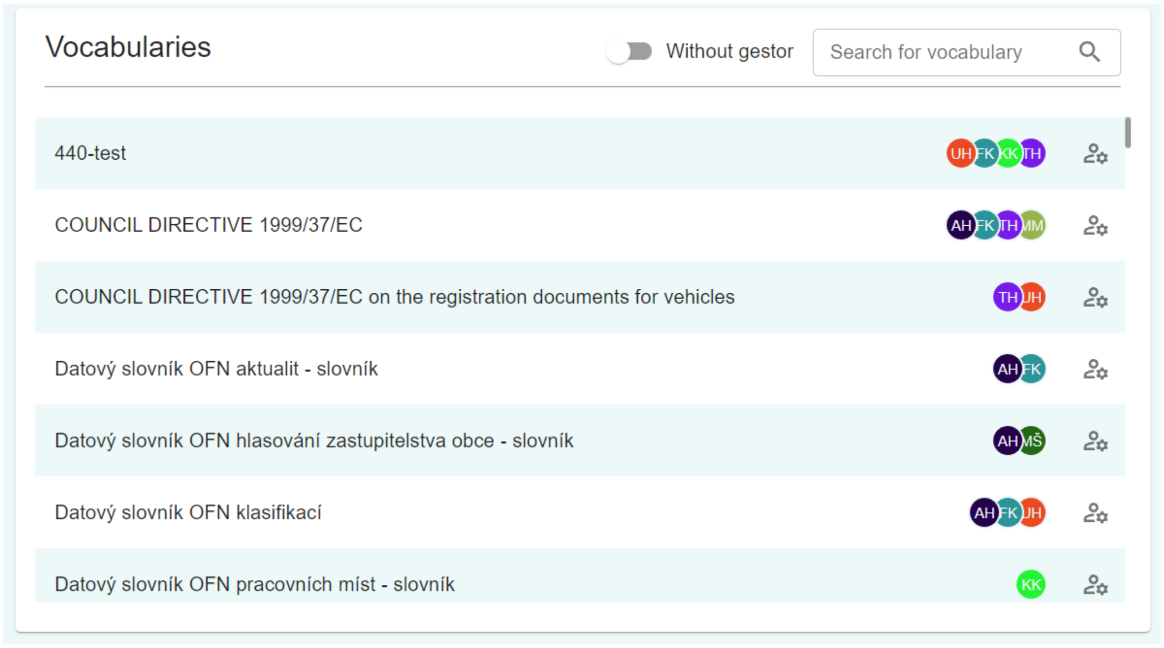
## B.2 Administrator panel


Pending requests
2

Assigned vocabularies
23/103

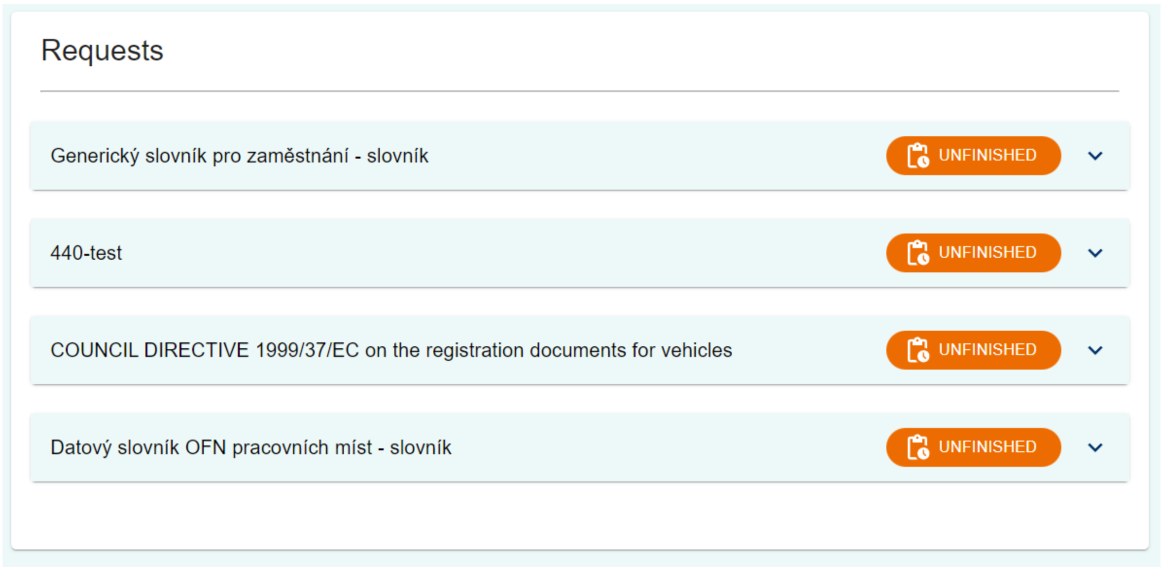
Administrators
1

**Figure B.3:** Administrator panel options


Vocabularies

| | |
|---|---|
| 440-test | UH FK KK TH |
| COUNCIL DIRECTIVE 1999/37/EC | AH FK TH MM |
| COUNCIL DIRECTIVE 1999/37/EC on the registration documents for vehicles | TH UH |
