CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Thesis

# Version System for Document Translations

Maksim Bilan

January 2021
Supervisor: doc. Ing. Daniel Novák, Ph.D.

# BACHELOR‘S THESIS ASSIGNMENT

## I. Personal and study details

| | | | |
|---|---|---|---|
| Student's name: | **Bilan Maksim** | Personal ID number: | **453477** |
| Faculty / Institute: | **Faculty of Electrical Engineering** | | |
| Department / Institute: | **Department of Cybernetics** | | |
| Study program: | **Open Informatics** | | |
| Specialisation: | **Artificial Intelligence and Computer Science** | | |

## II. Bachelor’s thesis details

Bachelor’s thesis title in English:

**Version System for Document Translations**

Bachelor’s thesis title in Czech:

**Verzovací systém pro překlady dokumentů**

Guidelines:

Version Control Systems (VCS) are very well developed and stable software nowadays. Most of the developer teams somehow use them, at least in some form. Due to their stability and versatility, they are increasingly used not only for code tracking but also to control versions of pure text documents (Markdown, LaTeX, HTML, ...). The effectiveness of version control systems is limited by the lack of translation support. Classic VCS do not allow to monitor changes effectively and apply them to different language mutations of the same document. Now there are no existing solutions that would allow solving this problem in an automated and efficient manner.
1. Study how modern version control systems (Git, SVN, Mercurial, ...) and text comparison algorithms work. Evaluate how they can be extended and used for translation versioning.
2. Suggest an algorithm for effective identification and making changes made in one language version of a document into another language versions of the same document in the repository. Existing version control systems may be used.
3. Create a web application that implements the algorithm you came up with. The application should show to translators the necessity to change the document at a specific place and allow them to edit the document and to control the synchronization, to log all changes made by translators, including the possibility of reverting to a previous version.
4. Examine how to use machine translation to apply changes to other language versions of the document automatically.

Bibliography / sources:

[1]Blischak JD, Davenport ER, Wilson G (2016) A Quick Introduction to Version Control with Git and GitHub. PLoS Comput Biol 12(1): e1004668. https://doi.org/10.1371/journal.pcbi.1004668
[2] BARABUCCI, Gioele, et al. Document Changes: Modeling, Detection, Storage and Visualization (DChanges 2016). In: Proceedings of the 2016 ACM Symposium on Document Engineering. ACM, 2016. p. 5-6.
[3] TAN, Ping Ping, VERSPOOR, Karin, MILLER, Tim. Structural Alignment as the Basis to Improve Significant Change Detection in Versioned Sentences.
[4] WOON, Wei Lee, WONG, Kuok-Shoong Daniel. String alignment for automated document versioning.
[5] MURATA, Masaki, ISAHARA, Hitoshi. Using the diff command for natural language processing. arXiv preprint cs/0208020, 2002

Name and workplace of bachelor’s thesis supervisor:

**doc. Ing. Daniel Novák, Ph.D., Analysis and Interpretation of Biomedical Data, FEE**

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

Date of bachelor’s thesis assignment: **10.01.2020** Deadline for bachelor thesis submission: **05.01.2021**

Assignment valid until: **30.09.2021**

| | | |
|---|---|---|
| doc. Ing. Daniel Novák, Ph.D. | doc. Ing. Tomáš Svoboda, Ph.D. | prof. Mgr. Petr Páta, Ph.D. |
| Supervisor’s signature | Head of department’s signature | Dean’s signature |

## III. Assignment receipt

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

| | |
|---|---|
| . | |
| Date of assignment receipt | Student’s signature |

## Acknowledgement / Declaration

I wish to acknowledge my supervisor doc. Ing. Daniel Novák, Ph.D., for providing such an interesting bachelor's thesis topic, guidance, and constructive criticism. I want to thank my friends for being supportive of my non-university interests. I would also like to extend my gratitude to my girlfriend; she supported me mentally for an extended period and financially for the last year.

Finally, my special thanks to my family. To my parents for their love, help, and support during my whole life and to my younger brother Dima for his interest in what I am passionate about.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, January 5, 2021

--------------------------------

Signature

# Abstrakt / Abstract

V této bakalářské práci se zabýváme implementací verzovacího systému pro překlady dokumentů. Takový systém zaručuje konzistenci více instancí dokumentu v různých jazycích. Pomáhá řídit práci překladatelů zavedením verzí a recenzí. Implementujeme webovou aplikaci, která splňuje tyto požadavky pomocí webového aplikačního frameworku Django.

**Klíčová slova:** systém pro správu verzí, diff, překlady dokumentů, vícejazyčné dokumenty, systém pro správu překladů

**Překlad titulu:** Verzovací systém pro překlady dokumentů

In this bachelor's thesis, we are dealing with the implementation of a version control system for document translations. Such a system maintains the consistency of multiple instances of a document in different languages. It helps to manage translators' work by introducing versions and reviews. We implement the web application which meets these requirements using the Django framework.

