

Bachelor Project



**Czech
Technical
University
in Prague**

F3

**Faculty of Electrical Engineering
Department of Computer Science**

Web browser plug-in for semantic vocabulary creation

Alan Buzek

**Supervisor: Ing. Petr Křemen, Ph.D.
May 2022**

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Buzek** Jméno: **Alan** Osobní číslo: **491979**
Fakulta/ústav: **Fakulta elektrotechnická**
Zadávající katedra/ústav: **Katedra počítačů**
Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Plugin do webového prohlížeče pro tvorbu sémantických slovníků

Název bakalářské práce anglicky:

Web browser plug-in for semantic vocabulary creation

Pokyny pro vypracování:

Semi-automatic creation of semantic vocabularies from documents can be tedious. During the process the created concepts are linked back to the parts of the document defining them. The goal of this thesis is to design, implement and evaluate a web browser plugin for annotating web documents using existing semantic vocabularies and allowing to create new vocabulary concepts from the document.

1. Become familiar with RDF, OWL, the Termlt system (<https://github.com/kbss-cvut/termit>) and the Annotace service (<https://github.com/kbss-cvut/annotace>).
2. Design and implement a web browser plugin that allows creating semantic vocabularies by semi-automatic annotation of web documents and robust annotation resolution in case a web document evolves. The plugin will use Annotace to provide automated suggestions and Termlt to manage the vocabularies.
3. Evaluate the robustness of the web browser plugin annotation of evolving web documents on selected examples from the Czech or international legislation.
4. Evaluate the usability of the web browser plugin by a user study with several vocabulary designers.

Seznam doporučené literatury:

- Martin Ledvinka, Petr Kremen, Lama Saeeda, Miroslav Blasko: Termlt: A Practical Semantic Vocabulary Manager. ICEIS (1) 2020: 759-766
- Web Annotation Vocabulary, R. Sanderson, P. Ciccarese, B. Young, Editors, W3C Recommendation, February 23, 2017, <https://www.w3.org/TR/2017/REC-annotation-vocab-20170223/>.
- OWL 2 Web Ontology Language Primer, P. Hitzler, M. Krötzsch, B. Parsia, P. Patel-Schneider, S. Rudolph, Editors, W3C Recommendation, October 27, 2009, <http://www.w3.org/TR/2009/REC-owl2-primer-20091027/>.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Petr Křemen, Ph.D. skupina znalostních softwarových systémů FEL

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

Datum zadání bakalářské práce: **07.02.2022**

Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Petr Křemen, Ph.D.
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

Datum převzetí zadání

Podpis studenta

Acknowledgements

I would like hereby thank my supervisor Ing. Petr Křemen, Ph.D. for his patient guidance, advice, and all the helpful feedback he has given me throughout my work on this thesis.

Declaration

I declare that this work is all my work, and I have cited all sources I have used in the bibliography.

Prague, May 20, 2022

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 20. května 2022

Abstract

The goal of this bachelor thesis is to design and implement a web browser plug-in for creating semantic vocabularies through the annotation of documents on the Web. The plug-in aims to provide robust annotation capabilities while integrating with the Annotace service for automatic annotation suggestions and the TermIt system for vocabulary management. After architecting and implementing the solution, the work evaluates the plug-in's effectiveness in evolving Web documents and conducts a usability study with users of the TermIt system.

Keywords: semantic web, browser extension, web annotations

Supervisor: Ing. Petr Křemen, Ph.D.

Abstrakt

Cílem této bakalářské práce je návrh a implementace pluginu do webového prohlížeče pro vytváření sémantických slovníků prostřednictvím anotací webových dokumentů. Účelem pluginu je poskytnout robustní možnosti anotování a zároveň být integrován se službou Annotace pro automatické návrhy anotací a systémem TermIt pro správu slovníků. Po návrhu architektury a implementaci řešení je součástí práce vyhodnocení efektivity pluginu v měnících se webových dokumentech a provedení studie použitelnosti s uživateli systému TermIt.

Klíčová slova: sémantický web, plug-in do prohlížeče, webové anotace

Překlad názvu: Plugin do webového prohlížeče pro tvorbu sémantických slovníků

Contents

1 Introduction	1		
1.1 Annotations	1		
1.2 TermIt	1		
1.3 Motivation	2		
1.4 Goal	2		
2 Related work	5		
2.1 Vocabulary management	5		
2.1.1 Pool Party Semantic Suite	5		
2.2 Web annotation extensions	6		
2.2.1 Weava Highlight	6		
2.2.2 Diigo Web Collector	7		
2.2.3 Hypothes.is	7		
2.3 Summary	7		
3 Background	9		
3.1 Semantic Web	9		
3.1.1 RDF	9		
3.1.2 OWL	10		
3.1.3 SKOS	10		
3.2 TermIt	10		
3.2.1 TermIt's current architecture	11		
3.2.2 Annotation workflow	13		
3.2.3 Annotator	13		
3.3 Used technologies	14		
3.3.1 React	14		
3.3.2 Browser extensions	14		
4 Architecture	17		
4.1 Requirements	17		
4.1.1 Functional requirements	17		
4.1.2 Non-functional requirements	19		
4.2 TermIt-wide architectural changes	20		
4.2.1 Annotation resolution redesign	20		
4.2.2 TermIt Server changes	23		
4.2.3 Annotations as Term occurrences	24		
4.2.4 TermIt UI changes	24		
4.2.5 Annotace changes	25		
4.3 Extension architecture	26		
4.3.1 Key factors	26		
4.3.2 Content Script	27		
4.3.3 Browser storage	27		
4.3.4 Background script and syncing with TermIt UI	28		
4.4 Extension's annotator	28		
4.4.1 Basic structure	29		
4.4.2 Control flow and state management	29		
4.4.3 The case for Redux	29		
4.5 Annotation resolution	30		
4.5.1 Initial annotation flow	30		
4.5.2 Dropped annotation suggestions	31		
4.5.3 Ensuring selector robustness	31		
5 Implementation	33		
5.1 React justification	33		
5.2 TermIt UI code reuse	33		
5.3 Sidebar performance optimizations	34		
5.4 Vocabularies caching	34		
5.5 Website URL matching	35		
5.6 Hypothes.is inspiration	35		
5.7 Annotation selector caveats	36		
5.8 Mark.js	36		
6 Evaluation	39		
6.1 Annotation resolution robustness	39		
6.1.1 Unchanged Web pages	39		
6.2 Evolving Web pages	41		
6.2.1 Expectations	41		
6.2.2 Testing	43		
6.3 User study	44		
6.3.1 Testing environment	45		
6.3.2 User group	45		
6.3.3 Test scenario and feedback	45		
6.3.4 User study results	46		
7 Conclusions	47		
7.1 Evaluation	47		
7.2 TermIt Annotate and its future	48		
7.2.1 Chrome Web Store	48		
7.2.2 Future work	48		
Bibliography	51		
A Tutorial	55		
B Proof of concept	61		
B.1 Functionality	61		
B.2 Tracked metrics	61		
B.2.1 Elements selection	61		
B.2.2 Term annotation	62		
B.2.3 Testing	62		

C Test scenario	65
D Description of electronic attachmens	67

Figures

1.1 Screenshot from the TermIt’s annotation page	3	A.5 5. step of the tutorial – See your annotations in time	58
2.1 Pool Party Semantic Suite’s text annotation [9]	6	A.6 6. step of the tutorial – Take full advantage of TermIt’s semantic vocabulary platform	58
2.2 Annotations in Weava Highlight [11]	7	A.7 1.step of the tutorial, a brief introduction to TermIt Annotate.	59
3.1 An example of two SKOS concepts being hierachically related [23]	11	B.1 Screenshot of using the extension annotations on the page https://www.zakonyprolidi.cz/cs/2006-499	62
3.2 Current TermIt architecture	11		
3.3 Annotace service input and output example with a the word “page” annotated using RDFa	12		
3.4 Basic annotation workflow in TermIt UI	13		
3.5 Browser extension architecture [27]	15		
3.6 manifest.json example	16		
4.1 New TermIt architecture	21		
4.2 Annotation selector generation algorithm	23		
4.3 TermIt’s Data Model after our modification	25		
4.4 Browser extension architecture	26		
4.5 Extension’s annotator architecture	28		
4.6 Annotation flow from browser extension	31		
6.1 Example of an annotation text not found after the document has evolved	42		
6.2 Example of an offset mismatch selector failure.	43		
7.1 The Chrome Web Store listing of TermIt Annotate	49		
A.1 1. step of the tutorial – A brief introduction to TermIt Annotate	56		
A.2 2. step of the tutorial – Start page annotation through the sidebar	56		
A.3 3. step of the tutorial – Confirm suggested annotations and create your own	57		
A.4 4. step of the tutorial – Use sidebar to keep things in grip	57		

Tables

6.1 Test results on evolving Web pages.....	44
B.1 Proof of Concept aggregate selector test results.	63



Chapter 1

Introduction

Vocabularies have proven to be a very useful tool over the years. With the rise of information technology, its capabilities were greatly amplified. Searching through a vocabulary is almost instantaneous and widely available, making all the information one could ever need about a word or a phrase readily available just a click or two away.

Furthermore, there can be more to vocabularies than just a list of words and their definitions. Terms can be enriched with meaningful metadata and linked together through defined relationships, forming a knowledge graph. More importantly, annotations can also connect terms with various text resources. Annotating a term's occurrence within a piece of text may seem like a seemingly trivial task. However, annotations can be tremendously powerful. They can facilitate terms definition and disambiguation and help infer new knowledge about both documents and terms. The importance of vocabulary terms' annotations expands even further if we consider the Web, as this vast source of documents we can extract knowledge from.



1.1 Annotations

Annotations are a way to add extra information to text or other resources and can be in the form of highlighting, underlining, adding additional notes, or other ways. Of course, annotations have existed on physical written media for a long time. They have, in fact, become quite common in texts as early as in the Medieval era [1].

With the advent of modern computers, the impact of annotations has increased manifold. Digital annotations have since seen wide use in fields such as linguistics, natural language processing (NLP) [2] or education [1]. Most importantly, as this work shows, annotations can be used with great efficacy to create semantic vocabularies on the Web.



1.2 TermIt

TermIt is an open-source vocabulary management and a glossary editing information system based on Semantic Web technologies. It aims to provide

an easy way for domain experts to create and manage terms without requiring ontology expertise [3].

Besides vocabulary management, it also supports annotating vocabulary terms' occurrences and definitions in HTML documents. Annotations provide a way to link resources with terms, easily create new terms, and later use this information to enable more accurate searching through documents and infer further knowledge of both documents and terms.

TermIt is being developed by the Knowledge Based Software System Group at the Czech Technical University in Prague. It has seen adoption in public administration [4], healthcare, and urban planning [3].

1.3 Motivation

TermIt allows the user to upload a new document and run automatic text analysis on it, annotating it with suggestions of new terms to create and possible occurrences of existing terms. Afterward, the user can easily create new terms and confirm or decline the suggestions of existing ones. This semi-automatic approach reduces the user's effort to create highly accurate semantic vocabularies. An example from the resource annotator in TermIt is to be found in Figure 1.1.

Despite this great functionality offered, the resource annotation workflow in TermIt has some significant shortcomings, namely:

1. Users first have to upload an HTML document into TermIt to start annotating it.
2. Storing a given Web page onto the user's computer and only then uploading it into the application is a cumbersome and error-prone process that may prove challenging for many users.
3. This way, the page's layout and styling are frequently broken, so it is hard to compare the uploaded document with the original copy.
4. Web documents typically evolve over time. There is no way to reflect those changes in already annotated documents with the current approach.
5. If the user encounters hyperlinks to different pages they would like to follow and continue annotating, they will be taken outside of TermIt's application again, having to repeat the entire process.

1.4 Goal

Considering the above issues, naturally, a thought comes to mind: *Would there be a way to provide a more immersive, user-friendly, and efficient experience?* We have pondered this question and arrived at a clear conclusion: all the above issues can be solved with a browser extension that will enable annotations

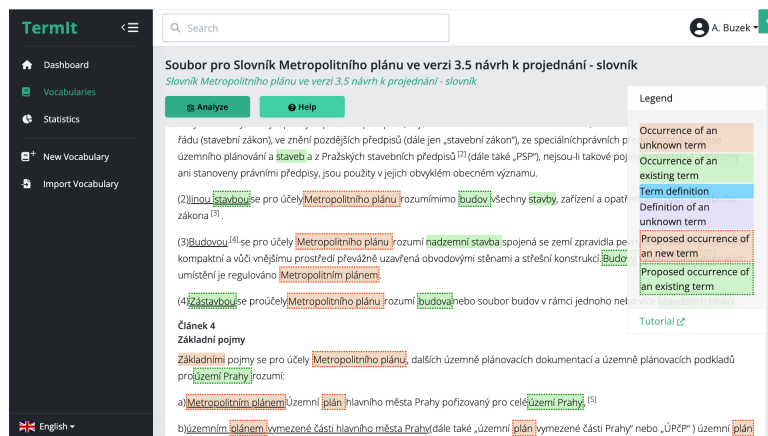


Figure 1.1: Screenshot from the TermIt's annotation page

seamlessly right on any Web page. Moreover, it can do so without the user first having to go into the original TermIt application while still taking full advantage of key features such as automated text analysis.

Browser extensions, also called plugins or add-ons, are programs that modify or extend the web browser's functions [5]. Extensions can take advantage of different capabilities allowed by the browser to, for example, modify the color of bookmarks, alter the content of viewed pages, or otherwise enhance the web browsing experience [6].

At a time when a fair amount of client software essentially runs just in web browsers in the form of websites and web applications [7], browser extensions can prove to be tremendously valuable for the end-users. Moreover, they generally come at a relatively low cost for the extension developer and easy distribution through official marketplaces offered by browser vendors [8].

Thus, this work aims to design and implement a browser extension called *TermIt Annotate* for creating semantic vocabularies through annotations on Web pages. It aims to be lightweight, easy to use, and available for a wide range of users. With that in mind, it must provide a consistent user experience with that of TermIt's original web application. Therefore, the extension will narrowly focus on the annotation piece of functionality while linking to the original application for other use cases, such as browsing existing vocabularies and terms, seeing statistics, login and registration, and others. Thus, a secondary but just as essential goal of this work is to extend the necessary TermIt system's software components to accommodate the needs of the implemented extension.

Chapter 2

Related work

We will first briefly examine select general semantic vocabulary management tools and then dive specifically into browser extensions supporting website annotation use cases to describe related work.

2.1 Vocabulary management

There are several vocabulary management solutions, many of which also support Semantic Web technologies. However, their common pitfall is that, unlike TermIt, they require ontology expertise, presenting a barrier for entry for domain experts [3]. Moreover, very few such systems provide any resource annotation capabilities, and none offer a browser extension to enable creating semantic vocabularies on the Web. In terms of resource annotation, we have found only one system that is relevant enough to be examined for our work, and it will be discussed as follows.

2.1.1 Pool Party Semantic Suite

Pool Party Semantic Suite [9], as the name suggests, provides a whole suite of tools built around ontologies and the Semantic Web. Most notably, it can manage controlled vocabularies, such as taxonomies or thesauri. Like TermIt, automatic text analysis on uploaded documents is possible through its “Corpus Analysis” feature to speed up vocabulary creation. A wide range of text file formats is supported, even the ability to insert a website’s URL to analyze. The vocabulary creation is also semi-automatic, in the sense that as a result of the text analysis, suggestions of existing and new concepts are shown in the uploaded file. The user can confirm concept (terms) occurrence suggestions and can do so in bulk, which speeds up the workflow. While the annotated document can be displayed within the user interface as shown in Figure 2.2, occurrences of a single term in the text can only be edited all at once, and there is no possibility of creating one’s own annotations after the text is analyzed automatically.