| Datový slovník OFN aktualit - slovník | AH FK |
| Datový slovník OFN hlasování zastupitelstva obce - slovník | AH MŠ |
| Datový slovník OFN klasifikací | AH FK UH |
| Datový slovník OFN pracovních míst - slovník | KK |

**Figure B.4:** Administrator panel vocabularies list



Requests

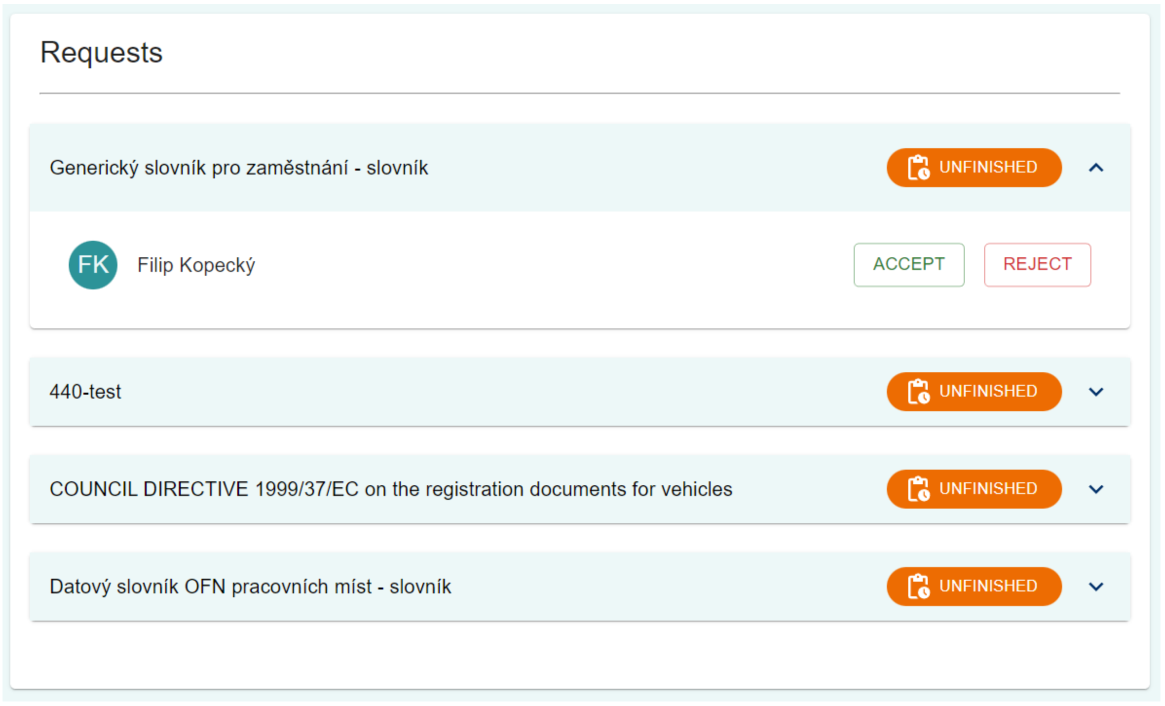
Generický slovník pro zaměstnání - slovník — UNFINISHED