**Keywords:** version control system, diff, document translations, multilingual document production, translation management system

# Contents /

# Tables / Figures

# Chapter 1
# Introduction

Due to globalization, more and more companies are forced to use documents that must be translated into different languages. As a result, companies are faced with document inconsistency problems and translation efficiency. Document inconsistency may result in the absence of some crucial parts of a document, which is not acceptable when documents are considered the same up to a different language. Translations delays may cost a lot of money for the company, e.g., when it is impossible to launch a product without translated user guides.

The final application called LinGit solves both of these problems by generating a strict workflow for translators, which guarantees document consistency and gives translators and moderators essential tools.

## 1.1 Glossary

Let us introduce definitions of some terms to reduce ambiguity. *Document* represents a document from the real world, which may exist in multiple languages, e.g., user manual. The application's fundamental element is *document mutation*, which represents the document in some language (e.g., user manual in the French language). For every document mutation *source-mutation* must exist (usually in English language[1]). Changes from source-mutation will propagate to other document mutations. For every document change, such as adding or removing some text, the application generates corresponding *commit*. Later, several consecutive commits in a source-mutation are indicated as a *document version*. Full list of terms follows.

- **Document** represents a document from the real world.
- **Document mutation** is a text in some language, which corresponds with other mutations covered by one document.
- **Source-mutation** is a document mutation that is a source of changes to propagate in other mutations.
- **Commit** is a change in some document mutation, e.g., adding or removing text.
- **Document version** is a collection of commits in a source-mutation.
- **Moderator** is a user that checks technical correctness of translations, creates new versions, and generates text in source-mutations.
- **Translator** is a user that translates source-mutations changes into chosen non-source-mutations to make text consistent in different languages.
- **Document consistency** is an abstract term that defines translations equality in a sense that if all non-source-mutations are translated into the source-mutation language, they will be equal in every relevant chapter.

## 1.2 Objectives of the thesis

The thesis's main objective is to implement a web application that should help organizations collaborate with translators in order to translate multilingual documents. It should help to make changes in a document and propagate them to all document languages. It should also help a translator find a place where the change should appear in a resulting text. It should have a possibility of translation review and keep track of the history of all the changes. Also, it should have a possibility of reverting the document to a previous version in case of incorrect changes were made.

There should be three user groups to maintain the application:

- **Translators** should be able to see all the document mutations that need to be translated. The application should support a translator in the way of creating a list of changes that should be translated, so after translations, the affected documents are consistent.
- **Moderators** should be able to review translations in terms of technical characteristics, e.g., to check paragraphs' location. A moderator should not check translation correctness. It should also be possible to edit source document mutations, which will cause the creation of new document versions that will be translated later. A moderator may want to see all documents in the system and all document mutations to check changes made by translators or other moderators.
- **Administrators** are taking care of the application. They may add new users, user groups, add users into user groups and set up translation languages for individual users.

# Chapter 2
# Existing approaches

Many tools might help translators and customers reduce the cost and time of the translation process. In this Chapter we discuss some of them. First, we describe what version control systems are in Section 2.1. In Section 2.2, we review the "translation memory" approach. In Section 2.3, we talk about machine translation. Finally, in Section 2.4 we describe "translation management systems" and disassemble some of them to understand the differences between them and LinGit.

## 2.1 Version Control Systems

Software developers widely use Version Control Systems (VCS). They are particularly useful for larger projects where there is a need for developers' collaboration. Let us describe them in more detail.

VCS are the systems that enable storing historical versions of the source code and retrieving them when needed. VCS are thoroughly described in [2]. Those systems share common terminology. Let us describe it briefly:

- **Repository**. VCS are saving version information for a set of files, which are usually stored in the same directory. A repository is a place where the changes are stored.
- **Commit** is a change in the repository.
- **Branch**. Inside one repository might be more parallel development lines. They are usually called branches.
- **Tag** is a snapshot of a branch. They are used for better recognition and navigation between significant changes in the repository.

## 2.2 Translation memory

Translation memory is a database of previous translations. Usually, it works on a sentence-by-sentence basis, as described in [3]. Translation memory provides a translation by finding the sentence that matches the source sentence as much as possible.

Almost every TMS utilizes translation memory because it makes translations more consistent, and it is a relatively easy to use tool. Translation memory is used for quite a long time from mid 1990s[3], and it is considered reliable, actively maintained, and field-proven software.