The tool is quite extensive, making it easy for a non-expert user to get lost. Moreover, its resource annotation capabilities are noticeably optimized for a workflow where a vast number of documents are uploaded and analyzed

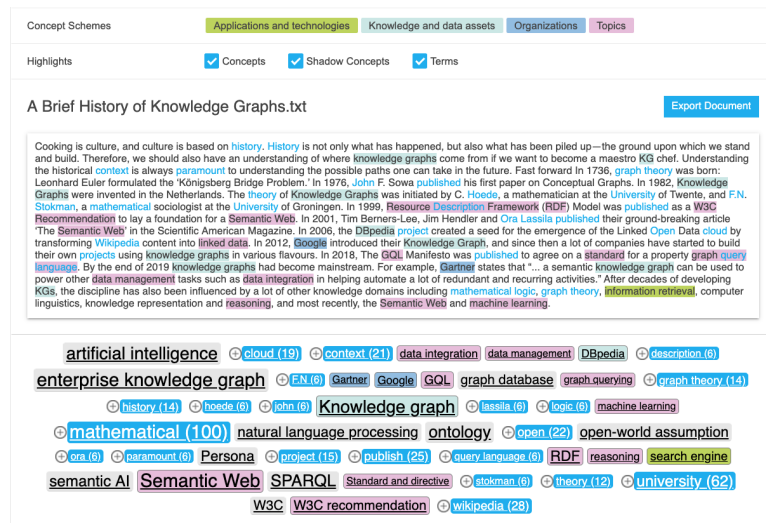


Figure 2.1: Pool Party Semantic Suite’s text annotation [9]

automatically. Then its results are examined manually in bulk, undoubtedly resulting in lesser accuracy. Consequently, this approach makes it a lot less suitable for our use case of creating highly accurate vocabularies.

2.2 Web annotation extensions

There exists a relatively large number of different browser extensions that offer annotations. Google Chrome’s web store even has a dedicated category titled “Annotate the Web” [10]. However, none can be used for generic vocabulary management, let alone take advantage of Semantic Web technologies like TermIt. Instead, most focus on general-purpose annotations for use cases such as research, note-taking, or education. As follows, we have selected three extensions to examine, based both on their popularity with users and the similarities with our use case.

2.2.1 Weava Highlight

Weava Highlight [11] is a Google Chrome extension that provides an easy and intuitive way to annotate any page elements and add notes to each annotation. Annotations are each put into a specific collection. Only those belonging to the currently selected collection will appear on web pages, allowing for switching between different browsing contexts.

Upon visiting its website at <https://www.weavatools.com/> and signing into one’s account, one can view and edit all previously saved annotations. In addition, Weava Highlight offers the option to collaborate on an annotation collection and generate citations.

- **Number of installs (Chrome web store):** 700,000+

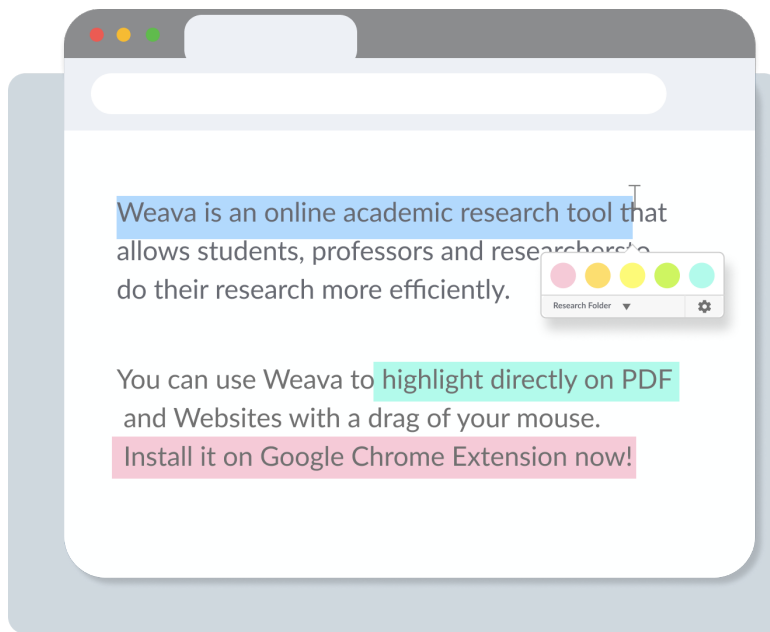


Figure 2.2: Annotations in Weava Highlight [11]

■ 2.2.2 Diigo Web Collector

Diigo Web Collector [12] does text annotation and supports other use cases such as screenshots and bookmark capture. It also comes with an "outliner" tool, allowing drag-and-drop annotations and arbitrarily texts to be captured while browsing the Web.

- **Number of installs (Chrome web store):** 300,000+

■ 2.2.3 Hypothes.is

Hypothes.is [13] takes a different approach to the two extensions mentioned above. Annotations created by users of the Hypothes.is extension are shared with other users by default, displaying it to anybody with the extension installed and comes to the same page. Hypothes.is is open-source and aims to bring back the old idea of annotations as a shared conversation layer on top of the Web itself. This feature was present in one of the first browsers, Mosaic [14]. However, it was ultimately cut due to technological challenges it brought at the time [15] and never re-appeared in any major browser again.

- **Number of installs (Chrome web store):** 200,000+

■ 2.3 Summary

From our research of related work, it is apparent that none of the examined solutions fit what we are trying to achieve with our browser extension.

Despite the Pool Party Semantic Suite offering a competent set of tools, including extensive resource annotation functionality, it does not fit our needs very well. The examined browser extensions are good at in-browser annotation functionality. However, their vocabulary management capabilities are non-existent, nor can they be integrated with any existing systems, let alone TermIt.

Thus, the browser extension we will implement, called TermIt Annotate, will be a unique, one-of-a-kind tool that could not only improve current TermIt users' annotation workflows. It could also very likely draw in new users interested in vocabulary management and annotation who have not been able to do so due to a lack of such a tool.

Chapter 3

Background

This chapter introduces the necessary background surrounding our work. Firstly, as the TermIt system is built on them, the Semantic Web technologies will be introduced. Afterward, putting those introduced concepts into practice, we will then be able to describe and understand the current TermIt system. Understanding all TermIt's components will be fundamental for our work. Finally, the necessary technical background surrounding Web browser plugin development and related used technologies will be introduced.

3.1 Semantic Web

The Semantic Web is an extension of the World Wide Web defined by a set of standards published by W3C [16]. Its main goal is to make data on the Web machine-readable. Traditionally, documents on the Web have only pointed at other documents through hyperlinks. The Semantic Web aims to connect the Web on the level of individual pieces of data that have traditionally been scattered in HTML documents and only understood by humans, thus making information machine-readable and connected. To achieve this, the Semantic Web is comprised of a set of technologies specified by its standards, namely RDF, RDFS, SKOS, OWL, and SPARQL [17], which will be described in briefly introduced in the following sections.

While the original goal of the Semantic Web as a replacement of the traditional World Wide Web as we know it *remains largely unrealized* [18], its technologies are still tremendously beneficial in many areas. Some of its most common applications are knowledge organization, ontologies [19] and publishing linked data [20].

3.1.1 RDF

RDF [21] stands for Resource Description Framework, and it is a data model describing concepts through so-called triples. A triple is an expression containing a subject, predicate, and object. The subject denotes the resource itself, and the predicate denotes the relationship between the subject and the object. The object describes the relationship itself. A triple is, therefore, a simple statement about a resource. For example, to represent the statement,

“Patrick is married to Paula” as an RDF triple is to “Patrick” be the subject, “is married to” be the predicate, and “Paula” be the object. This way, a triple is readable by both machines and humans alike.

RDF serves as the backbone upon which other Semantic Web technologies are built. RDFS, also called RDF Schema, is data modeling vocabulary for RDF, serving as its semantic extension. In addition, it provides the necessary elements for ontology description.

■ 3.1.2 OWL

The Web Ontology Language (OWL) [22] is a language to describe and represent knowledge used to create ontologies. Ontologies are formal specifications of concepts shared between humans and machines. They help define concepts in a particular domain and infer new knowledge based on set rules. Because OWL is a logic-based language, it can be used computationally to verify knowledge consistency and infer implicit knowledge.

■ 3.1.3 SKOS

SKOS [23] or Simple Knowledge Organization System specifies the representation of knowledge organization systems in the form of controller vocabularies such as thesauri, taxonomies, or glossaries. The core elements of a SKOS vocabulary are concepts organized within concept schemes. Concepts are identified with URIs and have designated labels that can be multilingual and contain various types of additional information. In addition, concepts can be semantically related to each other, as shown in Figure 3.1. One of the main objectives of SKOS is to provide a simple way to publish vocabularies as linked data [20]. Linked Data represents a type of structured data that is linked through the use of RDF(S) and URIs, adhering to the objects of the Semantic Web of machine-readable interconnected data described earlier in this chapter.

■ 3.2 TermIt

As outlined above, TermIt is a vocabulary manager for domain experts. It is compliant with the SKOS standard. More specifically, SKOS concept schemes represent its vocabulary. Each term represents a concept and contains rich metadata describing its label, definition, or other notes, all in line with properties defined by SKOS. Significantly, terms can be related to each other, forming a hierarchical tree, exploiting the SKOS-defined properties such as narrower, broader, or related. Also, entire vocabularies can be related to each other, using the same hierarchical relations. The system uses a triple store, a type of database for graph data stored as RDF triples. TermIt supports RDFS and OWL such that the knowledge can be inferred based on the defined rules.

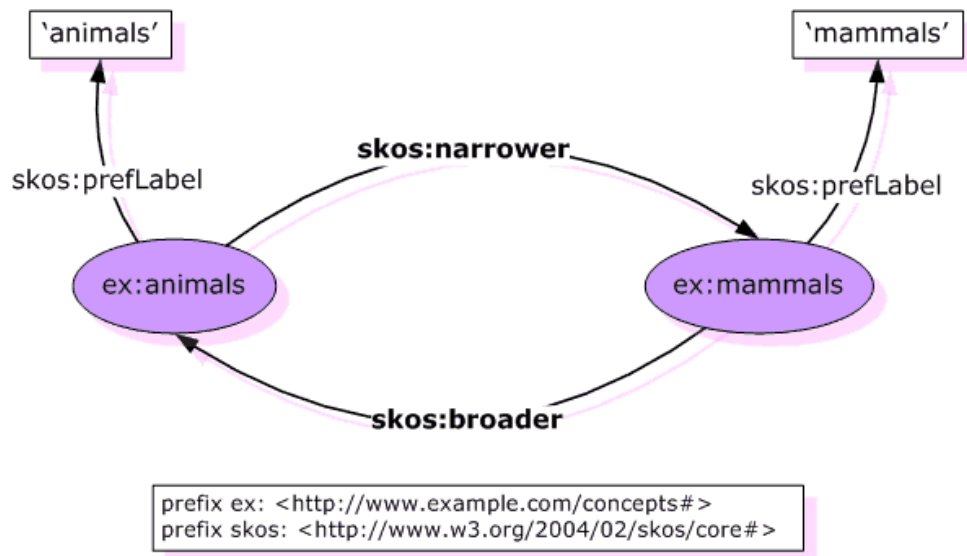


Figure 3.1: An example of two SKOS concepts being hierachically related [23]

As a result of this Semantic Web foundation, terms created and managed within TermIt’s vocabularies contain rich interconnected metadata. This allows for accurate searching based on the term’s ontological relations and, importantly, serves to be able to disambiguate similar terms. Furthermore, the end-users can enjoy these capabilities while not needing any prior ontological background. They are effectively fully abstracted from the ontological internals of the application, while still reaping its benefits.

■ 3.2.1 TermIt’s current architecture

This section introduces, on a high level, the current software architecture of the TermIt system and its directly related and dependent components. To get a quick overview of the architecture, its diagram is shown in Figure 3.2.

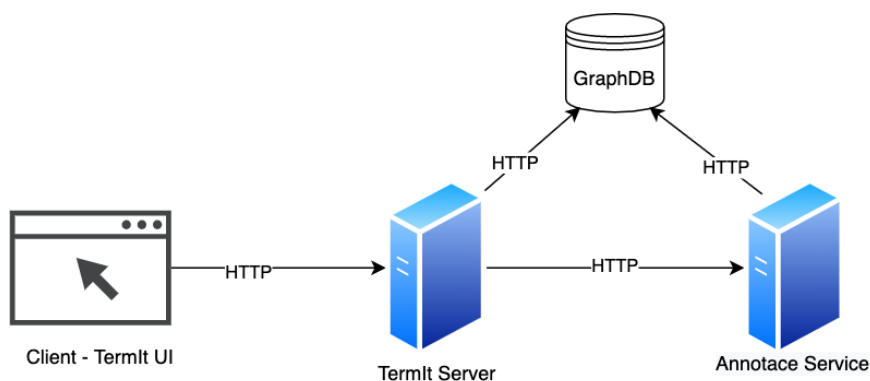


Figure 3.2: Current TermIt architecture

Input HTML:

```
<div>
  This page is an example.
</div>
```

Output HTML:

```
<div>
  This
  <span
    about="_:7d5c-3"
    property="ddo:je-výskytem-termu"
    content="page" typeof="ddo:výskyt-termu"
    score="1.0">
    page
  </span>
  is an example.
</div>
```

Figure 3.3: Annotace service input and output example with a the word “page” annotated using RDFa

■ TermIt Server and UI

The TermIt system itself has two main components – TermIt Server implemented as a Java application using Spring Boot¹, and TermIt UI implemented as a React single-page Web application. These two components communicate through a REST API and exchange data in the JSON or JSON-LD² format. TermIt’s data is stored on the server in the form of RDF triples using a GraphDB[24] database. To achieve mapping of underlying RDF triples to Java objects, the JOPA[25] library is used.

■ Annotace

Annotace is a stateless web service, independent of other components within TermIt, not requiring authentication. It provides a REST API with a single endpoint that automatically annotates HTML documents with suggestions based on selected vocabularies and commonly used keywords. It uses third-party libraries for term lemmatization and keyword extraction based on statistical analysis. Found annotation suggestions are inserted directly into the input HTML document as RDFa`https://rdfa.info/` tags, as shown in Figure 3.3. The document modified in this way is then returned from the endpoint.

¹<https://spring.io/projects/spring-boot>

²<https://json-ld.org/>

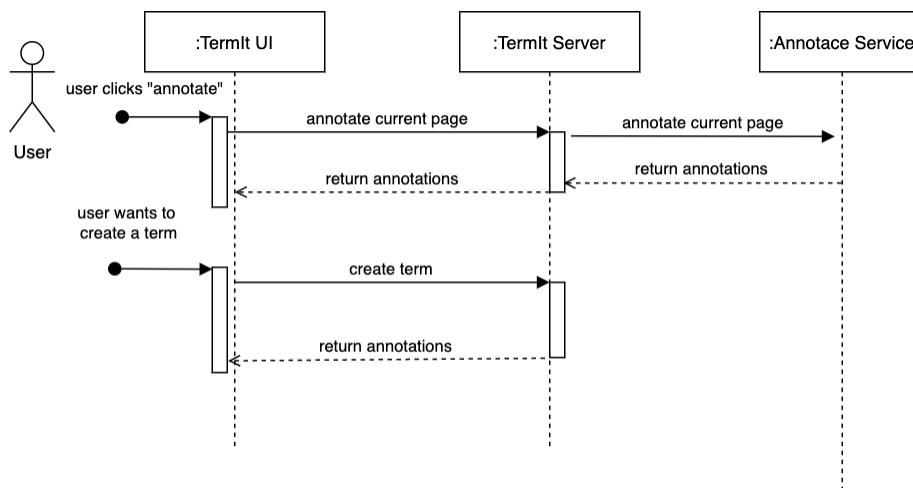


Figure 3.4: Basic annotation workflow in TermIt UI

3.2.2 Annotation workflow

After uploading a document to TermIt UI, the user can initiate the annotation process. The UI first calls TermIt Server, which then makes a request to Annotace. The annotated HTML document is then returned to the server, where annotations are parsed and saved as term occurrences into the database.

Further modifications of existing annotations or adding new annotations are done by directly modifying the HTML document. In most cases, any of these actions will not modify or create any TermOccurrence instances in the database.