440-test — UNFINISHED

COUNCIL DIRECTIVE 1999/37/EC on the registration documents for vehicles — UNFINISHED

Datový slovník OFN pracovních míst - slovník — UNFINISHED

**Figure B.5:** Administrator panel pending gestoring requests



Requests

Generický slovník pro zaměstnání - slovník — UNFINISHED

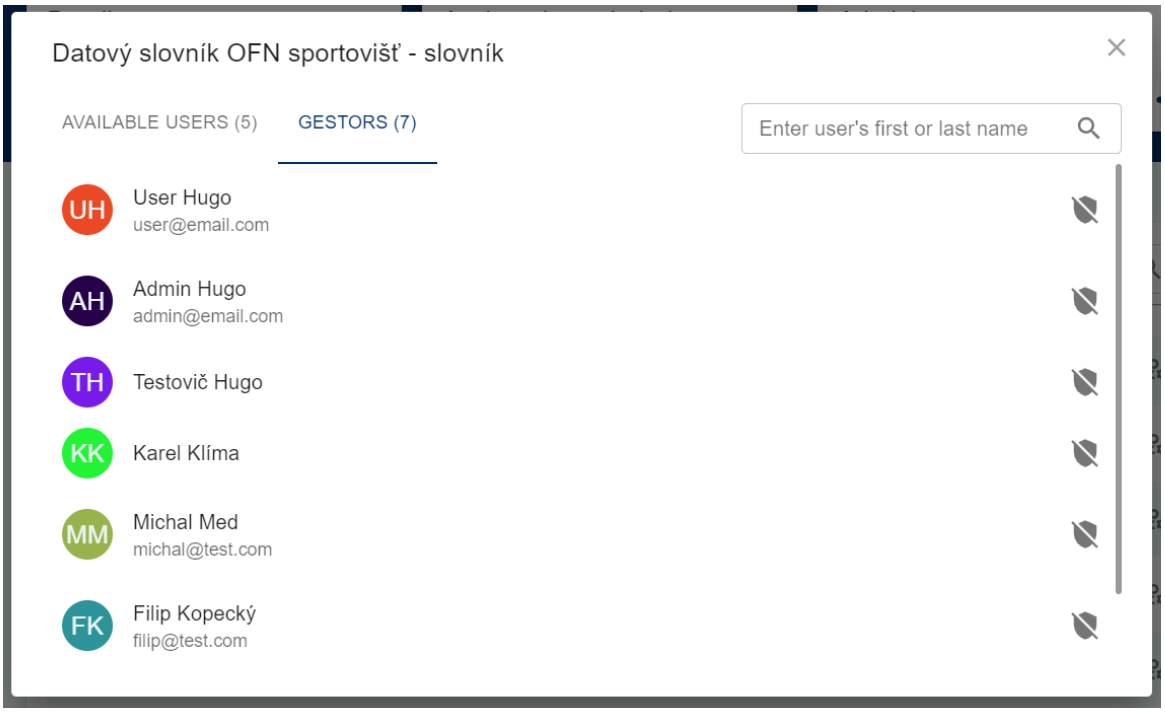
FK Filip Kopecký — ACCEPT REJECT

440-test — UNFINISHED

COUNCIL DIRECTIVE 1999/37/EC on the registration documents for vehicles — UNFINISHED

Datový slovník OFN pracovních míst - slovník — UNFINISHED

**Figure B.6:** Administrator panel pending gestoring requests expanded


Datový slovník OFN sportovišť - slovník

AVAILABLE USERS (5) GESTORS (7)