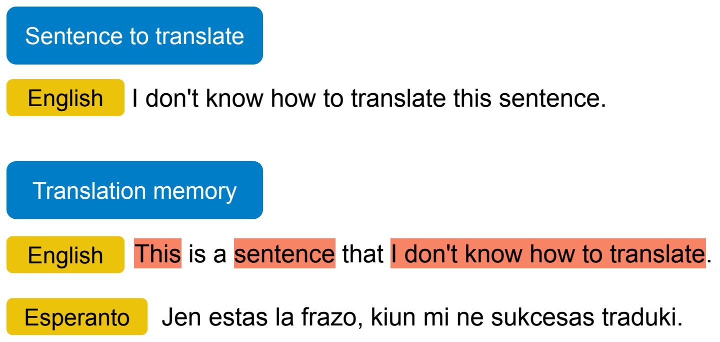

Sentence to translate
English
I don't know how to translate this sentence.
Translation memory
English
This is a sentence that I don't know how to translate.
Esperanto
Jen estas la frazo, kiun mi ne sukcesas traduki.

**Figure 2.1.** Translation memory example. Sentence matching and sentence translation was provided by `https://tatoeba.org`

As we might see in figure 2.1, there is a sentence that needs to be translated from the English language to Esperanto. Translation memory finds an already translated sentence that either equal to the original sentence either matches a sentence that is as similar to the original as possible.

As a consequence of how the Translation Memory is designed, it becomes more effective for the text with an increasing number of repetitive phrases. It is the case for technical, legal, or even medical texts. The article [4] confirms our hypothesis. TM usage rate declining as the repetition levels decrease, e.g., for more general texts and literary translations.

## 2.3 Machine translation

Machine translation (MT) is an automatic translation from one language to another done by a machine. It is also a sub-field of computational linguistics, where MT is scrutinized. As described in [5], machine translation systems can be divided into 3 main parts:

- **Rule-based machine translation (RBMT)** consists of two parts, rules and lexicon [6]. RBMT provides high translation accuracy, but this approach is very labor-intensive because rules must be provided by a human. On the other hand, RBMT might be used with a relatively small corpus[1].
- **Example-based machine translation (EBMT)** utilizes bilingual corpus[2] with parallel aligned texts. The idea is to translate by analogy. This approach is suitable to translate context-dependent language entities, such as phrasal verbs.
- **Statistical-based machine translation (SBMT)** also as SBMT uses bilingual corpora. It creates three models: language, translation, and decoder model. SBMT provides a translation by combining those models, while EBMT provides a translation by adapting already existing translation.

There are many MT approaches, and every approach is suitable for a different kind of text.

Nowadays, machine translation still cannot fully replace the translator, as it often makes mistakes and cannot always get into context. But it might be used as an aid for

[1] A corpus is a collection of texts in some language.
[2] Bilingual corpus consists of two corpora when texts from one corpus are translations of the texts from another.

translators, same as translation memory. It is possible to use both of these tools at the same time. First, try to match sentences from the local TM. For every sentence that does not have a pair in TM, MT might be used. To achieve more reliable results, it is better to connect consecutive sentences before passing them to MT engine to give a context to it.

One of the things to consider before using MT is confidential information. For example, according to the Google Privacy & Terms in [7], you give Google a worldwide license to use, host, store, and produce content provided via Google's services. That means that information passed to translation might be used to train the translation algorithms. To avoid that, it is possible to use MT engines that are more secure from a confidential perspective. However, they may suffer from a lack of training data, resulting in less accurate translations.

By using such an approach, a translator will see fully translated text. Some sentences do not need any corrections. Most probably, it would be the sentences from a translation memory. Some of them a translator will correct a bit to fit the context, and some will require a complete rewrite. The last case should dissolve by the constant usage of the translation memory.

## 2.4 Translation Management Systems

Translation management systems are described in the patent [8] created by David Lakritz. He describes them as a computer environment, which detects changes in master language and notifies the user which document mutations require translation. Later TMS coordinates the delivery of a translated document back to the user for translation installation and optional review.

Let us describe some of the most popular TM systems in the following subsections.

### 2.4.1 Pairaphrase

Pairaphrase is one of the popular Translate Management Systems. The company was found in 2014 by Rick Woyde. He founded such a tool to offer more private Google translate alternative[9]. According to Google Privacy & Terms, you allow Google to use your content to host, distribute, and publish your content as described in Google terms of service [7].

It is focused on data security and supporting many document formats. Using so-called File Translator, Pairaphrase is compatible with 24 file types, e.g., Microsoft PowerPoint, Excel Spreadsheets, or PDFs.

Microsoft and Google translation engines are used inside Pairaphrase. It helps Pairaphrase to support over 100 languages for machine translation.

Pairaphrase uses its own Dynamic Machine Learning technology in its TMS to use translation memory (described in 2.2) more efficiently.

### 2.4.2 Memsource

Memsource is AI-powered TMS. The company was founded in 2010 in Prague. For 10 years, they have constantly been improving their product, and by the end of 2017, they had around 80 employees working on Memsource. It is essential to draw on such teams' experience to discover potential mistakes in our application design.

The main Memsource features are:

- **REST API**. By using so-called Connectors, it is possible to integrate Memsource into 3rd-party systems. For example, Dropbox to translate file content, GitHub to send repositories content for translation automatically, and WordPress to translate already written posts.
- **Machine translation** helps to score different translation engines to chose the best fitting to the content.
- **Translation memory** (you can read more about it in Section 2.2)

The software utilizes artificial intelligence to identify content that might be translated automatically before translation needs to be done by a human translator.