A basic scenario of annotating a document and later creating a term from it is shown in Figure 3.4.

3.2.3 Annotator

The TermIt UI's Annotator has a dedicated React component within the application and works as follows. First, the HTML of the annotated document is requested from the server and rendered as a set of React components. Then, importantly, each existing annotation is rendered as its component within the React tree, which is shown overlaying the text if the annotation is clicked.

The overlaying popup can assign an annotation to a term, and the annotation thereby becomes the term occurrence of that term. Moreover, it can delete annotations or create entirely new terms and assign those to annotations. Besides, there are other components relevant to the page, such as a Modal window for creating a term, another popup component to appear above selected text to select the purpose of the selection, or a dropdown to select which vocabulary to get annotation suggestions for.

■ 3.3 Used technologies

Having introduced the Semantic Web and the TermIt system, we now have a solid knowledge foundation for our work. Before proceeding, however, we must also get familiar with the technologies used in the browser extension that will be designed and implemented.

■ 3.3.1 React

React [26] is a front-end JavaScript library for creating user interfaces created by Jordan Walke and now developed by Meta. Its key feature is a concept called Virtual DOM, a layer of abstraction on top of the browser's native Document Object Model (DOM). It means that the state of the UI is kept in memory and synchronized into the browser DOM on updates through a process called reconciliation. Reconciliation compares the in-memory (virtual) state with the state of the actual DOM and then selectively updates the DOM only where necessary, maximizing performance. Importantly, virtual DOM allows React to have a declarative API, abstracting the developer away from complicated DOM updates [26].

React code is split into components, which are reusable pieces of the user interface defined as JavaScript functions or classes. Each component has its own encapsulated state that can be defined and updated over time. The components' markup is written using JSX, an HTML-like syntax used within JavaScript. A component can have any number of child components, to which state can be passed down as so-called props, named properties for the child component to access.

■ 3.3.2 Browser extensions

As extensions run in the browser, they are developed by using standard Web technologies – HTML, CSS, and JavaScript. The following sections will introduce the files and concepts relevant to browser extension development [6]. Note that due to slight incompatibilities between extensions in different browsers, the following section and the rest of this work will primarily focus on Google Chrome's specification of extensions, knowing that there might be slight incompatibilities with other browsers here and there.

■ Manifest file

Browser extensions vary in their structure depending on the use case, but a core and mandatory piece of every browser extension is a manifest file. It is a JSON document that declares basic information about the extension, such as its name, description, version number, and different browser APIs and permissions that will be leveraged [27]. See an example manifest.json file in Figure 3.6.

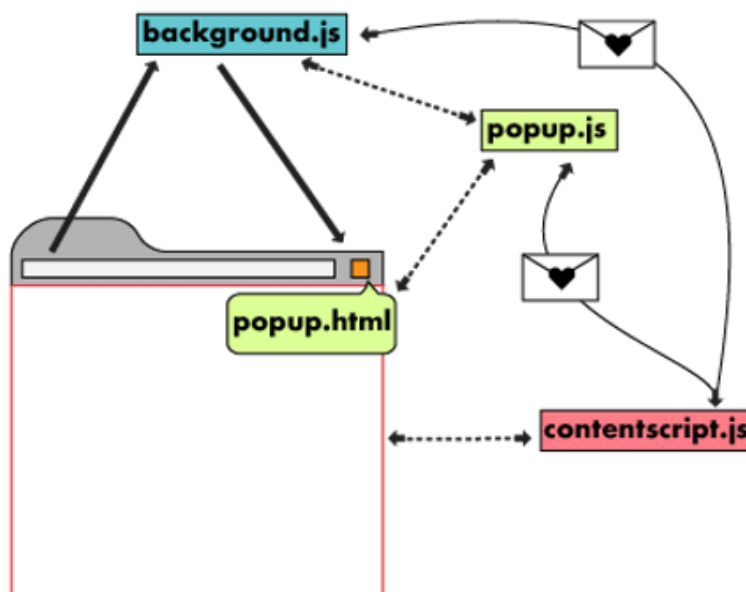


Figure 3.5: Browser extension architecture [27]

■ Background script

The background script is the extension’s event handler in the form of a service worker. It runs once an event has been fired and then goes dormant after handling it [27]. Unlike other extension scripts, it is not directly tied to any piece of the user interface.

Note that, the background script has seen very recent changes in its capabilities with the introduction of the Manifest v3 specification by Google Chrome. Previously, a persistent background page would be open throughout the entire browser session, sometimes resulting in unnecessary resource allocation [28]. Support for the previous versions of the manifest specification is still partly there for existing plugins but is to be phased out entirely by 2023 [29]. Other browser vendors, such as Mozilla Firefox, have announced plans to support Manifest v3 soon [30], but are not compatible as of now (05/2022).

■ Content script

Content script is a piece of code injected into pages the user visits [27]. Its capabilities include reading and modifying the DOM, registering event listeners, and otherwise interacting with the currently visited page to which they have access. Essentially, the content script’s code can perform similar to what JavaScript downloaded from the host page can do. It is worth noting that both the host page’s script as well as the content script run in so-called “isolated worlds”, meaning that their JavaScript environments cannot affect each other [31]. The extensions’ host permissions declaration in the manifest file determines into what pages (if any) is the content script injected.

```
{
  "name": "Example extension",
  "description": "This is an example description.",
  "version": "1.0",
  "manifest_version": 3,
  "icons": {
    "16": "icon_16.png",
    "32": "icon_32.png"
  },
  "background": {
    "service_worker": "background-script.js"
  },
  "permissions": [],
  "host_permissions": ["https://*.my-page-example.com/*"],
  "action": {
    "default_icon": "icon_16.png",
    "default_popup": "popup-page.html"
  }
}
```

Figure 3.6: manifest.json example

■ Browser action and popup

Each extension is required to have an icon [27], which will appear in the browser toolbar, typically in the top right of the browser window. This icon is sometimes also referred to as the browser action.

Upon the user clicking the browser action, typically a popup appears, with a popup script loaded into it [32]. While an extension’s popup window represents a common way for users to interact with extensions, it is not required per se. Users’ click on the browser action icon can be handled by an event listener and trigger a different arbitrarily action instead. One disadvantage of using the popup is that it cannot be opened programmatically, rather only when the user manually clicks the extension icon.

■ Message passing

Background, content, and other extension scripts can communicate through a mechanism known as “message passing”. Event listeners need to be registered first in a receiving script, and after that, a message can be sent from other scripts. This communication is handled asynchronously via JavaScript callbacks or promises. Note that only JSON-serializable objects can be sent [33]. Importantly, a website running within the same browser as the extension can also exchange messages with it, provided that the website initiates the communication first.

Chapter 4

Architecture

We will first describe the requirements we have identified for our work in the following section.

Afterward, the architecture of our solution will be presented in three separate sections, going from the highest level to the lowest. On the level of the whole TermIt system and its components, we will discuss the significant architectural adjustments required to accommodate our browser extension, especially its annotation needs. Afterward, on a level lower, the architecture of the extension itself will be presented. We will then present the design of an Annotator component that will be contained within the browser extension, presenting its most significant part.

Finally, we will circle back to annotations themselves, detailing the new annotation workflow, and the related design decisions.

4.1 Requirements

This section lists the functional and non-functional requirements for the browser extension.

4.1.1 Functional requirements

- FR1: TermIt instance selection
 - The extension must allow the user to choose from a dropdown list what TermIt instance (server) will it be connected to.
- FR2: Authentication
 - The extension must allow the user to login and logout into their account. A link to a TermIt sign up page must be provided for users who don't have an account yet.
- FR3: Anonymous annotations
 - The extension must support annotating web pages even if not the user is not logged in. Annotations created this way will not persisted

to the server and lost upon leaving the given web page. The user will be prompted to register to save progress.

- FR4: Vocabulary selection
 - The extension must allow the user to choose a vocabulary from a dropdown list, based on which suggested annotations of web pages will be generated.
 - (desirable): By default, annotations should be based on all vocabularies available in the selected TermIt instance.
- FR5: Annotation of current web page
 - The extension must allow the user to generate annotation suggestions for the current web page and be able to create annotations manually. Annotations will be highlighted in page's text by different colors and borders.
 - The extension must support all types of annotations of TermIt (term occurrences):
 - Occurrence of an unknown term
 - Occurrence of an existing term
 - Term definition
 - Definition of an unknown term
 - Proposed occurrence of a new term
 - Proposed occurrence of an existing term
- FR6: Term creation from a proposed new term
 - The extension must allow the user to create a new term from a proposed new term annotation.
- FR7: Term creation from unannotated text
 - The extension must allow the user to create a new term from arbitrarily unannotated text on the web page.
- FR8: Existing term occurrence creation from unannotated text
 - The extension must allow the user to create a new occurrence of an existing term from arbitrarily unannotated text on the web page.
- FR9: Existing term occurrence confirmation
 - The extension must allow the user to confirm the occurrence of an existing term from a proposed occurrence of existing term.
- FR10: Annotation deletion
 - The extension must allow the user to delete an annotation of any of the supported annotation types.

- FR11: Deletion of all annotations on page
 - The extension must allow the user to delete all annotations at once from the current page.
- FR12: Deletion of all suggested annotations on page
 - The extension must allow the user to delete all suggested annotations at once from the current page.
- FR13: Showing previously created annotations
 - The extension must automatically show previously created annotations when the user revisits any of previously annotated pages.
- FR14: Annotator Sidebar
 - Upon clicking the extension's browser action icon, the extension must display a sidebar panel, showing the list of annotations on the the current web page and allowing the user to take the following actions:
 - Annotate the current web page
 - Delete annotations from the current web page
 - Follow a link to TermIt to show an assigned Term or selected vocabulary
 - Login or logout depending on the current status
- FR15: Disable and enable annotations
 - The extension must allow the user to disable or enable annotations in their browser.
- FR16: Show not found annotations
 - When revisiting a page with previously created annotations, the extension must show the user any existing terms occurrences or term definitions that were not found on the page.

■ 4.1.2 Non-functional requirements

- NFR1: Browser compatibility
 - The extension must be compatible with Google Chrome.
 - (desirable) The extension should be compatible with Mozilla Firefox and Microsoft Edge.
- NFR2: Extension distribution (desirable)
 - The extension should be able to be distributed through an official browser extension store (Chrome Web Store, Firefox Browser Add-ons, or Microsoft Edge Add-ons).

- NFR3: User friendliness and user interface consistency
 - The extension's user interface must be user-friendly and functionally similar to TermIt's web application user interface. There can be visual improvements to the UI, but the fundamental functionality should remain the same.
- NFR4: Noninvasiveness
 - The extension must not disturb the user while browsing the Web, and annotations must appear only after explicit activation for a given page or when revisiting a previously annotated page.
- NFR5: Static web pages support
 - The extension's annotations must work reliably on static HTML pages, not necessarily on websites with dynamic content or single-page applications.
 - Other documents like PDFs opened in the browser will not be supported.
- NFR6: Robust annotation resolution
 - The extension must save created annotations reliably so that they will be shown on the page again on subsequent visits, even after minor modifications of the page by its author.
- NFR7: Data persistence on server
 - The extension must save terms, annotations, and annotated website information to the selected TermIt server, not in memory or local storage in the browser. Annotations do not need to be stored on the server if the user is not logged in.

■ 4.2 TermIt-wide architectural changes

Firstly, this section describes the redesign of annotations that will need to happen to support the extension's annotations within the TermIt system. After that, it will describe the designed solution for annotations resolution in detail. As a result of our redesign, there will be changes required across different TermIt components, which we will discuss followingly. The updated high-level architecture, as explained in this section, is depicted in Figure 4.1

■ 4.2.1 Annotation resolution redign

As stated in Section 3.2.2, annotations within the TermIt system only occur by modifying a specific HTML element within the document returned to TermIt from the Annotation service.

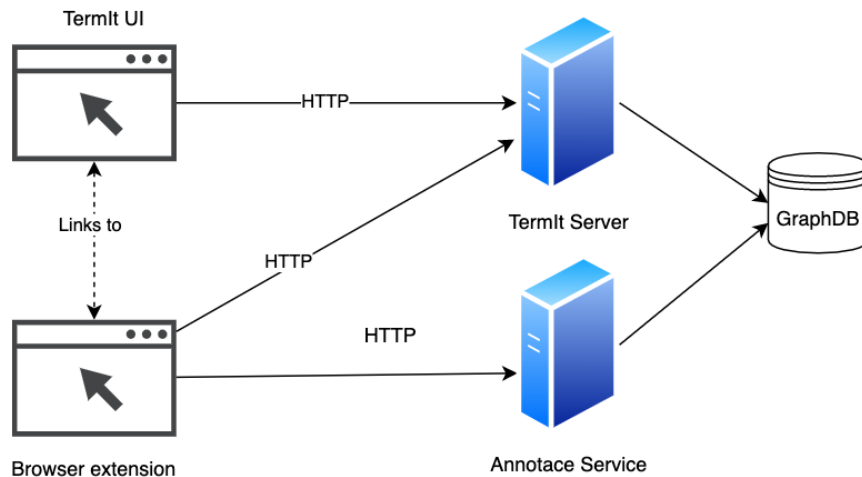


Figure 4.1: New TermIt architecture

This approach is not suitable for our browser extension use case, where annotation of any text within an open web page is assumed. We could simply take the returned annotated HTML and try to overwrite the current content of the page (using e.g., “document.body.innerHTML = <modified-html-from-annotation-services>”). However, some parts of the page would break in many cases, and event listeners stop working because they have not been registered for replaced HTML elements and other issues. In the case of a single-page application relying solely on rendering markup using JavaScript, it would most likely make the page completely unusable. In addition, even if the existing method did work, we would run into incompatibility problems with any website changes by its author. In its nature, this solution would not be much different from what TermIt already offers. Also, storing entire HTML documents and transferring them over the network consumes unnecessary storage and bandwidth resources.

Finding a more flexible solution for the extension requirements will therefore be necessary. Having experimented with a few different approaches, it quickly became apparent that generating and storing some form of CSS selectors (or possibly XPath selectors) is practically the only viable solution for this problem. CSS selectors can select the parent element where the annotation is found, and that can later be scanned for the occurrence of annotation text that we will also store.

To account for possible duplicates of the same term in one HTML element, its position (starting index) within the element needs to be stored. This way, annotations can be performed in the browser as the user views the page by replacing only annotated terms as they occur on the page.

■ Approach validation

The natural disadvantage of CSS selectors is that they can be susceptible to changes in the page. For example, if we have the selector

`div > .element-class > nth-child (1)` and the author of the website changes the class name of the given element from `element-class` to `element-class-2`, this selector will no longer work. Furthermore, it may happen that selectors are not unique and select more than one element. Thus, we must ensure to generate robust and unique selectors. Moreover, it will be needed to generate multiple different such selectors so that in the case of an occasional unavoidable failure of a selector, there would always be a spare selector to fall back on.

With these issues in mind, to test that our approach works correctly and reliably enough, we have created a simple proof of concept extension to validate our idea, which is described in detail in Appendix B. Its results were successful, and we could go forward with the design and implementation.

■ Selector generation and resolution algorithm

As described in the previous paragraphs, we will employ a CSS selector, a left offset, and the value of the annotations' text to generate an annotation selector. While a relatively simple algorithm, it is relied upon in the rest of our work and should be broken down in detail, so it is visualized in Figure 4.2 and described in this section.

1. We have identified a piece of text on the page to annotate, either by user selection in the browser or programmatically during text analysis.
2. Find the direct parent of the text identified. If the identified text spans multiple elements and thus does not have a direct parent that would contain it entirely, find a higher ancestor element that does contain the complete selection.
3. Once we have the parent element, generate its CSS selector(s) using a library.
4. Within the parent element, take all the text to the left text to be annotated and measure its length after removing any whitespace (to account for issues as per Section 5.7).
5. Take the string value of the text to be annotated itself and store it along with the measured offset and the parent CSS selector(s) as per the previous two steps.