Enter user's first or last name

UH User Hugo user@email.com

AH Admin Hugo admin@email.com

TH Testovič Hugo

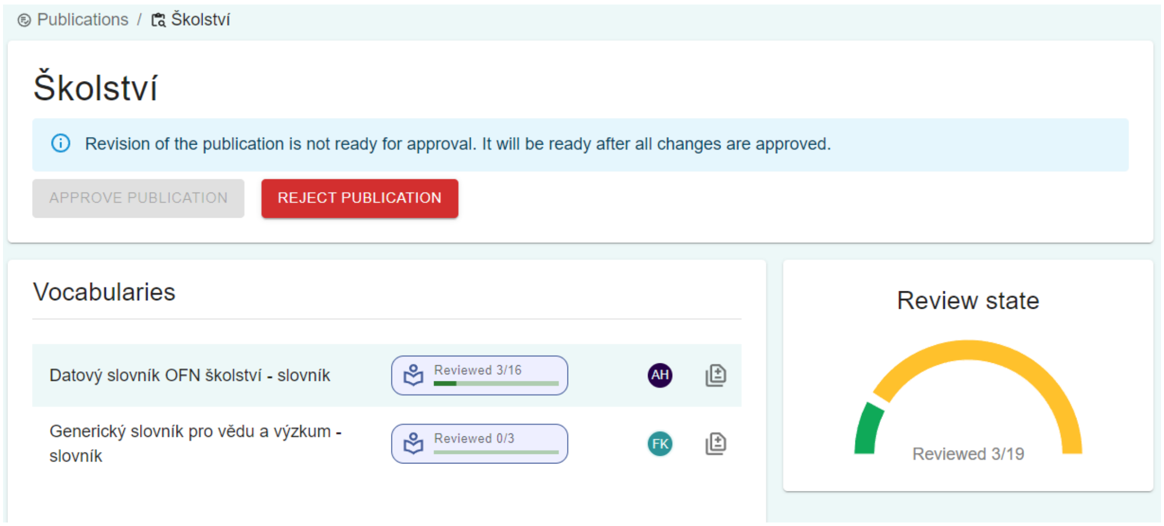
KK Karel Klíma

MM Michal Med michal@test.com

FK Filip Kopecký filip@test.com

**Figure B.7:** Removing gestors from a vocabulary

## B.3 Publications


Publications / Školství

Školství

Revision of the publication is not ready for approval. It will be ready after all changes are approved.