### 2.4.3 Transifex

Transifex was originally open-source software, but from mid-2012, there were not added any major changes. The project is written in Django and Python.

Service core features:

- **File & content hosting** enables to store content and translation inside the platform.
- **Team collaboration**. Service introduces user roles such as product managers, developers, and translators, etc.
- **Translation tools**. The service provides many translation tools, such as translation memory (2.2), glossary, or screenshot text mapping, to bring a translator into the context of translation.

### 2.4.4 Main differences

The main difference between Translation Management Systems described above and LinGit is that TMSs are mostly focused on translating the document from one version to another, e.g., by using translation memory to optimize translation flow. At the same time, our application provides a continuous translating cycle of every document without the need for external managing. It means that it is possible to have any number of newer versions in the source mutation, while LinGit will provide the list of needed changes in other document mutations, leading to document consistency.

# Chapter 3
# Application design

In Section 1.2, we introduced the thesis objectives. Our primary goal is to be able to track changes inside multilingual documents that are constantly changing. The application should indicate to translators the need to make changes in a certain place. It also must provide a correct workflow for translators to make document mutations consistent and for moderators to review translations. In Section 3.1 we show how data are organized inside the application. In Section 3.2, we discuss why a relational database is used inside the project and its design.

## 3.1 Commits model

In Section 1.1, we introduced the core concept of the application. Let us narrow it down.

All changes made by a user after the editing page open (you may see how the editing page looks like in Subsection 5.2.4) and before the Save button click are stored in the **commit** objects.

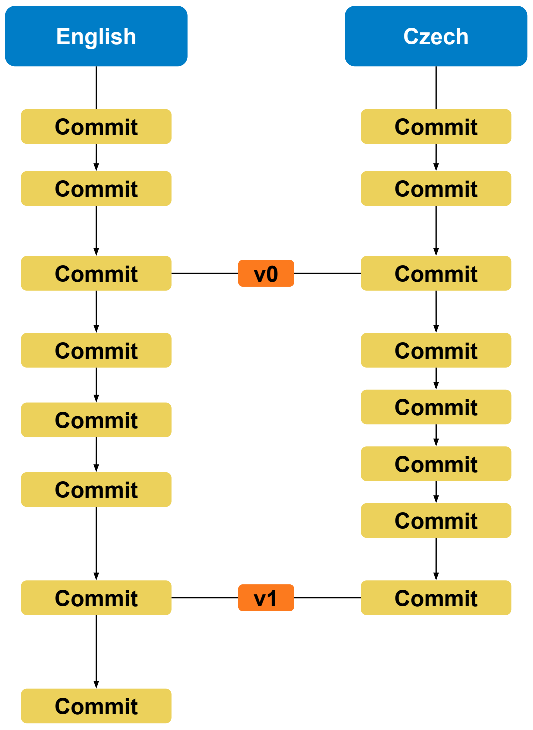


**Figure 3.1.** Commits model

On the left side of Figure 3.1 we see the English language commits. They are created one after another. On the right side, there is the Czech language document mutation. Commits in different mutations are created independently, but we may consider them equal after some commits in the Czech mutation. It means that they are sharing the same version tag at this moment.

As changes are created independently, there might be a situation when the source-mutation is several versions older, e.g., the English mutation has version tag **v13**, when the Czech mutation has only **v10**, supposing that versions are sorted linearly, **v1** is older than **v2** and so on. The LinGit system will lead a translator through the versions **v11**, **v12**, **v13**, by providing relevant version's commits.

## 3.2 Database

One of the key objectives at the beginning of the development process is to choose where to store the data. After longer considerations between NoSQL and relational databases, an SQLite database was chosen. Let us explain why such a choice was made and why we considered NoSQL databases to store documents.

One of the possible solutions is to store already parsed document mutations in a tree, and each time the document is needed to be viewed, generate it from such an object. This approach was considered because of text natural hierarchy, where there are different text parts, such as Introduction, Body, and Conclusion (see Figure 3.2). Every part of the text consists of smaller parts, such as paragraphs, sentences, lists, etc.