To resolve the selector at any time, we simply need to select the parent with the CSS selector, check the text at its stored index, and compare the stored text content with the text on the page. If everything matches, we have found our annotation and can wrap it in the page's DOM. Note that the implementation details of the selector resolution process are further discussed in Section 5.8.

Step 1 - we have selected a text to annotate

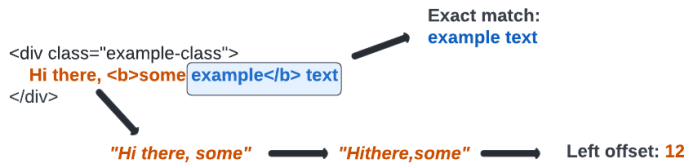
```
<body>
  <div id="example-id">
    <div class="example-class">
      Hi there, <b>some example</b> text
    </div>
  </div>
</body>
```

Steps 2-3 - find the closest parent and generate its CSS selector

CSS Selector: #example-id > example-class

```
<body>
  <div id="example-id">
    <div class="example-class">
      Hi there, <b>some example</b> text
    </div>
  </div>
</body>
```

Steps 3-5 - calculate offset and extract exact match



Resulting annotation selector:

CSS Selector: #example-id > example-class **Exact match:** example text **Left offset:** 12

Figure 4.2: Annotation selector generation algorithm

4.2.2 TermIt Server changes

While TermIt’s core components of its server and UI will largely stay intact, some changes will be needed to support the annotation redesign. Furthermore, at least for now, any changes must be backward compatible with the existing annotation capabilities of TermIt (i.e., the old way of annotating uploaded documents still needs to work).

The required changes will predominantly need take place on the level of TermIt’s data model, described in detail in the following paragraphs. As a result of the adjustment, some concerned endpoints will need to be slightly modified, and a few new ones will be added. Please refer to Figure 4.3 to see the updated data model, distinguishing between existing entities, newly added ones, and ones that were previously present but either only partly or not at all used, prior to our integration effort. Note that the diagram only contains a relevant subset of the entire data model, with lesser attributes and entities omitted.

■ Data model adjustments

RDF triples in TermIt's GraphDB database are mapped to and from Java objects automatically using the library JOPA [25], providing Object-ontological mapping (OOM) with support of OWL ontologies. The described changes will be designed and implemented on the level of JOPA entities – Java classes with JOPA annotations, requiring the definition of an IRI for each ontological class and relationship and allowing for setting ontological integrity constraints.

Firstly, a new resource data type will have to be introduced, not holding the HTML file itself but only a URL to the web page. For this purpose, a Website entity will be created as a counterpart of previously used File entity for representing uploaded documents for annotation.

■ 4.2.3 Annotations as Term occurrences

Apart from creating a Website entity, annotations and their generated selectors will need to be stored as instances of the TermOccurrence class. Optionally, they will have hold a link to a term if the assigned to one. Also, occurrences will now be linked to websites where they were created, allowing for subsequent retrieval of all TermOccurrences for a single page. Note that the notion of each annotation being represented by TermOccurrence had partially been implemented already. However, due to annotations being primarily stored in the document itself, occurrences were not consistently updated or deleted after initial automatic text analysis. Our redesign will ensure that TermOccurrences are always up to date and thus unlock new future possibilities for querying our vocabularies. For example, all occurrences of a single selected term could be shown in TermIt UI in the term's detail, with a link to their original location on the Web. For Web annotations, the TermWebsiteOccurrence subclass of TermOccurrence will be created.

To be able to store, retrieve, and delete web term occurrences, a new set of endpoints will be introduced to the TermIt Server's REST API.

■ 4.2.4 TermIt UI changes

The changes from TermIt Server also need to be reflected in the UI, where files will be stored in the list of vocabularies' resources and websites that will link to their URLs. Crucially, TermIt UI needs to be able to send messages to the extension through message passing (Section 3.3.2) to allow syncing logged in and configuration state. Message passing will be used for authentication as a JWT token¹ received from the server upon logging in will be sent to the extension to use for its calls to TermIt Server.

Moreover, TermIt UI also needs to interpret a definition of a term correctly. If it happens to have a Website as its origin, the user will be redirected to it when wanting to display the source of a definition.

¹<https://jwt.io/>

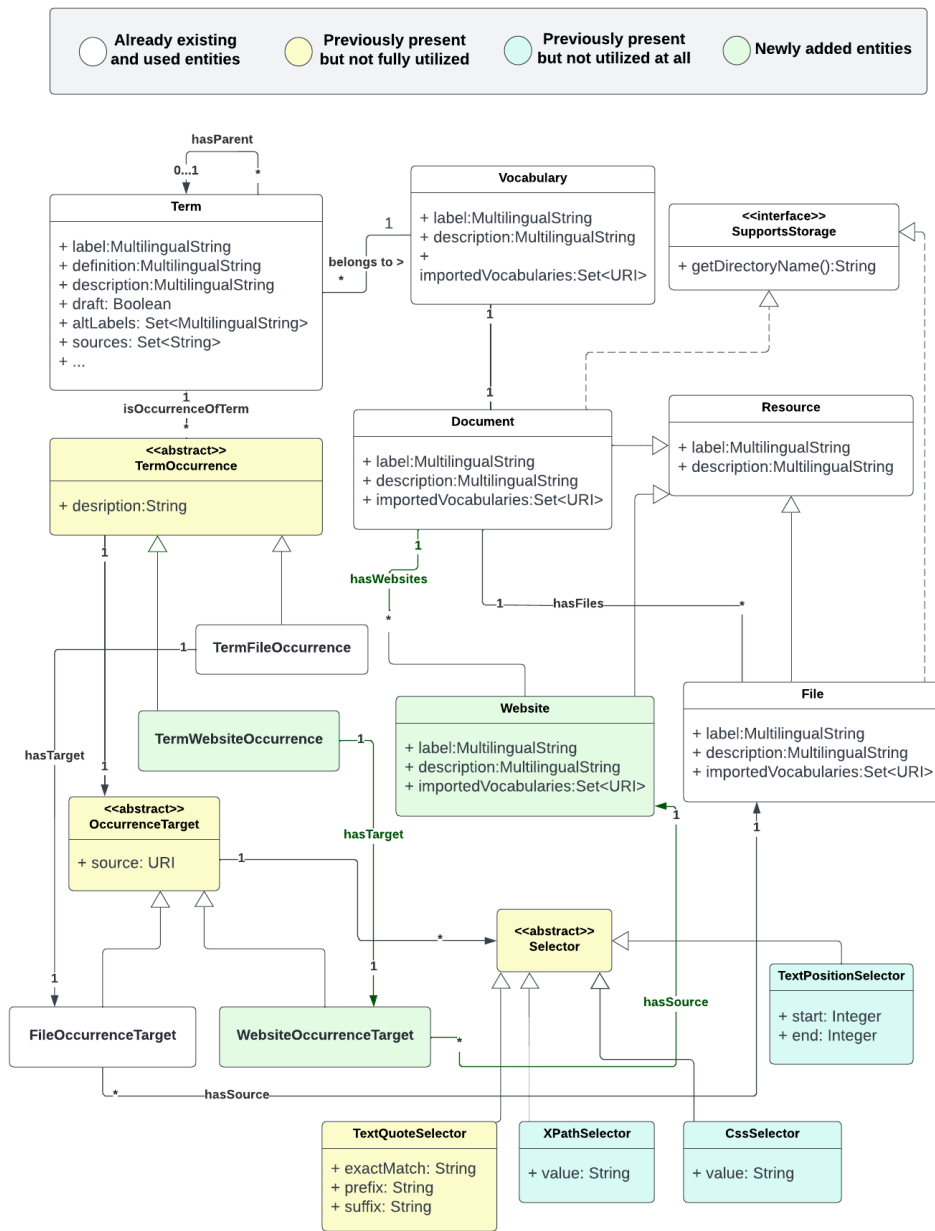


Figure 4.3: TermIt's Data Model after our modification

4.2.5 Annotace changes

A new endpoint will need to be added to Annotace, with identical input as the original one – an HTML document to annotate. However, instead of returning the modified HTML, found annotation suggestions will be mapped to term occurrences objects with generated selectors and returned in JSON format.

4.3 Extension architecture

Now that we have described changes needed on the level of the whole TermIt system, we can introduce the design of the browser extension itself, breaking down its different parts and their responsibilities. The entire architecture of the extension, along with its communication with TermIt UI, is visualized in Figure 4.4.

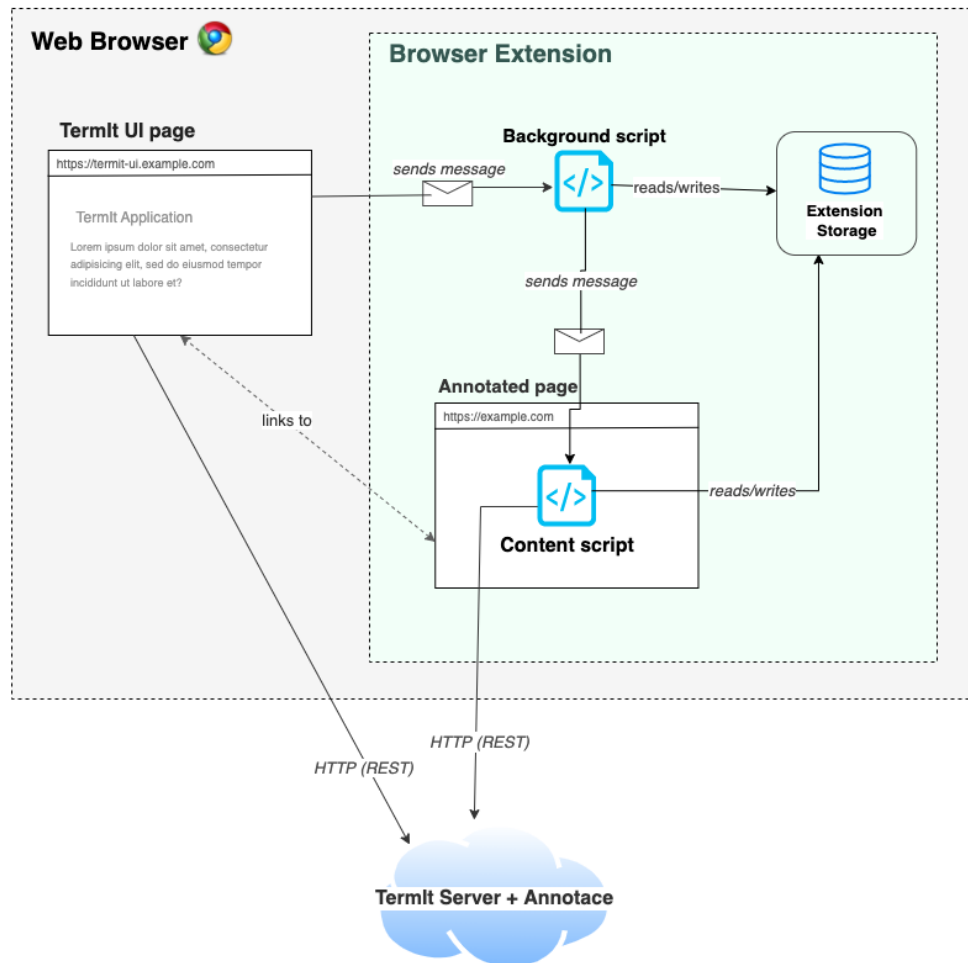


Figure 4.4: Browser extension architecture

4.3.1 Key factors

Before we present the design of our browser extension, it is essential to mention the critical defining factors influencing our architectural decisions.

- **Data persistence** – The need to persist all relevant data on the server (except for certain temporary and cached data), resulting in communication with the server to handle most user actions.

- **No data overlap across pages** – For each page, the user is required to choose a TermIt vocabulary, each having a different set of saved terms, and thus can effectively annotate pages within different contexts. The annotations themselves are also strictly tied to the given page. There could thus be little overlap between the data requested by different pages the extension is used on.
- **Interaction with the host page** – to achieve the core functionality of page annotations, the bulk of the user interaction is, of course, to happen right on the annotated page itself (e.g., as opposed to a browser action popup, a separate page dedicated to the extension).
- **Non-persistent background scripts** – It used to be quite a typical pattern to have data and some of the logic kept in a persistent background script of one's extension. However, with the advent of ephemeral service workers, this approach is no longer possible in browsers that only support the Manifest v3 extension specification.

■ 4.3.2 Content Script

Having the factors mentioned above in mind, the right design decision was, in our opinion, to structure the extension in the following way.

Injected into each new page, the content script essentially acts as its own independent application providing complete page annotation functionality by itself. It is similar to that of TermIt UI's Annotator page component but runs on any host page open in the browser instead of within TermIt UI's React application.

The content script manages its own state, calls the TermIt server and Annotace's REST APIs directly (similar to TermIt UI), and in most cases, has its independent lifecycle entirely. It thus has little dependency over any other parts of the extension, such as the background service worker or content scripts running in other opened tabs. Therefore, its internal subcomponents can be highly cohesive, revolving around the annotation functionality while only loosely coupled to other extension parts. The internals of the Annotator is further discussed below in Section 4.4.

■ 4.3.3 Browser storage

Naturally, some state and data still need to be shared outside the Annotator component, notably, the user's logged-in status, basic profile information, and preferences. This information is stored on an extension-wide level utilizing the extension's storage API² shared among all scripts and loaded in the content script upon opening a new page. Its critical attribute is that anything stored in the storage remains persistent, even if the browser is shut down, presenting a reliable source of storing necessary data on the client-side.

²<https://developer.chrome.com/docs/extensions/reference/storage/>

4.3.4 Background script and syncing with TermIt UI

The background script listens to the extension's install event and browser action icon clicks. Also, the browser's external message passing API and external messages from the TermIt UI are being handled to sync logged in and configuration state across the extension and the UI. To handle events, it updates the extension's shared storage or, occasionally, sends messages to content scripts that expect it. Typically it only sends messages to content scripts when synchronization needs to happen in real-time, such as when the user is logging in.

4.4 Extension's annotator

Having established the TermIt-wide architectural changes and the design of the extension as a whole, we will now have a detailed look at its Annotator component that runs in the content script. As discussed in Section 4.3.2, it almost acts as its own independent application.

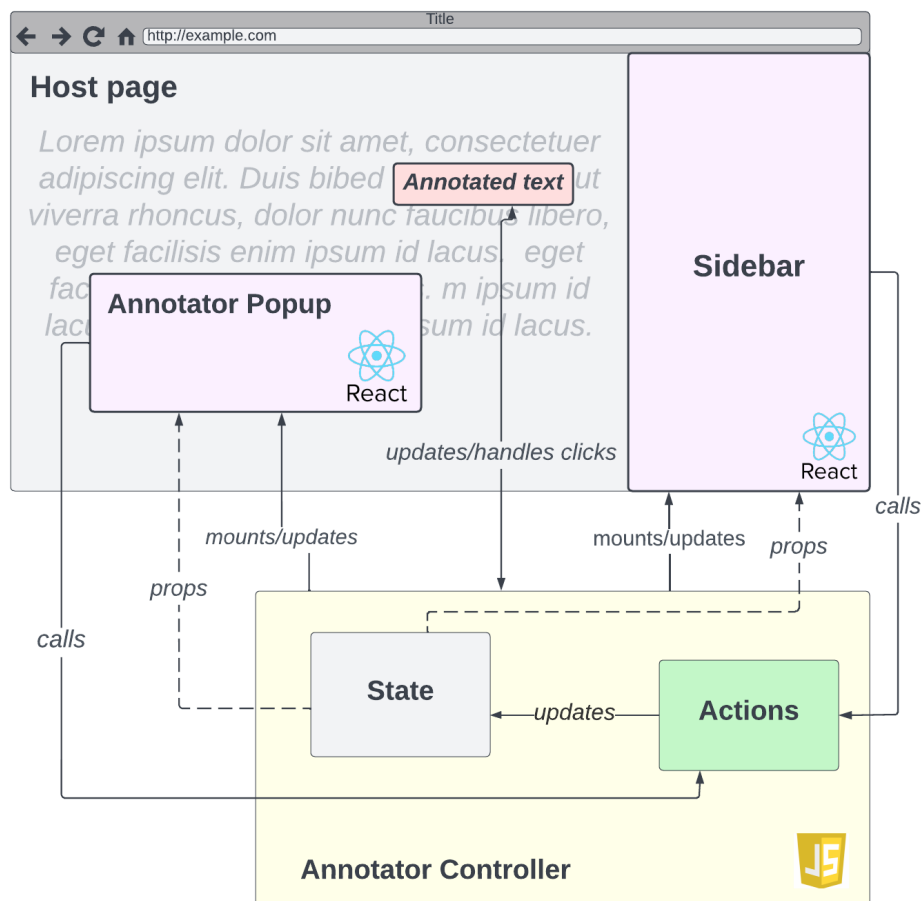


Figure 4.5: Extension's annotator architecture

■ 4.4.1 Basic structure

Firstly, there is the host page itself. Even though the extension will likely be used for primarily static websites containing normative documents, we should still focus on designing a generic solution instead of one that would be tailored to a minimal subset of specific pages.