APPROVE PUBLICATION REJECT PUBLICATION

Vocabularies

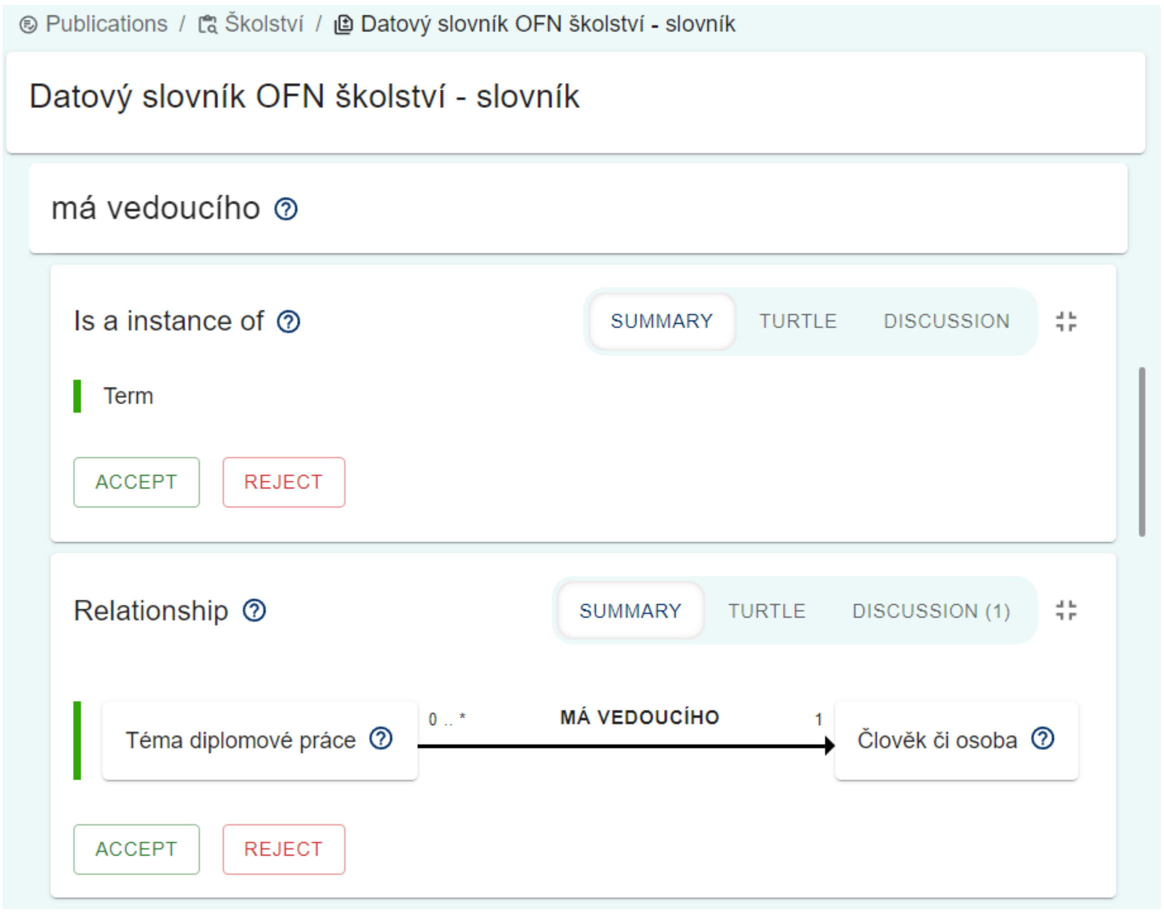
Datový slovník OFN školství - slovník — Reviewed 3/16 — AH