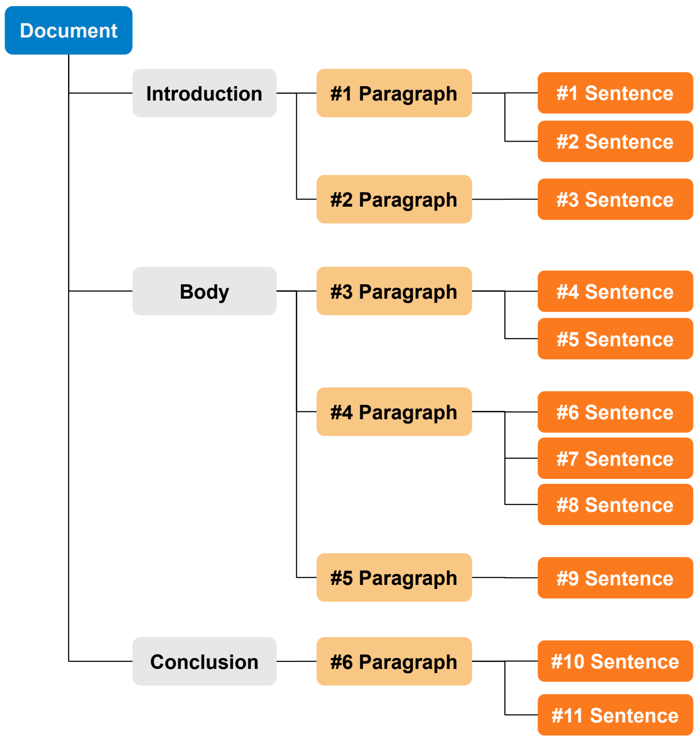


**Figure 3.2.** General document structure

This approach might help to detect more deep structural changes in the document, such as paragraph moving. To implement this idea, it is possible to use the approach described in [10]. The algorithm accepts two trees and generates a descriptive list of changes (edit script) that gives a sequence of actions to make one tree from another.

It turned out that in the natural translation process, most of the features of this algorithm are irrelevant, as every change is most probably an insert. So we will not benefit from using the tree structure. That is why we turned to relational databases.

### 3.2.1 SQLite

As a relational database, we choose between PostgreSQL and SQLite. SQLite is a better option for application development for the following reasons:

- Django provides SQLite by default
- SQLite database is stored in the file. That means that it is effortless to send the whole database to someone, e.g., to provide example data.
- No need to run database server, SQLite is serverless[11].
- Migration from SQLite to PostgreSQL is straightforward by using Django framework (more about Django you may read in Implementation section 4.1). There are two reasons for that.
  - Django provides Object-Relational Mapping (ORM), meaning that we are not using row SQL commands.
  - Django may load database structure and populate it with data by previously connected database dump (SQLite in our case).

The main SQLite strength is its easiness. Migration to PostgreSQL might be considered when the number of users working simultaneously grows. Because of SQLite default implementation of atomic commit and rollback, it is impossible to read from the database during the writing.

By default, SQLite uses the rollback journal to handle concurrent access to a database. The whole database file is locked with an exclusive lock during the writing to avoid concurrent reads and writes. To solve such problem, “Write-Ahead Log” (WAL) was introduced[12]. With WAL database itself becomes much faster, it also provides more concurrency when writers do not block readers and vice versa.

This means that by using WAL, we may get rid of such a limitation of SQLite.

### 3.2.2 Database schema

In this Section we will describe the schema of the resulting database. Database schema contains service tables, among others. Such tables are called **auth_***, **django_***, where asterisk replaces an arbitrary symbols sequence.

Tables names consists of two parts **«application-component-name»_«database-table-name»**. Let us write only database tables’ names to avoid cluttering.

- **Document**. This table stores all documents. Every document has a name and its comprehensive description.
- **Language**. All supported languages are stored in this table.
- **Mutation** represents a document mutation, e.g., the Czech version of the User Manual. Every mutation has a description and **is_main** characteristic, which determines whether a current mutation is source mutation for this document. Document mutation has a reference to some document and language, meaning that there is a document mutation in some language, e.g., User Manual in the English language.

- **User**
  - The first user field is **password**. The field contains password hash, which is generated by default by using PBKDF2 with SHA-256 hash function.
  - **is_superuser** and **is_staff** are the fields that determine access to the Administration section generated by the Django framework. Superusers can create, read, update, and delete records on the Administration page. Staff users only have access to the Administration page.
  - **is_active** field denotes whether a user is still active. If a user is not active, the default function for permission control will return `False` for each permission for such user.
  - Fields names for **last_login**, **username**, **first_name**, **last_name**, **email**, **date_joined** are self descriptive.

- **Language translators**. Records in this table determine what languages do translators know.
- **Commit**. Every commit contains **text** with a change made by user (**committer_id**). The change is made for a specific document mutation (**mutation_id**). A field named **translated_commit_id** is a recursive reference to the same table, which defines a commit with translation the chosen commit. Might be empty, meaning, that such commit is not already translated. Fields **description**, **create_date** are self descriptive.
- **Version** consists of **name**, which is intended for short version names such as a `v14`. The field **contains** an ID of a user which was added a version. Fields **description**, **create_date** are self descriptive.
- **Documents review**. This table stands for a reviewing process. When a translation is done, there appears a new record which contains the date (**create_date**) where the translation was finished, **commit_id** of the last commit from translations, and **version_id** which shows to what version this mutation was translated. After the translation is approved the **id_approved** becomes `True`, **approve_time** is automatically updated and the field **approved_by_id** will be updated with ID of a user which approved current translation.
- **User permissions**. Permissions in Django are given either to the groups either to users. For example, the Translators group has permission to the translation page, meaning that every translator has permission for that. But they might be assigned to a user, e.g., give access to someone to review translations.