The annotator itself can then further be broken down into the following subcomponents. First, in terms of the user interface, an annotation popup window will allow the user to interact with existing annotations or create new ones, similar to the existing solution in TermIt UI. Also, there needs to be a sidebar panel to manage annotations that can be opened or closed by the user.

To achieve this within the host page, we can render two distinct React components by mounting their React trees into elements we first insert into the DOM that serve as container elements. To help us achieve encapsulation and avoid CSS style conflicts from the host pages' style, Shadow DOM will be utilized.

■ 4.4.2 Control flow and state management

When a new page is opened, an `AnnotatorController` object is created and first waits for the page's, `load` event and then runs its initiation method - getting the current users' information and all annotated website information from browser storage, if any. It also initiates the sidebar and annotation popup React components.

Suppose the current page already has been annotated before, or the user later manually starts annotation through the Sidebar. In that case, annotations are requested from the server, and they are subsequently highlighted and shown on the page if found.

The `AnnotatorController` class holds a state object containing global annotator state - vocabularies, the user object, their configuration, and all page annotations and other fields, which are passed down to the React components as props. In addition, a set of functions called "actions" is exposed to be called from React components and elsewhere to modify the state. Only actions can modify the global state, call back-end APIs (`TermIt` and `Annotace`), abstract away state updates, and ensure a more predictable behavior than if all parts of code could modify that state directly.

Content actions also call React components container classes, such as for showing or hiding the annotation popup or sidebar. Finally, it updates the annotations displayed within the host page by calling helper classes or methods.

■ 4.4.3 The case for Redux

Redux [34] is JavaScript library for state management. In a nutshell, it can be described as follows. It has three core concepts which are actions, reducers, and stores. Actions are plain JavaScript objects that are sent to reducers,

which then, in turn, modify the store based on the action's type and payload. Redux thus offers a predictable and central means for managing a state that is not framework-dependent per se but is most commonly used with React.

TermIt UI already uses Redux for its application-wide state management needs. We have designed quite similar functionality ourselves as per the last section, compared to what Redux provides out-of-the-box. So naturally, one could ask why didn't we also use Redux for our browser extension use case and instead implemented our own above described solution?

It could be a viable option. A single store would be created in each active content script and then passed down to both distinct React trees we mount (Sidebar and AnnotationPopup). It would also allow for a certain amount of Redux-related code reuse from TermIt UI. However, we wanted our actions, among other things, to also trigger updates to our page annotations accordingly (outside of either of our React trees) and be responsible for the entire control flow of the content script. In addition, we were concerned this would not strictly fit into the Redux paradigm of managing state and introduce unnecessary complexity. Finally, we currently have just a few state fields to keep track of, so we felt confident implementing a simple custom solution.

Nevertheless, it was not, in any case, a clear-cut decision. Should our state grow a lot more complex over time and such a need arise, we are open to adding Redux. We have tried to write our current code to be as agnostic to the state implementation as possible, making it easier to replace it later.

■ 4.5 Annotation resolution

Now that we have described the architecture on all three different levels, we will discuss the annotation workflow and the relevant annotations resolution design decision.

■ 4.5.1 Initial annotation flow

To generate CSS selectors, we first use a rather basic yet quite reliable in practice method from the `ssoup` library in the Annotace service, where annotation suggestions and their selectors are initially generated.

Those suggested annotations then come to the extension's annotator, grouped by their parent selector. We then take each selector and try running a simple `document.querySelectorAll` on it, ensuring that exactly one element is returned. Conversely, we drop the selector and all associated annotations, no longer considering as there is little we can do to recover them.

Afterward, each found parent element is scanned for annotation matches starting at the specified index, which are then wrapped in a custom `<h-termit/>` HTML tags. Extra CSS classes are also added to the element, depending on the annotation type as listed in Section 4.1.1. At this stage, similarly, any annotations that failed to be found are filtered out and not

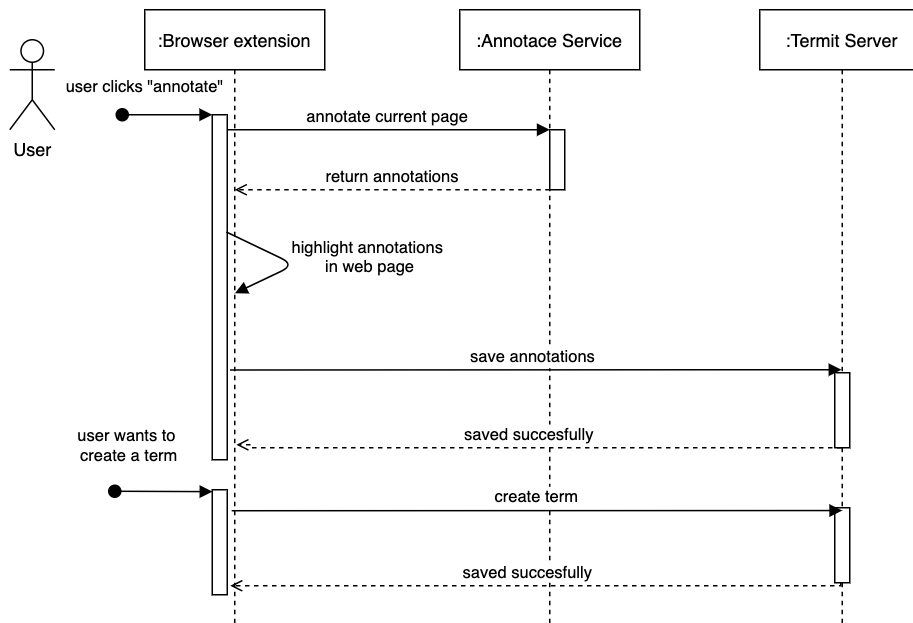


Figure 4.6: Annotation flow from browser extension

considered anymore. Finally, all successful annotations are persisted to TermIt Server.

4.5.2 Dropped annotation suggestions

The previous section mentions two separate instances of the selector failure and dropped annotations. Therefore, one could see this as a design flaw and a failure on our part. To refute this, let us elaborate a bit further.

Firstly, the dropped annotations are only suggestions, meaning that we are not losing any data explicitly created by the user. Secondly, we do not filter any parts of the page’s HTML that are being sent to Annotace in the first place (literally, the whole “document.body.outerHTML” is being sent). As a result, those failed selectors are often for annotations in ephemeral page elements (e.g., ads, cookie compliance messages). They have likely disappeared from the page or significantly changed by the time we get a response from Annotace and have little meaning for the end-user from an annotation perspective either way. If anything, such annotations could be distracting the user away from other, actually meaningful, annotations. While there will inevitably occasionally be legitimate suggested annotations dropped as a result of this filtering, it is clear that its benefits far outweigh its cost.

4.5.3 Ensuring selector robustness

It is only after we have a list of “clean” found annotations that are displayed to the user that we need to start to tread carefully and make sure to be able to select it on every subsequent visit of the page in the future. Therefore, a few (2-3) extra CSS selectors are generated through different JavaScript

libraries that try to generate the most robust and unique selectors possible. These additional selectors then serve as a fallback should the original one fail in the future as the page evolves (or one of them is the primary selector if they are manually created annotations that have not gone through Annotace).

Note that generating a variety of robust CSS selectors is only easily feasible on the front-end (as opposed to right in Annotace), as there exists a large number of selector generating libraries ³, unlike, for example, in Java, where we have not found a single one. When the user manually creates a new annotation, a set of robust selectors is generated right away on the front-end and persisted to TermIt Server.

³<https://github.com/fczbkk/css-selector-generator-benchmark>

Chapter 5

Implementation

The following chapter clarifies why we chose some of the critical libraries we did during implementation and contains other unexpected and worth mentioning implementation-related notes.

5.1 React justification

There are other prominent front-end frameworks, but for us, picking React as the framework of choice was an easy decision:

- TermIt UI already uses React. Therefore, some components or parts can be reused in the extension, which aligns well with the goal of providing a consistent user interface across the web application and the extension.
- React provides an excellent developer experience and is by far the most popular front-end framework of this type [35].
- It is what we are the most familiar with.

5.2 TermIt UI code reuse

The functionality of the newly implemented annotator and the existing one is quite similar, both written in TypeScript and React as their framework, and both calling TermIt Server's REST API. As a result, we could reuse reasonably large parts of the TermIt UI codebase in our extension. Reusing parts of the code allowed us to focus more narrowly on the newly designed annotation resolving functionality and other features. It also helped keep the two annotators consistent both on a user experience level and the data level of payloads sent and received by the TermIt server.

Specifically, we were able to reuse the following categories of code:

- Classes representing models of objects returned from TermIt Server, such as Term, TermOccurrence, Vocabulary, and Document
- Code related to calling TermIt Server, notably the RequestBuilder class.

- Relevant React components, predominantly those related used in Annotation Popup and modal window for creating new terms. While the core of those components usually stayed the same, some modifications were required more often than not (e.g., get state from props instead of Redux).
- Utility functions and constants

All the reused files are contained in the “/src/termit-ui-common” directory of the extension’s repository, and its internal file structure is the same as in TermIt UI. So, for example, what exists in TermIt UI in “src/model/Vocabulary.ts” is to be found in “src/termit-ui-common/model/Vocabulary.ts” in termit-extension. This approach allows for easier monitoring of changes across the two codebases. Furthermore, the assumption is that identical files in both projects will eventually be extracted to a separate NPM package that can be imported back into both repositories.

5.3 Sidebar performance optimizations

The extension’s sidebar is the primary way for the user to interact with the extension, apart from creating/editing/deleting individual annotations right where they occur on the page through the Annotation popup. Notably, the sidebar contains a list of all annotations that occur on the page. If annotating a larger page, hundreds or even thousands of annotation suggestions can be returned from Annotace alone, with many more potential annotations the user can eventually create manually. After initially implementing the extension and testing it on larger pages, very noticeable slowness started to bog down the user experience, with the page sometimes being stuck for seconds at a time when a new annotation was added or removed, thus forcing the sidebar to rerender.

To combat this, we have wrapped the original annotations list component with a virtualized list component from the `react-virtualized` library. It keeps the state of all the elements in memory and only renders to the DOM what is currently visible, depending on how far the user has scrolled. Fortunately, adding a virtualized list has instantly solved the problem, and we have not encountered any performance issues since, even when testing on the largest of pages.

5.4 Vocabularies caching

Previously annotated pages are automatically annotated again on subsequent page visits. A list of all annotated websites’ URLs needs to be accessed to scan and find a possible match for this to happen. Despite having the extension installed and active, the user may likely be doing a completely unrelated activity the vast majority time while browsing the Web. We thus want to avoid having to send a request to TermIt Server every time a new

page is opened in the browser to limit the load on the server. Therefore, a cached list of all user vocabularies is kept in browser storage, containing annotated websites for each vocabulary.

This cached list can grow quite large when serialized to browser storage. However, it may contain nested references to other objects, ballooning its size and sometimes resulting in a storage quota exceeded error thrown by the browser. We addressed this by mapping Vocabulary objects upon creation to plain objects containing only literal properties that used the extension's codebase without any references.

5.5 Website URL matching

As discussed in the previous section, a check needs to run to see if that page has previously been annotated or not and highlight annotations on the page accordingly. It is done by comparing the URL of the current page against all the URLs of all previously annotated websites (cached in vocabularies, as per last section). The issue is that a change in the URL can be just a change in a query parameter or a hash fragment, neither of which typically designate a separate page but rather provide some auxiliary information to the page, such as different filters and other preferences and others. Therefore, we have decided to ignore a hash fragment and query parameters and map all URLs to a value not containing them for our comparison.

Unfortunately, some websites will use a query parameter to decide what page will be shown, e.g., “<https://example.com?article=1>” will have shown a different article than “<https://example.com?article=2>”, but mapped to the same website in TermIt Annotate. Nevertheless, taking this approach works more reliably than not having such a mechanism in place.

5.6 Hypothes.is inspiration

Hypothes.is is probably the most similar browser extension to the one we wanted to build. So naturally, as it is open-source and its license¹ allows for it, we have examined its source code² before starting during our implementation. Simply creating a fork of its codebase and adjusting it to fit our use case was never seriously considered and quickly out ruled since there was not enough overlap between the functionality it offers and what we needed. It would make us maintain a huge codebase full of complexity unnecessary for our needs.

However, there were some low-level design patterns (not anything driving our major architectural decisions in Chapter 4) that were a fit for our implementation — for instance, taking Hypothes.is as the blueprint, we adopted wrapping React trees such as the sidebar and annotation popup in helper container classes. Also, a few pieces of mainly utility functions and classes

¹<https://github.com/hypothesis/client/blob/master/LICENSE>

²<https://github.com/hypothesis/client>

could be reused, such as the `SelectionObserver` class that keeps track of users' clicks and selections on the page. Such reused, unmodified files are found in the directory `src/content/util/hypothesis` of our extension repository in the attachment of this work, as described in Appendix D.

5.7 Annotation selector caveats

When it came to implementing selectors, we encountered an issue in calculating the annotations' offset. There turned out to be a mismatch between the actual representation of DOM in the browser and its jsoup (a Java HTML library) counterpart in Annotace. Specifically, whitespace present in the DOM was sometimes missing in jsoup and vice-versa, resulting in the index calculation being sometimes slightly off and selectors failing. As a result, we now do not consider any whitespace for our offset calculation purposes, arguably resulting in a slightly more robust resolution for other purposes as well, as any whitespace changes on the page will be ignored.

5.8 Mark.js

As we have touched on in the preceding paragraphs, the annotated text is in its parent element and wrapped in an extra HTML tag to show annotations on the page. While we could fully implement its functionality ourselves, there are certain intricacies in replacing parts of DOM elements. Specifically, it can be easy to break the original page's layout and event listeners or negatively affect the page when modifying existing elements on an unknown page not carefully enough, especially if the annotation spans multiple elements. We have therefore looked for an existing library that provides this functionality. Our criteria were as follows. To be reliable in selecting our text occurrences within an element, cause minimal unintended consequences (e.g., breaking events listeners or styles) and ideally offer enough configuration options to make it be able to pair well with our selectors.

Despite initially looking at a few different libraries such as `hrjs`³ or `lumin`⁴, we quickly settled on `Mark.js`⁵, as it seemed to have been the only library even to come close to meeting our needs. All other alternatives either did not offer much in terms of configuration, there was hardly any documentation, or it was outright doubtful how well the library would work. Furthermore, unlike `Mark.js`, with 260k weekly downloads on npm, none of them have become at least moderately popular among developers. Finally, the fact that we already had some prior experience with `Mark.js` also helped sway our decision towards it.

`Mark.js` takes in a DOM element to work on and then an arbitrarily string to try to match within the text of that element. It exposes many configuration

³<https://mburakerman.github.io/hrjs/>

⁴<https://github.com/pshihn/lumin>

⁵<https://markjs.io/>

options, such as the type of HTML tag wrapped around matches, classes that will be added, and, more importantly, a filter callback function to be run on each potential match. This is key because, within this callback, we can then check the offset of the current potential match and compare it with the value stored in our annotation selector and thus filter out any duplicates at wrong indices.