Generický slovník pro vědu a výzkum - slovník — Reviewed 0/3 — FK

Review state

Reviewed 3/19

**Figure B.8:** Publication overview


Publications / Školství / Datový slovník OFN školství - slovník

Datový slovník OFN školství - slovník

má vedoucího

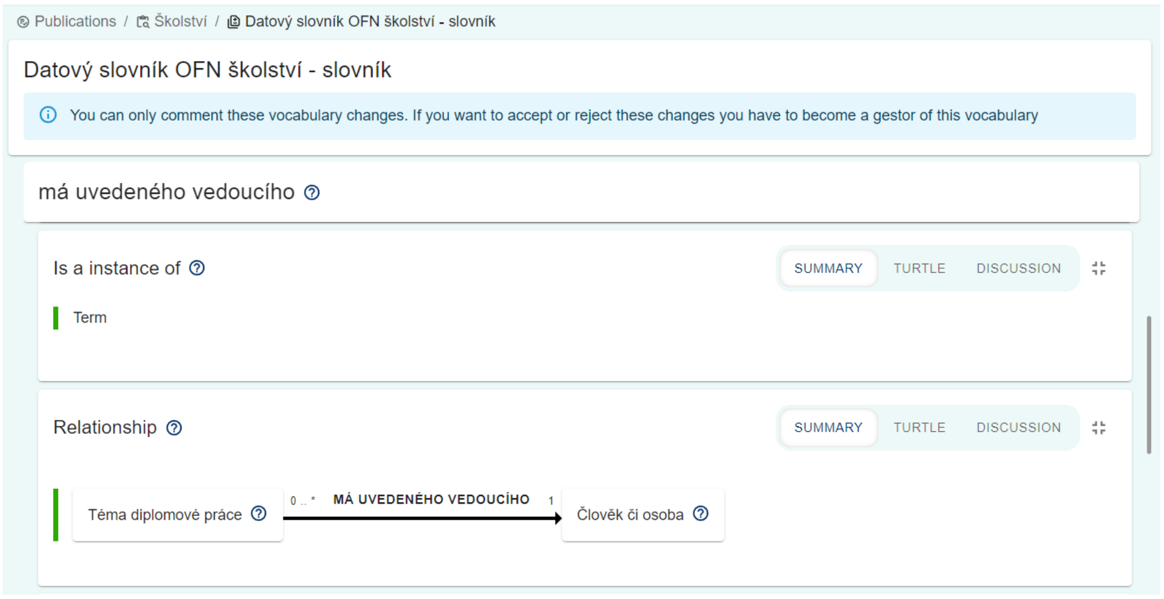
Is a instance of

Term

Relationship

Téma diplomové práce 0 .. * MÁ VEDOUCÍHO 1 Člověk či osoba

**Figure B.9:** Review of changes


Publications / Školství / Datový slovník OFN školství - slovník

Datový slovník OFN školství - slovník

You can only comment these vocabulary changes. If you want to accept or reject these changes you have to become a gestor of this vocabulary