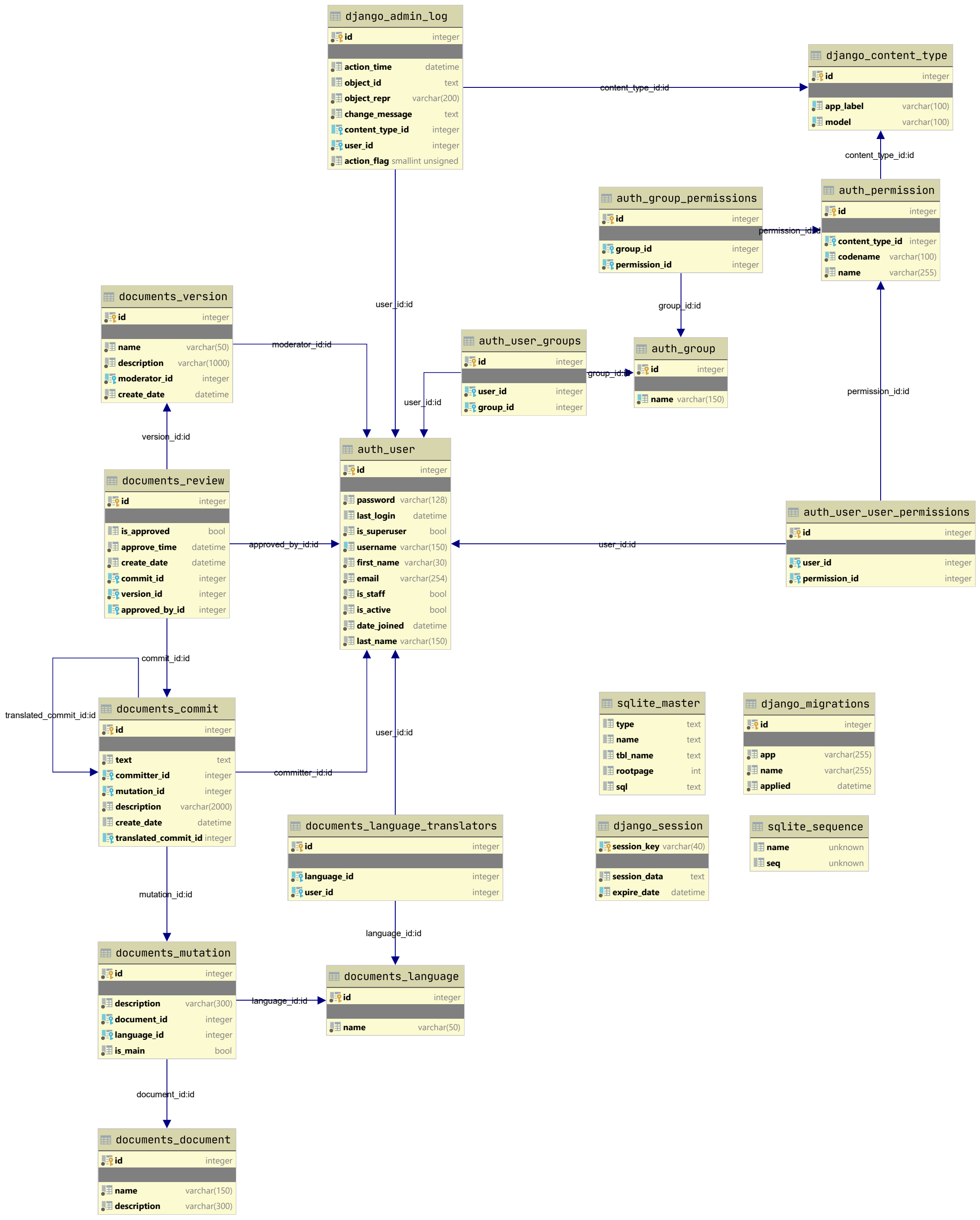


**Figure 3.3.** Database schema

# Chapter 4
# Implementation

This chapter shows how the application is organized from the inside. First, we discuss the choice of the framework in Section 4.1. Then we talk about Markdown language in Section 4.2, which translators will use inside the application. After this, in Section 4.3, we show the library called wikEd diff, which is used to show the text changes. In Section 4.4, we describe the implemented change detector, capable of scrolling the text to the place where the translation is needed. Finally, we describe different methods used to test the application in Section 4.5.

## 4.1 Python and Django

Even during the concept development phase, it was clear that the application should be web-based. It should be easily accessible to every user so that there would not be any problems with its installation or platform restrictions. The user only needs to have a web browser.

According to the Stack Overflow website survey, which is based on answers of 63,585 software developers[13], the Django framework is the 8th most popular web framework, including frontend frameworks like jQuery. By excluding frontend frameworks from the list, Django is the 4th most popular backend web framework.