Unfortunately, the parameters passed into the filter callback did not contain the relevant information from which we could calculate the current offset. Since the library is open-source and published under the MIT license ⁶, we forked its repository ⁷. We made the necessary changes to calculate the offset and pass it into the filter function as required. Forking the library would probably have to be done at some point in any case, even if it was not for this adjustment, as the open-source community is actively maintaining it.

We have expected the library to work reliably, to begin with, but after implementation, our expectations were exceeded. It works incredibly well for virtually any type of plausible text selections the user could make on the page, even if spanning across many different elements. Also, different annotations can even be stacked with ease, allowing the user not to be limited to a single annotation in each piece of text.

⁶<https://opensource.org/licenses/MIT>

⁷<https://github.com/alanbuzek/mark.js>

Chapter 6

Evaluation

One of the significant challenges when designing and implementing this work was the resolution of annotations, especially in Web documents that evolve over time. It is thus paramount to thoroughly examine annotations' robustness in this context, as will be done in the following section. Afterward, the results of a user study on our browser extension will be presented, also representing a significant indicator when evaluating our work.

6.1 Annotation resolution robustness

At the start, when analyzing the architecture of current annotations in TermIt and designing our solution, an unknown essential factor, and later a challenge during implementation, presented annotation resolution.

Before diving into evaluating annotation resolution in documents evolving in time, we will first examine the annotation resolution efficacy in Web pages in a single point of time, without the underlying HTML document evolving. The findings we establish in this part of testing will then serve as the basis and a reference point for evaluating evolving documents.

6.1.1 Unchanged Web pages

For this part of testing, we conducted subsequent page visits within a brief time span, such that there would be no explicit modification of the page by its publisher.

Firstly, even before implementing our solution, we came up with a simple proof of concept extension, using an adjusted version of Annotace, testing our proposed design of CSS selectors and offset to resolve annotations. The proof of concept is discussed in detail in Appendix B — its results were overwhelmingly positive, resulting in an annotation success rate of about 99.6%. In other words, if an annotation was generated on the server in Annotace, it had roughly a 99.6% chance of being successfully found and highlighted on the page in the users' browser. As discussed in Section 4.5.2, we decided to drop any remaining annotations (0.4%) as they were most likely not relevant to the user.

This initial success from our proof of concept carried over to implementing our full-fledged extension. Once the implementation was done, we tested the ability to resolve annotations on any subsequent page visits, during which the user may manually create or delete more annotations. While it was required for the extension to only work on static pages, for the sake of completeness, we also mention how it fares on (partly) dynamic pages.

■ Static pages

Static pages, such as legal and other normative documents accessible on the Web as HTML, represent the primary use case for our extension. Per our testing, on a static page, virtually all annotations would be found (ignoring any initially dropped annotations as stated above). Moreover, that was the case on subsequent page visits, even as the user manually created or deleted arbitrary annotations as they visited the page each time. Rare instances of resolution failure could be produced, but likely only as a result of a deliberate effort (e.g., by creating an extensive annotation spanning the whole page).

■ Partly dynamic pages

An example of a partly dynamic page could be an article on a news site or a blog. The bulk of the page's content, such as the text of the article itself, in this case, is static. However, it also contains smaller parts of dynamic content, such as a list of other recommended articles to read or user-generated comments that may be different each time. In this scenario, the extension also works as expected, resolving annotations in static parts of the page, with annotations from dynamic parts left unresolved if no longer present.

■ Entirely dynamic pages

Finally, there are also websites with entirely dynamic content, such as algorithmic social media feeds, where most of the content is dynamic. The contents of such pages are different on each page load, despite the URL unchanged and interpreted as the same page by our extension. There is little benefit to annotating such pages, and we were not considering them for our testing.

■ Shortcomings

While the results presented look overwhelmingly positive, and they certainly are, we have encountered a few minor issues that need to be mentioned and could not be easily assigned into one of the three categories of pages above. Notably, selectors may also fail if the page has changed in some, maybe not even a noticeable way. For example, if we ran the automatic annotations when a cookie panel was open, annotations would persist in that part of the page. Afterward, when revisiting the page and having all annotations resolved, the ones the panel would fail. Thus, even a page that we would consider static would have some annotations fail.

Apart from that, it also sometimes happens that results of automatic annotation annotate text that is entirely not visible on the page to the user. However, this may be easily fixable in the future, as it should suffice to check whether the parent element is visible on the page and within specific dimensions.

■ 6.2 Evolving Web pages

Knowing how well annotation resolution works on static pages in a single point, we can now accurately assess its capability to cope in situations where the document has changed in time. Note that for this assessment, we will only consider static pages. Specifically, select examples from Czech legislation will be used. Furthermore, to simulate Web pages evolving over time, we will compare different versions of the same legal document.

■ 6.2.1 Expectations

A Web page can, over time, evolve in an arbitrary number of ways, and, inevitably, there is a chance for an annotation that previously existed on the page not to be found anymore. Not found annotations are not necessarily a failure on our part but rather an unavoidable consequence of the evolving nature of the Web that we should plan for. To address this scenario, any not found annotations are shown in the sidebar to the user, allowing them to delete them and show what element of the document the annotation previously appeared in, if possible, such that they can create new, more appropriate annotations.

As follows, we can categorize the types of annotation failures into four main types that can occur in evolving documents.

■ Element not found

Firstly, certain HTML elements can disappear entirely as the page changes in time. If the removed element were a direct parent containing an annotation, it would, of course, not be found. In legal documents, this can typically occur when, for example, a section is removed in its new version. In such cases, with little to be recovered, the user will most likely delete the annotation.

■ Text not found

When the parent element of an annotation is found through its CSS selector, the annotation text still needs to be matched precisely within the text. As shown in Figure 6.1, for example, when a new version of law changes, a section may be kept, but its content change, which may result in the annotation's text no longer being there. In such cases, the user can click on the annotation in the extension's sidebar, and the parent element is then highlighted to them, allowing for possibly creating a new annotation in the modified part of the document.

1. annotations before change
j) údaj o spáchaném přestupku podle tohoto zákona a uloženém správním trestu, pokud neuplynulo v
2. change visualization
j) údaj o spáchaném přestupku podle tohoto zákona a uloženém správním trestu, nabytí právní moci rozhodnutí o přestupku, j) údaj o odmítnutí provedení tlumočnického úkonu podle § 19 odst. 1 písm. c),
3. annotations after change
j) údaj o odmítnutí provedení tlumočnického úkonu podle § 19 odst. 1 písm. c),

Figure 6.1: Example of an annotation text not found after the document has evolved

■ Offset mismatch

This failure can occur when the annotated text still appears in the same element, but its position has been shifted to the right or left. For example, when any text to the left of the annotation has its length changed, it results in the original annotated text appearing in a different position than the selector, an example of which is shown in Figure 6.2. Notice that in that example, there is only a subtle difference – the string “f)” of length two is replaced by a single letter “e”, causing a mismatch by one.

■ Unexpected failure

The three types mentioned above of failures are expected when the described changes occur, given how our deterministic annotation resolution algorithm works. However, there can also be an unexpected and undesired failure. The parent element may fail to be selected despite being on the page. For example, different CSS classes are assigned to it. This may, in turn, cause our CSS selector to fail. This is precisely why we generate a multitude of selectors, such that we can always have one to fall back on and hopefully succeed in selecting the desired element.

While unexpected failures cannot be avoided with absolute certainty, they should be minimized to the lowest possible level. Luckily, in our experience, legal and other normative documents on the Web seem to have a consistent structure across different versions, minimizing any potential CSS selector failure.

1. annotations before change
<p>d) nebyla v posledních 3 letech před podáním žádosti o zápis potrestána podle § 37 odst. 1 písm. a) až d), f) až l) nebo podle § 37 odst. 2 písm. za přestupek podle § 38 odst. 1 písm. a) až f) nebo podle § 38 odst. 2 přestupek podle § 39 odst. 1 písm. a), b) až d), ani pokutou ve výši 100 Kč až d), nebo které v posledních 5 letech před podáním žádosti o zápis činnost podle § 14 odst. 1 písm. d),</p>
2. change visualization
<p>d) nebyla v posledních 3 letech před podáním žádosti o zápis potrestána podle § 37 odst. 1 písm. a) až d), f) až l) nebo podle § 37 odst. 2 písm. KČ za přestupek podle § 38 odst. 1 písm. a) až f) nebo podle § 38 odst. 2 KČ za přestupek podle § 39 odst. 1 písm. a), b) až d), ani pokutou ve výši 100 a), b) až d), nebo které v posledních 5 letech před podáním žádosti o tlumočnickou činnost podle § 14 odst. 1 písm. d),</p>
<p>d) nebyla v posledních 3 letech před podáním žádosti o zápis potrestána podle § 37 odst. 1 písm. a) až d), f) až k) nebo podle § 37 odst. 2 písm. za přestupek podle § 38 odst. 1 písm. a) až e) nebo podle § 38 odst. 2 přestupek podle § 39 odst. 1 písm. a), b) až d), ani pokutou ve výši 100 až d), nebo které v posledních 5 letech před podáním žádosti o zápis činnost podle § 14 odst. 1 písm. c),</p>
3. annotations after change
<p>d) nebyla v posledních 3 letech před podáním žádosti o zápis potrestána podle § 37 odst. 1 písm. a) až d), f) až k) nebo podle § 37 odst. 2 písm. za přestupek podle § 38 odst. 1 písm. a) až e) nebo podle § 38 odst. 2 přestupek podle § 39 odst. 1 písm. a), b) až d), ani pokutou ve výši 100 Kč d), nebo které v posledních 5 letech před podáním žádosti o zápis nebyly činnost podle § 14 odst. 1 písm. c),</p>

Figure 6.2: Example of an offset mismatch selector failure.

6.2.2 Testing

The website zakonyprolidi.cz, providing an archive of Czech legal regulations, will be leveraged to source versions of legal documents for our assessment. It can visualize the differences between versions of a legal document, making it easy to conduct our testing. Additionally, it is generally a popular source of Czech legal documents, making it a likely website to be used by our extension's users in the future.

Test cases

We have selected three legal documents at our discretion, each to be compared across two different versions. The document versions were either immediately succeeding or across many versions. Either way, we made sure that there were a fair amount of changes such that there would be annotation resolution failures to investigate.

First, the respective earlier version of the document was annotated with

automatic suggestions and annotations manually created by the user. Afterward, we opened the respective later version of the document in a new tab, and the extension showed us a list of failed annotations to investigate. For this simulation of the website’s evolution over time, slight, temporary modifications were made to the extension’s code to interpret all URLs. The two pages would be interpreted as the same one which has evolved.

Once on the second respective version of the legal document, we examined each annotation failure, comparing it to the previous version and using the version of the comparison tool offered by zakonyprolidi.cz. The root cause of each failed annotation was determined and assigned to its respective failure category as defined in Section 6.2.1

Results

The results of our assessment were overwhelmingly positive and are broken down in detail in Table B.1. There were 3081 total annotations, out of which 144 failed to be found after visiting the second version of the respective legal document. Incredibly, all 144 were a function of the changes in the legal norm itself across its versions, falling under one of the three types of expected failures, which is precisely what was expected and desired.

Document name	Versions compared	Annotations total	Failures breakdown ¹	Difference visualization
Zákon č. 194/2017 Sb.	3.version ² 4.version ³	1009	29/10/6/0	at zakonyprolidi.cz ⁴
Zákon č. 354/2019 Sb.	0.version ⁵ 1.version ⁶	785	4/16/7/0	at zakonyprolidi.cz ⁷
Zákon č. 549/1991 Sb.	30.version 45.version ⁸	1287	14/43/15/0	not publicly available

Table 6.1: Test results on evolving Web pages

6.3 User study

Apart from thoroughly evaluating the implemented solution on a critical technical requirement that resolving annotations represents, it is also imperative to assess its usability. To do so, we have conducted a user study, which will be presented in the section.

¹Failure types as per s Section 6.2.1: Element not found / Text not found / Offset mismatch / Unexpected failure

²<https://www.zakonyprolidi.cz/print/cs/2017-194/zneni-20210101.htm>

³<https://www.zakonyprolidi.cz/print/cs/2017-194/zneni-20220101.htm>

⁴<https://www.zakonyprolidi.cz/cs/2017-194/zneni-20220101?porov=20210101text=>

⁵<https://www.zakonyprolidi.cz/print/cs/2019-354/zneni-0.htm>

⁶<https://www.zakonyprolidi.cz/print/cs/2019-354/zneni-20210101.htm>

⁷<https://www.zakonyprolidi.cz/cs/2019-354/zneni-20210101?porov=0>

⁸<https://www.zakonyprolidi.cz/print/cs/1991-549/zneni-20220101.htm>

■ 6.3.1 Testing environment

Before conducting the study, it was necessary to have the finished browser extension distributed to testers. Apart from that, as the extension is dependent on other components of TermIt as per the designed architecture in Section 4, it was necessary to deploy TermIt Server, Termit UI, and Annotace, all of which contain changes we have made in the respective repositories as part of this work. All three modified components have been deployed to our server, with a fresh instance of the GraphDB database.

■ 6.3.2 User group

Given that the target user group for the current TermIt system is primarily domain experts who use TermIt for designing vocabularies, we have decided to include testers from this target groups as our study participants. More specifically, three domain experts as well as two ontologist were selected for our study, all of whom had prior, some quite extensive, experience with using TermIt and its annotation capabilities. There are certainly trade-offs to consider only including testers with prior TermIt experience, such as not having a completely new users' perspective. Nevertheless, we concluded that such a group would best put things into perspective knowing TermIt's current annotation capabilities and provide valuable insight into how our extension fairs against its incumbent, which is ultimately the most important for evaluating our work and deciding TermIt Annotate's future direction.

■ 6.3.3 Test scenario and feedback

There was one common test scenario prepared and sent to each tester and additional instructions to go through on their own. The test scenario's steps were logically grouped into a separate section, sets of actions to fulfill. After completing the test scenario, testers submitted their feedback into a form. They were asked to rate each section of the test scenario on a scale from 1 (worst) to 5 (best) and provide additional feedback for each section and rate TermIt Annotate as a whole.

The test scenarios' description, along with its respective user feedback, is accessible in Appendix C. Screenshots of the extension's tutorial are also included in Appendix A, offering an easy way to familirize oneself with the extension.

■ Overall impressions

Apart from being asked to rate and comment on each section of the test scenario separately, testers were asked more general questions at the end and rated their overall impression.

- Would TermIt Annotate be more suitable for your work than the current Annotator in the TermIt system?

Chapter 7

Conclusions

The goal of this work was to create a browser extension for creating semantic vocabularies through annotations on pages of the Web and integrate it with the TermIt system. Moreover, it was necessary to test its annotation resolution capabilities and assess its reusability.

7.1 Evaluation

As described in this work, the main goals of our work have now been achieved. Moreover, all the specified requirements have been implemented and met, except for the “desirable” parts of the requirements in Sections 4.1.1 and 4.1.2, both of which can easily be added in the future.

We have designed and subsequently implemented a working browser extension. Apart from that, a significant part of our work presented getting deeply familiar, analyzing, and ultimately making non-trivial changes in the repositories of Annotace, TermIt UI, and TermIt Server and ensuring all four software components are adequately integrated for our use case. Additionally, our implementation also required making changes to the open-source library of Mark.js, enabling wrapping annotation text in pages.

The critical architectural and later implementation challenge we faced was handling annotation resolution when starting our work. Our proposed solution presented a complete paradigm shift from it was handled previously, where annotations would be saved along as part of its underlying HTML document. While daunting at first, we were able first to validate our selector design with a proof of concept extension and subsequently implemented a full-fledge solution that handles annotation resolution well, even in evolving Web documents.