má uvedeného vedoucího

Is a instance of

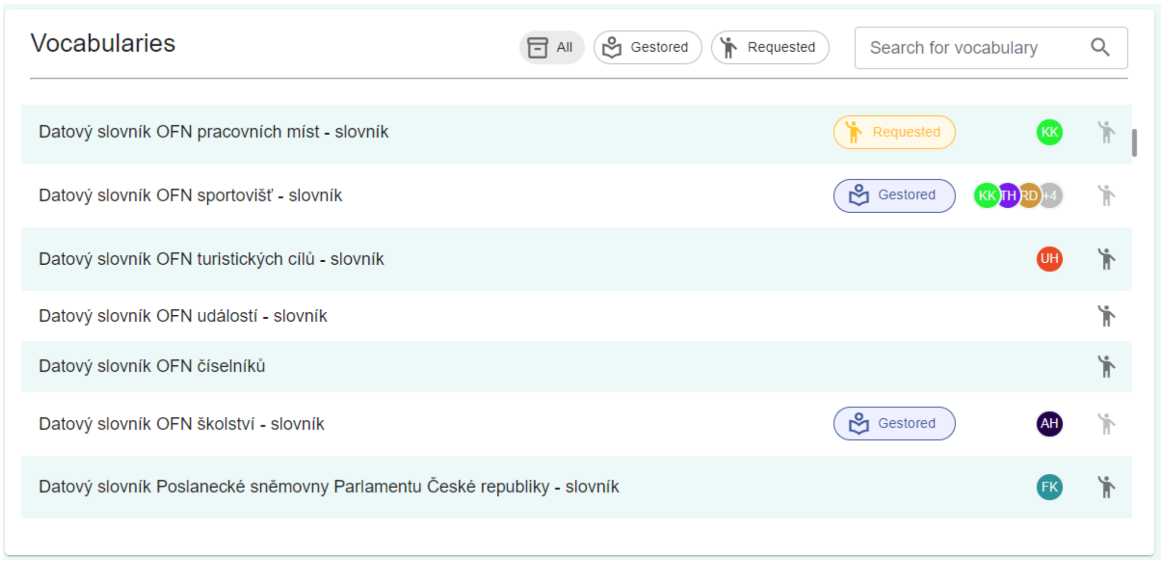
Term

Relationship

Téma diplomové práce 0 .. * MÁ UVEDENÉHO VEDOUCÍHO 1 Člověk či osoba

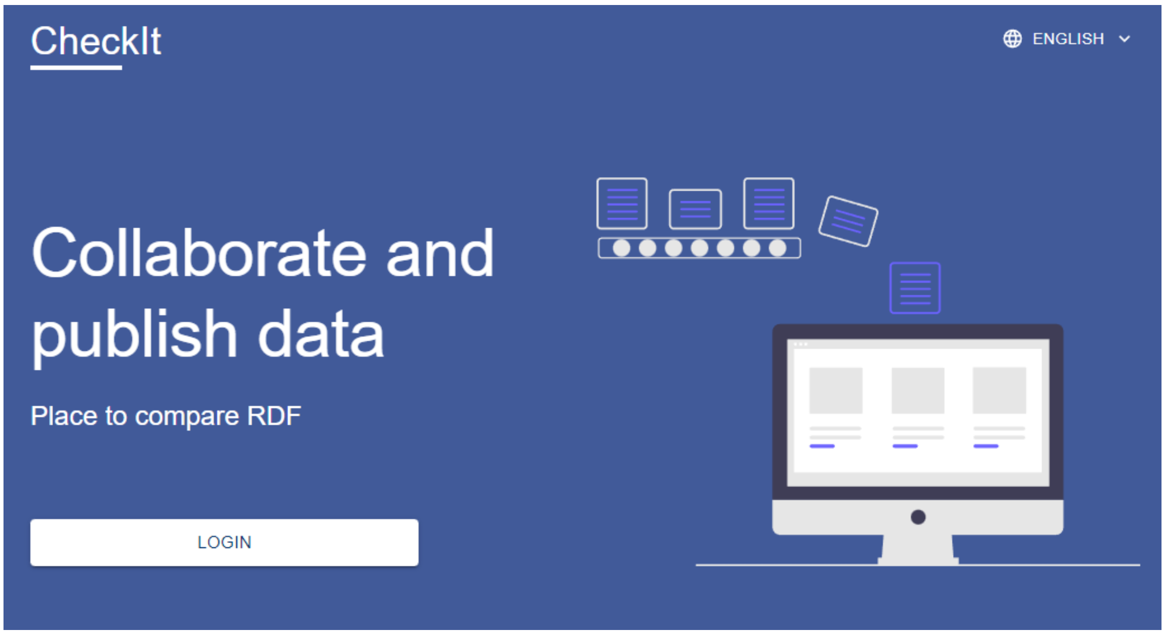
**Figure B.10:** Limited review access

## B.4 Others


Vocabularies

| | | |
|---|---|---|
| Datový slovník OFN pracovních míst - slovník | Requested | KK |
| Datový slovník OFN sportovišť - slovník | Gestored | KK TH RD +4 |
| Datový slovník OFN turistických cílů - slovník | | UH |
| Datový slovník OFN událostí - slovník | | |
| Datový slovník OFN číselníků | | |
| Datový slovník OFN školství - slovník | Gestored | AH |
| Datový slovník Poslanecké sněmovny Parlamentu České republiky - slovník | | FK |

**Figure B.11:** List of available vocabularies for users


CheckIt

Collaborate and publish data

Place to compare RDF

**Figure B.12:** Landing page

# Appendix C

# Content of electronic attachment

```
application.zip
└── checkit-ui-main .................... Root of the project
    ├── .github
    ├── nginx .......................... Nginx configuration
    ├── public ......................... Public assets
    ├── .src ........................... Source code of the application
    │   ├── api ........................ All API logic
    │   ├── app ........................ General app directory
    │   ├── assets ..................... Images
    │   ├── components ................. React components
    │   ├── hooks ...................... Custom React hooks
    │   ├── model ...................... Data model
    │   ├── slices ..................... Redux slices
    │   ├── store ...................... Redux store
    │   ├── translations ............... Localization files
    │   ├── App.tsx .................... Top level component
    │   ├── main.tsx ................... Entry point
    │   ├── styles.css ................. Additional styles
    │   ├── vite-env.d.ts
    │   └── window.d.ts
    ├── tests .......................... Unit tests - setup
    ├── .env ........................... Development environment variables
    ├── .env.production ................ Production environment variables
    ├── .eslintrc ...................... Linter rules
    ├── .gitignore
    ├── Dockerfile
    ├── README.md ...................... Installation guide
    ├── .config.js.template
    ├── index.html
    ├── package-lock.json
    ├── package.json
    └── tsconfig.json
```

```
│
├── tsconfig.node.json
└── vite.config.ts
```