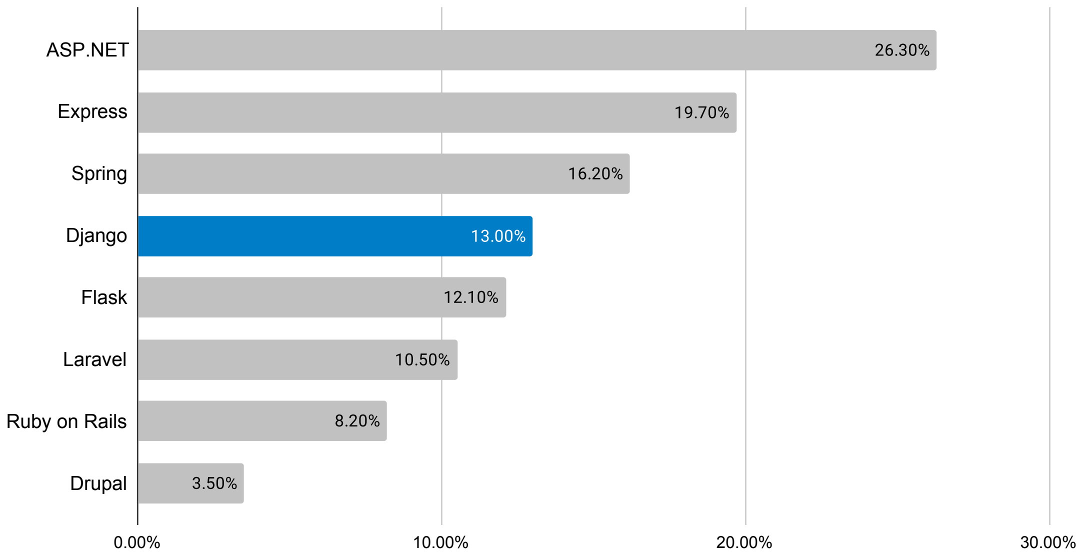


**Figure 4.1.** Backend web Frameworks popularity - Stack Overflow survey 2019[13].

In Figure 4.1 we may see that the Django framework is used by 13% of developers, which makes it the 4th most popular backend web framework.

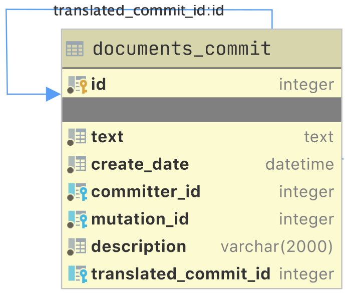
To detect most popular frameworks, Stack Overflow used “select all that apply” technique. That means, that a developer had a list of different web frameworks and he might to choose all frameworks that he uses. More about survey methodology you may read in the last section of [13].

By using Django, we take all benefits from the Python world. Python is easy to learn, powerful programming language[14] that provides a wide range of different libraries and is popular in the machine learning field, which might be helpful in the further development of the app. Django itself provides high-quality documentation, a high-security level, and a variety of essential tools for development.

Django uses database-abstraction API that helps to make to Update, Read, Update and Delete (CRUD) objects in database[15]. Such API is called Object-Relational Mapping (ORM). To use Django ORM, a developer should create data models, which are Python classes inherited from Django abstract model. As an example of such a class let us consider Commit model from the application:

```
class Commit(models.Model):
    committer = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.DO_NOTHING
    )

    versions = models.ManyToManyField(Version, through='Review')

    description = models.CharField(
        max_length=2000,
        null=False
    )

    mutation = models.ForeignKey(
        Mutation,
        on_delete=models.DO_NOTHING
    )

    text = models.TextField()

    create_date = models.DateTimeField(auto_now=True)

    translated_commit = models.ForeignKey(
        "Commit",
        null=True,
        on_delete=models.DO_NOTHING
    )
```

This would create a following database table (ignoring the objects associated with the commit object):



| documents_commit | |
|---|---|
| id | integer |
| text | text |
| create_date | datetime |
| committer_id | integer |
| mutation_id | integer |
| description | varchar(2000) |
| translated_commit_id | integer |

**Figure 4.2.** Database table: Commit

A more detailed analysis of the database schema could be found in Section 3.2.

## 4.2 Markdown

Markdown is a markup language convertible into HTML. This feature is quite useful for a web application for two reasons: Markdown has understandable and interpretable syntax so that translators may use some of its features without the need for long training. Markdown, as a text convertible into HTML, has a tree structure. Such text property allows us to work with a text as with a mathematical object, e.g., to detect differences in two document mutations by comparing its structures. Inside the application, Markdown language is used for text representation, Martor[16] editor for Django was chosen as a text editor. The Markdown editor provides syntax highlighting and live preview generation, so the user knows exactly how the output text would look like.