An unknown factor when starting our work also presented how well users will respond to TermIt Annotate compared to TermIt’s previous annotation functionality. With the conclusion of the usability study, this question has also been answered – current TermIt’s users have positively welcomed the extension. Some users see it as a better alternative for their needs already, and for others, more minor adjustments will be needed to make it fully ready.

7.2 TermIt Annotate and its future

This section briefly summarizes the current state of the TermIt Annotate browser extension and its path forward.

7.2.1 Chrome Web Store

We have submitted the TermIt Annotate browser extension to the Chrome Web Store (CWS), a public marketplace for the Google Chrome browser extensions. Our submission has since been approved and published for the public to see. While there are still some minor shortcomings, as described in Section 6.3.4, the extension is functional and meets its defined requirements.

Before it is deemed ready for use on a production deployment of TermIt, we have published its current testing version in this way. As specified in Section 7.2.1.

For future versions, the CWS's item can easily be updated. Once the time comes for it to be used in a production TermIt environment (or any other new TermIt environment), all that suffices is to add the environments' parameters in the `"/src/annotator/component/shared/InstanceSelection.tsx"` file in the `termit-extension` repository and upload the updated code to CWS.

Regarding other browsers, there are slight incompatibilities with other major browsers, but those could be quickly addressed. Notably, some browsers, such as Mozilla Firefox, have not switched to the Manifest v3 specification yet [30], meaning that at the minimum, the `manifest.json` file will need to be adjusted.

Public testing account

As outlined above, we have published the extension to the public in the Chrome store, and the following information can be used for anybody to try it out as a logged-in user.

- Extension Chrome Store Link: <https://chrome.google.com/webstore/detail/termit-annotate-semantic/penpnbbgbibnedecnkbnemoilfdjlbh>
- Instance to select: "Testování"
- Username: "termitAnnotateUser"
- Password: "termitAnnotateDemo"

7.2.2 Future work

With any piece of software these days, there is always so much to be improved, added, optimized, or adjusted in an infinite number of ways. In addition, users' requirements and business needs change over time, and so do other related software systems and technical specifications, requiring a constant effort by the developers to keep the software running smoothly and staying relevant.

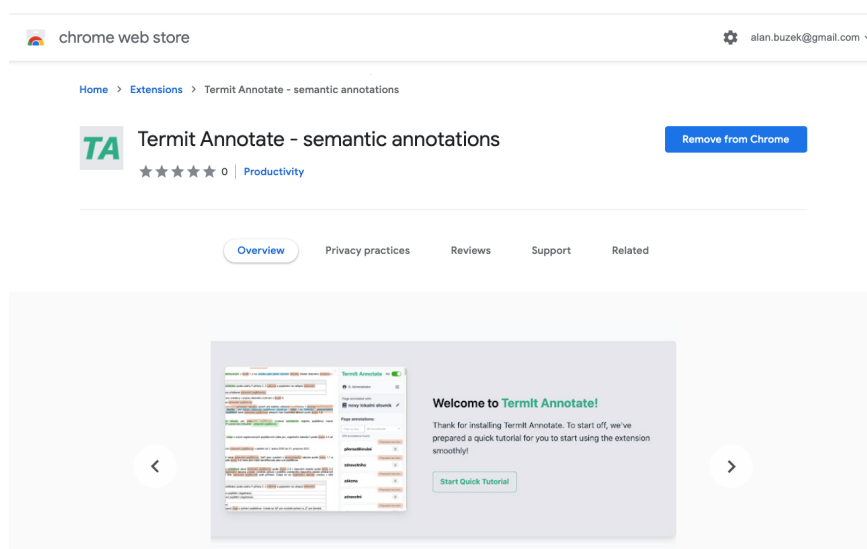


Figure 7.1: The Chrome Web Store listing of TermIt Annotate

This is especially true when developing a user-facing product running in a Web browser.

In this sense, of course, TermIt Annotate is no different. Despite our best efforts during the implementation, there is so much left to do for the product to be the best it can be, both from the end-user's perspective and from a technical standpoint. Nevertheless, we must not get overwhelmed by this impression, as chasing perfection is a futile effort, especially when it comes to software development.

That being said, we have identified the following areas of improvement to focus on next.

■ Flexible vocabulary and annotation selection

Currently, each annotated website is strictly tied to a single vocabulary. That may sometimes pose a problem for the user, who would like to annotate with multiple vocabularies simultaneously or iteratively, one at a time, as their needs change. At a minimum, the user should be allowed to annotate their website multiple times with the same vocabulary, which is currently not possible either.

■ UI/UX improvements

As apparent from users' feedback so far and our observations, the overall user experience is pleasant, but there are still quirks and minor issues here and there to address to make users' lives easier. This applies to the annotation popup and the sidebar's menus and buttons.

■ Sidebar functionality

While the current sidebar works well, there could be simple additions of functionality that would make it a lot more functional. Firstly, the list of occurrences should be sortable alphabetically, its position in the document, time of creation, or otherwise. When there are not found annotations, it should be possible to modify the annotation such that it now appears on the website that has evolved. Finally, bulk deleting of annotations should be possible for all suggested annotations and an arbitrary subset of annotations filtered out in the sidebar.

■ Bug fixing and test coverage

There are minor, miscellaneous bugs to be fixed that either emerged as reported during the user study or observed during our usage of the extension. For example, the user can delete annotations through the sidebar in anonymous mode. However, once logged in and asked to annotate the page again with a vocabulary, those same deleted annotation suggestions appear. Also, no unit or integration tests have been written to test the extension. This should also be an essential next task to focus on.



Bibliography

1. WOLFE, Joanna; NEUWIRTH, C M. From the margins to the center - The future of annotation. *Journal of Business and Technical Communication*. 2001, vol. 15, pp. 333–334.
2. IDE, Nancy. Introduction: The Handbook of Linguistic Annotation. In: 2017, p. 1. Available from DOI: 10.1007/978-94-024-0881-2_1.
3. LEDVINKA, Martin; KŘEMEN, Petr; SAEEDA, Lama; BLAŠKO, Miroslav. TermIt: A Practical Semantic Vocabulary Manager. 2020.
4. *Otevřená data II - Ministerstvo vnitra České republiky* [online] [visited on 2022-05-19]. Available from: <https://www.mvcr.cz/clanek/otevrena-data-ii.aspx>.
5. *Browser Extensions - Mozilla / MDN* [online]. [N.d.] [visited on 2021-12-21]. Available from: <https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions>.
6. *What are extensions? - Mozilla / MDN* [online] [visited on 2021-12-21]. Available from: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/What_are_WebExtensions.
7. *Goodbye, Desktop Apps. Modern web applications are replacing... / by Shalitha Suranga / Level Up Coding* [online] [visited on 2021-12-21]. Available from: <https://levelup.gitconnected.com/goodbye-desktop-apps-bf4d2c0438c4>.
8. *Create and publish custom Chrome apps extensions - Google Chrome Enterprise Help* [online] [visited on 2021-12-21]. Available from: <https://support.google.com/chrome/a/answer/2714278?hl=en>.
9. *PoolParty Semantic Suite - Your Complete Semantic Platform* [online] [visited on 2021-12-21]. Available from: <https://www.poolparty.biz/>.
10. *Chrome Web Store - Annotate the Web*. Available also from: https://chrome.google.com/webstore/category/collection/annotate_the_web.
11. *Weava Highlighter - Free Research Tool for PDFs Webpages* [online] [visited on 2021-12-21]. Available from: <https://www.weavatools.com/>.

12. *Diigo - Better reading and research with annotation, highlighter, sticky notes, archiving, bookmarking more.* [Online] [visited on 2021-12-21]. Available from: <https://www.diigo.com/>.
13. *Home : Hypothesis* [online] [visited on 2021-12-21]. Available from: <https://web.hypothes.is/>.
14. *Mosaic User's Guide: Annotations* [online]. [N.d.] [visited on 2021-12-21]. Available from: <https://www.desy.de/web/mosaic/help-on-annotate-win.html>.
15. *There's a Feature That Was Supposed Be in Web Browsers From the Very Beginning, but It Was Dropped at the Last Minute* [online] [visited on 2021-12-21]. Available from: <https://www.businessinsider.com/theres-a-feature-that-was-supposed-be-in-web-browsers-from-the-very-beginning-but-it-was-dropped-at-the-last-minute-2012-10>.
16. *Semantic Web - W3C* [online] [visited on 2021-12-21]. Available from: <https://www.w3.org/standards/semanticweb/>.
17. *SPARQL Query Language for RDF* [online] [visited on 2022-05-20]. Available from: <https://www.w3.org/TR/rdf-sparql-query/>.
18. SHADBOLT, Nigel; HALL, Wendy; BERNERS-LEE, Tim. The semantic web revisited. *IEEE Intelligent Systems*. 2006, vol. 21, no. 3, p. 96. ISSN 15411672. Available from DOI: 10.1109/MIS.2006.62.
19. *Ontologies - W3C* [online] [visited on 2021-12-21]. Available from: <https://www.w3.org/standards/semanticweb/ontology>.
20. *Data - W3C* [online] [visited on 2022-05-17]. Available from: <https://www.w3.org/standards/semanticweb/data>.
21. *RDF 1.1 Concepts and Abstract Syntax* [online] [visited on 2021-12-21]. Available from: <https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>.
22. *OWL 2 Web Ontology Language Document Overview (Second Edition)* [online] [visited on 2021-12-21]. Available from: <https://www.w3.org/TR/owl2-overview/>.
23. *SKOS Core Guide* [online] [visited on 2022-05-20]. Available from: <https://www.w3.org/TR/2005/WD-swbp-skos-core-guide-20051102/>.
24. *GraphDB Downloads and Resources* [online] [visited on 2022-01-10]. Available from: <https://graphdb.ontotext.com/>.
25. LEDVINKA, Martin; KREMEN, Petr. JOPA: Stay Object-Oriented When Persisting Ontologies. 2015, pp. 408–428. ISBN 978-3-319-29132-1. Available from DOI: 10.1007/978-3-319-29133-8_20.
26. *React - A JavaScript library for building user interfaces* [online] [visited on 2021-12-21]. Available from: <https://reactjs.org/>.

27. *Chrome Developers*. Architecture overview [online] [visited on 2021-12-20]. Available from: <https://developer.chrome.com/docs/extensions/mv3/architecture-overview/>.
28. *Overview of Manifest V3 - Chrome Developers* [online] [visited on 2021-12-21]. Available from: <https://developer.chrome.com/docs/extensions/mv3/intro/mv3-overview/>.
29. *Manifest V2 support timeline - Chrome Developers* [online] [visited on 2021-12-21]. Available from: <https://developer.chrome.com/docs/extensions/mv3/mv2-sunset/>.
30. *Manifest v3 in Firefox: Recap Next Steps | Mozilla Add-ons Community Blog* [online] [visited on 2022-05-20]. Available from: <https://blog.mozilla.org/addons/2022/05/18/manifest-v3-in-firefox-recap-next-steps/>.
31. *Content scripts - Chrome Developers* [online] [visited on 2022-05-17]. Available from: https://developer.chrome.com/docs/extensions/mv3/content_scripts/#isolated_world.
32. *Popups - Mozilla | MDN* [online] [visited on 2021-12-21]. Available from: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/user_interface/Popups.
33. *Message passing - Chrome Developers* [online] [visited on 2021-12-21]. Available from: <https://developer.chrome.com/docs/extensions/mv3/messaging/>.
34. *Redux - A predictable state container for JavaScript apps. | Redux* [online] [visited on 2022-05-20]. Available from: <https://redux.js.org/>.
35. *React vs Angular, which one is more popular among JavaScript developers* [online] [visited on 2021-12-21]. Available from: <https://www.peerbits.com/blog/react-vs-angular-which-one-popular-javascript-developers.html>.
36. *Zákony pro lidi - Sbírka zákonů ČR v aktuálním konsolidovaném znění* [online] [visited on 2022-01-12]. Available from: <https://www.zakonyprolidi.cz/>.



Appendix A

Tutorial

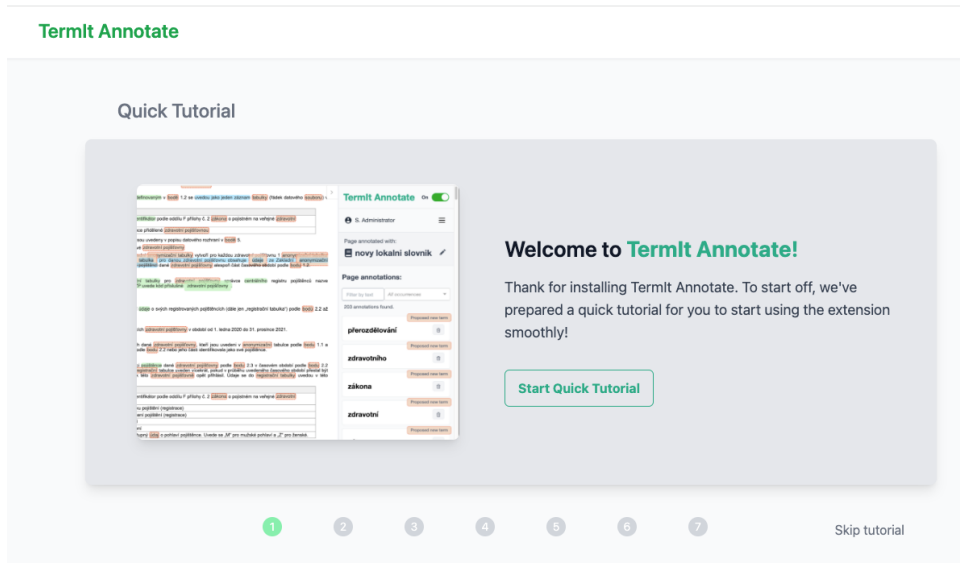


Figure A.1: 1. step of the tutorial – A brief introduction to Termit Annotate

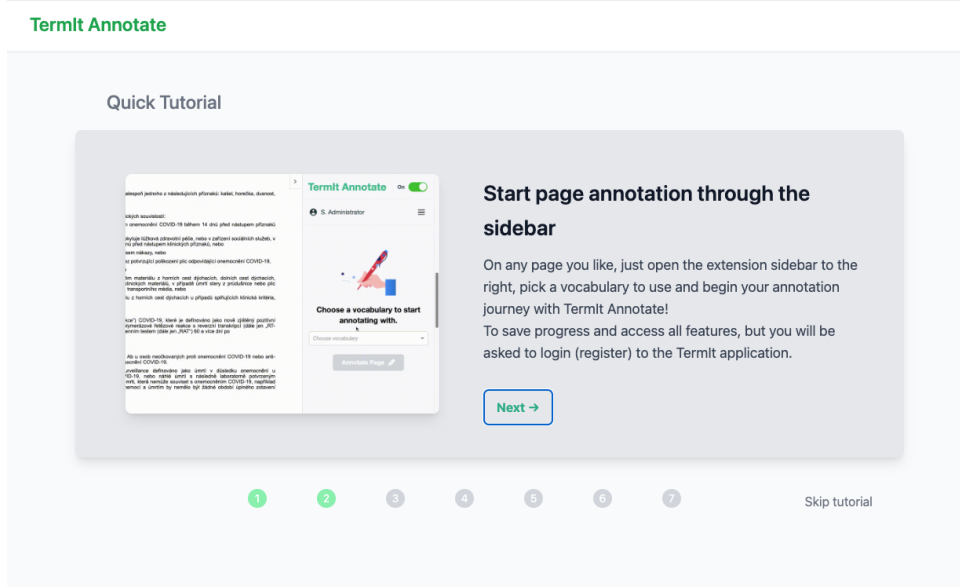


Figure A.2: 2. step of the tutorial – Start page annotation through the sidebar

Termit Annotate

Quick Tutorial

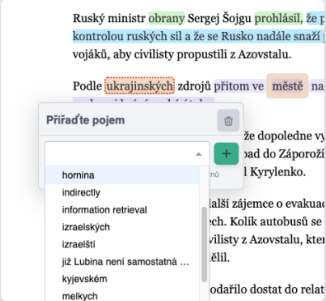
Confirm suggested annotations and create your own

After first triggering page annotation, you'll have suggested term occurrences appear on the page, based on the vocabulary you've chosen.

Confirm, remove or reassign annotation suggestions to terms as well as highlight any text to create your own annotations.

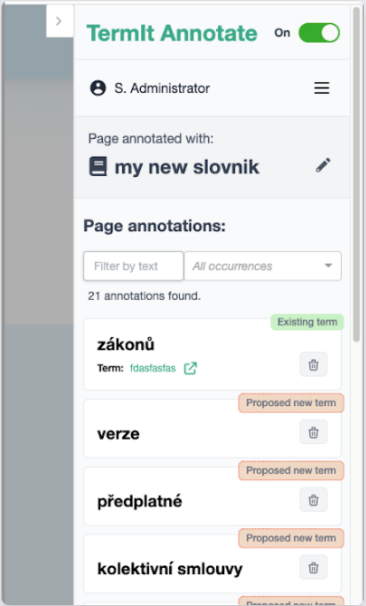
You can also create new terms in the selected vocabulary by clicking the '+' sign in the show popup.

[Next →](#)



1 2 3 4 5 6 7 Skip tutorial

Figure A.3: 3. step of the tutorial – Confirm suggested annotations and create your own



Use sidebar to keep things in grip

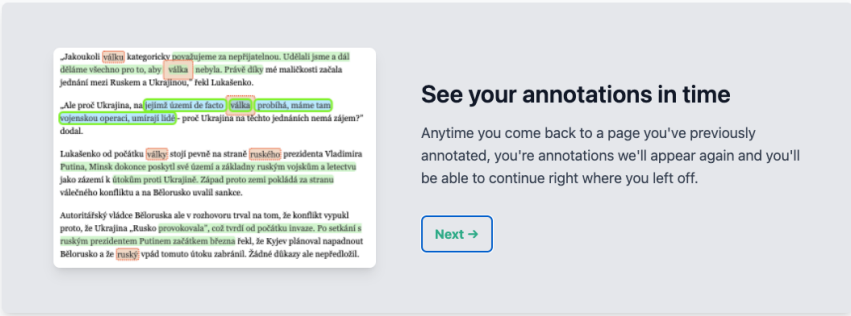
Through the extension's sidebar, you'll be able to search and manage existing page annotations, delete annotations or access a list of all annotated term pages.

[Next →](#)

Figure A.4: 4. step of the tutorial – Use sidebar to keep things in grip

TermIt Annotate

Quick Tutorial



„Jakoukoli válku kategoricky považujeme za nepřijatelnou. Udělali jsme a dál děláme všechno pro to, aby válka nebyla. Právě díky mě malíčosti začala jednání mezi Ruskem a Ukrajinou,“ řekl Lukašenko.

„Ale proč Ukrajina, na vojinné území de facto válka probíhá, máme tam vojenskou operaci, umírají lidé – proč Ukrajina na těchto jednáních nemá zájem?“ dodal.

Lukašenko od počátku války stojí pevně na straně ruského prezidenta Vladimira Putina, Minsk dokonce poskytl své území a základny ruským vojákům a letectvu jako zázemí k útokům proti Ukrajině. Západ proto zemi pokládá za stranu vícešlachového konfliktu a na Bělorusko svalil sankce.

Autoritářský vládce Běloruska ale v rozhovoru trval na tom, že konflikt vypukl proto, že Ukrajina „Rusko provokovala“, což tvrdí od počátku invaze. Po setkání s ruským prezidentem Putinem začátkem března řekl, že Kyjev plánuje napadnout Bělorusko a že Rusky vpadl tímto útokem zabránit. Žádné důkazy ale nepředložil.

See your annotations in time

Anytime you come back to a page you've previously annotated, you're annotations we'll appear again and you'll be able to continue right where you left off.

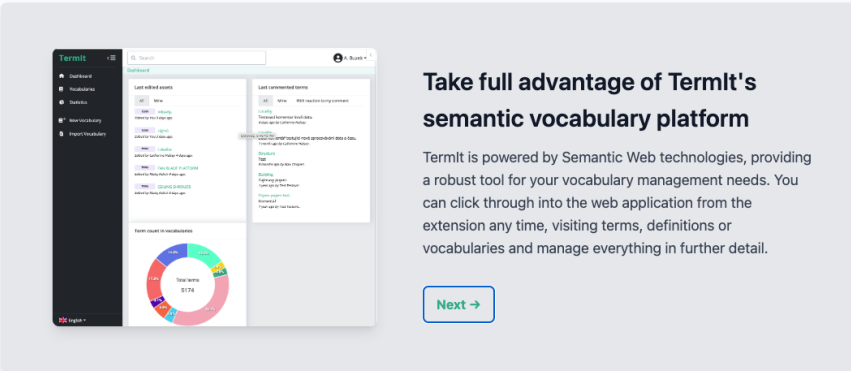
Next →

1 2 3 4 5 6 7 Skip tutorial

Figure A.5: 5. step of the tutorial – See your annotations in time

TermIt Annotate

Quick Tutorial



Take full advantage of TermIt's semantic vocabulary platform

TermIt is powered by Semantic Web technologies, providing a robust tool for your vocabulary management needs. You can click through into the web application from the extension any time, visiting terms, definitions or vocabularies and manage everything in further detail.

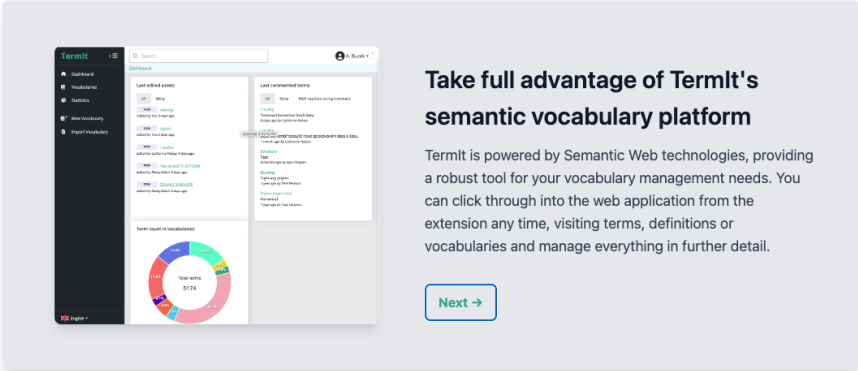
Next →

1 2 3 4 5 6 7 Skip tutorial

Figure A.6: 6. step of the tutorial – Take full advantage of TermIt's semantic vocabulary platform

Termt Annotate

Quick Tutorial



Take full advantage of Termt's semantic vocabulary platform

Termt is powered by Semantic Web technologies, providing a robust tool for your vocabulary management needs. You can click through into the web application from the extension any time, visiting terms, definitions or vocabularies and manage everything in further detail.

[Next →](#)

1 2 3 4 5 6 7 Skip tutorial

Figure A.7: 1.step of the tutorial, a brief introduction to TermIt Annotate.

Appendix B

Proof of concept

In order to verify that the proposed architecture can be implemented as specified, we implemented a proof of concept of the browser extension preceding the implementation of our work. Its main focus was on implementing the CSS selector functionality as described in Section 4.2.1 and ensuring that this approach is indeed the correct one. Fortunately, this has proven to be the case.

B.1 Functionality

The proof of concept is a Google Chrome browser extension that effectively has a single piece of functionality: to annotate (highlight) terms on all visited pages automatically. Terms to be highlighted are created by calling Annotace Service, which has been adjusted to return annotations in the form of CSS selectors instead of modified HTML. While not covering the vast majority of requirements in Sections 4.1.1 and 4.1.2, it is enough to put CSS selector annotations to the test comprehensively.

B.2 Tracked metrics

The following metrics were collected, displayed on the current page, and aggregated for further analysis to track the effectiveness of annotation selectors.

B.2.1 Elements selection

This metric indicates the success rate of selecting elements where annotations occur. When using the supplied CSS selector to get the element, there can be one of three outcomes:

- **Success** – exactly one element is returned.
- **Failure** – no elements are returned. While this number can be minimized by generating reliable selectors, a failure can sometimes happen for reasons outside our control. For example, there can be a temporary element on the page when sending the page’s HTML for analysis to the

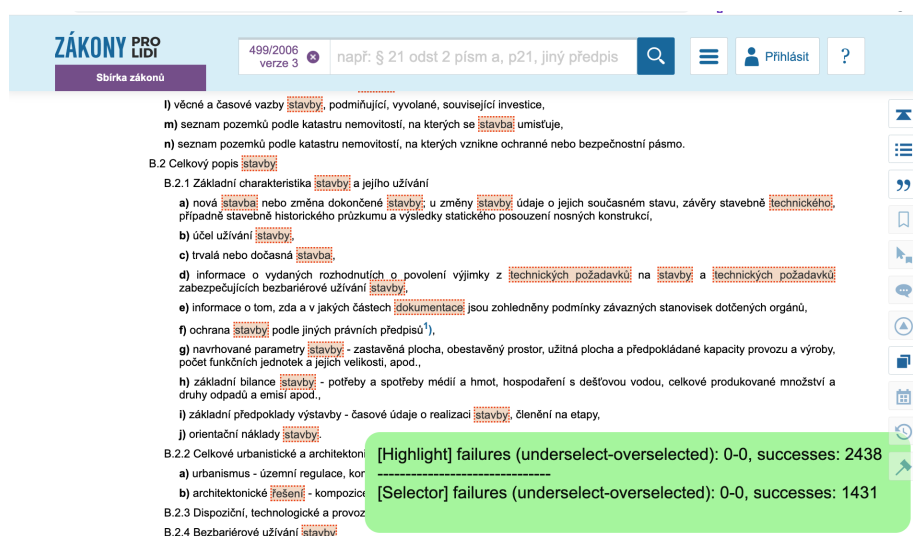


Figure B.1: Screenshot of using the extension annotations on the page <https://www.zakonyprolidi.cz/cs/2006-499>.

Annotace service. Later, that element disappears, and once Annotace returns and page highlighting starts, it can no longer be selected.

- **Overselected failure** – more than one element is selected. This can occur when the generated selector(s) is not unique enough, which can be eliminated through better implementation.

■ B.2.2 Term annotation

A selected element can contain multiple terms to be annotated. This metric measures the success rate of the individual terms being annotated within the selected elements' HTML. A term annotation can have the following outcomes.

- **Success** – exactly one term is annotated.
- **Failure** – no terms are annotated in the text. A matching text is not found at the specified position within the element.
- **Overselected failure** – more than one term is highlighted. This never happens with the current implementation as we keep the annotation's offset within the parent element.

■ B.2.3 Testing

To put our proof of concept extension to the test, we have visited 50 different web pages with it and aggregated the results into Table 5.1. It shows that the success rate of both tracked metrics has been over 99.5%, out of 7380 elements and 19604 annotations.

	Successes	Failures	Overselected	Success rate
Elements selection	7347	32	1	99.553%
Term annotation	19538	66	0	99.663%

Table B.1: Proof of Concept aggregate selector test results.

The pages that the extension was tested on were selected at our discretion, with the main focus being normative documents in Czech, such as those on zakonyprolidi.cz [36] or zakony.cz. Other pages included social media sites, discussion forums, search engine results, and news sites. More testing will need to be done to ensure our implementation’s correctness. However, our tests have proven that this is undoubtedly the correct path forward.

Appendix C

Test scenario

This section describes all steps of the test scenario and users' feedback for each section.

1. Tutorial

- Description: When the extension is first installed, its dedicated tutorial page is automatically open. It gives the user insight into how to use the extension in concise chronological steps, aimed at getting the user up to speed quickly. The tutorial is included in Appendix A.
- User rating: 3.8/5 (Ratings: 3, 3, 4, 4, 5)
- User feedback:
 - *The tutorial is illustrative, it describes the key features of the plugin. I like the inclusion of an animated image.*

2. Anonymous annotation and login

- Description: Once the user has completed the tutorial, they were instructed to navigate to a page with a Czech legal document (at *zakonyprolidi.cz*¹) that was assigned to them. They could annotate the page with automatic suggestions and create their annotations. Afterward, to save progress, they were redirected to log in to TermIt.
- User rating: 3.4/5 (Ratings: 3, 3, 3, 4, 4)
- User feedback:
 - *I am missing an instruction / help / tip for manual annotation of the page (ie a tip for manual text marking). In the "Annotator" tab, I miss the button for (re) running (automatic) page annotation*

3. Annotation as a logged in user

¹An example of a tested page: <https://www.zakonyprolidi.cz/judikat/nscr/29-icdo-20-2018>

- Description: Once the user logs into their TermIt account, they were redirected back to the page they were annotating anonymously before, where they were asked to pick a vocabulary to annotate with. Afterward, they could to operate fully on existing annotations as well as create their own.
- User rating: 4.0/5 (Ratings: 3, 4, 4, 4, 5)
- User feedback:
 - *After logging in and deleting new term suggestions, I can't restart the page annotation.*
 - *The definition of the annotation in the text cannot be edited - I can only delete the annotation and recreate it.*
 - *Everything worked nicely, except that I could not annotate the same page again.*
 - *I like that the overlapping annotations are suitably graphically differentiated.*

4. Sidebar panel and TermIt integration

- Description: Users were asked to use the sidebar panel to filter a certain type of annotations and show their location on the page. Afterward, they are assigned a definition to a term, click through the term's link to show it in TermIt UI, and then finally open the same page again, linking to that term's definition.
- User rating: 4.6/5 (Ratings: 4, 4, 5, 5, 5)
- User feedback:
 - *The delete button for annotations in the sidebar could have a tooltip with the information that it is a deletion of the annotation (so that someone does not feel that they delete the term in TermIt).*

5. Termination

- Description: Finally, users were asked to navigate to other previously annotated page as listed in the sidebar, and then are able disable the extension through the sidebar panel and see how the current page reacts.
- User rating: 4.8/5 (Ratings: 4, 5, 5, 5, 5)
- User feedback:
 - *It's nice that the extension is easy to turn on / off.*
 - *Visually very nice, I missed being able to filter through annotated websites.*

Appendix D

Description of electronic attachmens

...

```
/
├── termit-ui.....TermIt UI's code with our changes
├── termit.....TermIt Servers's code with our changes
├── annotace.....Annotace's code with our changes
├── termit-extension.....Broser extension's source code
│   ├── config.....webpack build configuration
│   ├── public.....static assets
│   ├── releases.....past released builds
│   ├── scripts.....build and helper scripts
│   └── src.....extensions' source code
│       ├── background.....background script files
│       ├── content.....content script - annotator
│       ├── shared.....shared typescript files
│       ├── styles.....CSS styles
│       ├── termit-ui-common.....shared typescript files
│       └── tutorial.....extension's tutorial
├── .eslintrc.js.....ES Lint configuration file
├── .gitignore.....Files outside of git
├── .package-lock.json.....Files outside of git
├── .package.json.....Files outside of git
├── .README.MD.....Repository description, setup guide
├── .tailwind.config.js.....TailwindCSS configuration
└── .tsconfig.json.....TypeScript configuration
```

Existing repository changes description

This is to describe what changes have been done in the 3 already existing repositories that we have submitted in the attachment on top of the extension that was written from scratch.

- **termit-ui changes:** – Added src/util/Extension.ts for syncing with browser extension, added src/model/Website.ts to represent ad website, made other minor changes to sync logged-in status and account for Website object, vs only files previously. Please refer to here for an

exact diff of the changes: <https://github.com/alanbuzek/termit-ui/pull/1/files>

- **termit changes:** As extensively described in this work, added Website, TermWebsiteOccurrence and WebsiteOccurrenceTarget entities, also added a few endpoints to store these entities. Please refer to here for an exact diff: <https://github.com/alanbuzek/termit/pull/1/files>
- **annotace changes:** Added “/annotate-to-occurrences” endpoint and “transformAnnotationOutputToOccurrences” method. Please refer to here for an exact diff: <https://github.com/alanbuzek/annotace/pull/1/files>