Markdown usage does not restrict the user because users may still use plain text without using Markdown-specific tags. The translator does not need to create the document from scratch, so he could only edit plain text pieces. The rendered text will still have an identical structure.

## 4.3 wikEd diff

As a tool to show differences between the two texts, wikEd diff library[17] was chosen. It is a JavaScript library to show text differences inline, so people who are not familiar with different kinds of differs would instantly understand what changes were done. This library is used to show differences between some Wikipedia article versions. It means that such a tool is used by more than 298,000[18] Wikipedia active users[1]. This library was chosen because it is extensively tested and has proven reliability.

### 4.3.1 Key features

- Detects moved pieces of text
- Unchanged pieces of text are omitted, which is particularly useful for longer texts.

[1] According to Wikipedia, an active user is a user who makes at least one change or other action on Wikipedia in a given month

### 4.3.2 Legend

WidEd diff has an online demo with different options. We will briefly explain how does it work from the user perspective.

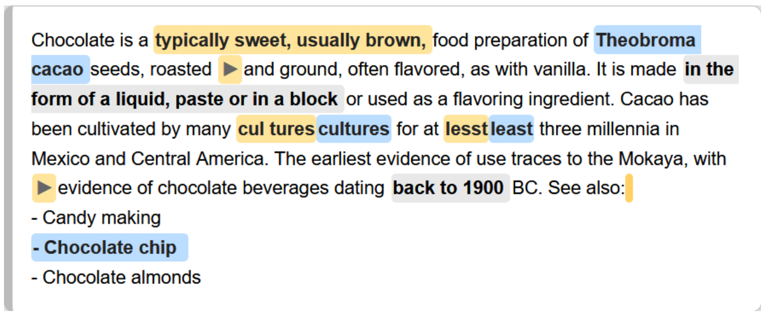

Chocolate is a typically sweet, usually brown, food preparation of Theobroma cacao seeds, roasted ▶ and ground, often flavored, as with vanilla. It is made in the form of a liquid, paste or in a block or used as a flavoring ingredient. Cacao has been cultivated by many cul tures cultures for at lesst least three millennia in Mexico and Central America. The earliest evidence of use traces to the Mokaya, with ▶ evidence of chocolate beverages dating back to 1900 BC. See also:
- Candy making
- Chocolate chip
- Chocolate almonds

**Figure 4.3.** Resulting diff for example text from web[1]

Let us describe the main text blocks in Figure 4.3:

- Inserted piece of text is colored blue.
- Removed text is colored in tints of creme brulee[2] (let us call it yellow).
- Moved blocks of text are colored gray.
- Previous text position for moved text is marked as a triangle turned to the side with a yellow background.

Special characters are shown by pointing the cursor over them.

- Newline ¶
- Space ·
- Tab →

People with colorblindness can also use this tool. WikEd diff uses span classes inside the HTML document structure (such as *wikEdDiffDelete*) in order to mark different parts of a text. For example, special symbols are visible by holding the cursor over the changed piece of text (+, - and ◀ signs) to distinguish between different colors.

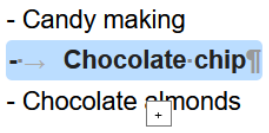

- Candy making
-→ Chocolate·chip¶
- Chocolate almonds

**Figure 4.4.** Tab symbol, space, and the title for the inserted text block

## 4.4 Naive change detector

Working with long documents, maybe not very convenient for translators. A translator may need to scroll hundreds and, in some cases, thousands of rows to find the place that needs editing. We implemented a naive change detector to solve this problem. It predicts where the translator should change the language mutation.

Our naive change detector is not complicated. Its algorithm will be described in Subsection 4.4.1. Then, in Subsection 4.4.2, we will provide some benchmarks to know if something better must be invented. Finally, in Subsection 4.4.3 we will explain possible improvement of our algorithm using a converter of Markdown text into a directed rooted tree.

[1] http://cacycle.altervista.org/wikEd-diff-tool.html
[2] https://www.htmlcsscolor.com/hex/FFE49C

### 4.4.1 Implementation

The idea behind the detector is relatively simple. To start doing the required changes, a translator needs to find the first change from the generated text diff.

On translation page described in Subsection 5.2.6, on the right side we may see the output of the WikEd[4.3] library. Thanks to this library, we may iterate through the diff fragments to know where the first change occurs and compute the text's total length. Knowing this, we may find where potentially the first change was made in the mutation, which needed to be translated simply by jumping to the corresponding percentage of the lines.

All the changes are performed on the frontend side:

1. WikEd computes the changes provided by commit which is needed to be translated (e.g., English language)
2. By iteration through the WikEd object fragments, we find the first change and compute overall text size
3. We compute where the first change occurred as a percentage of the number of rows in the text
4. Using Ace[1] editor we find number of lines in the document mutation which is currently in edit (i.e. Czech language mutation)
5. We find the row which is located on the same percentage of rows as the first change
6. Ace editor scrolls to the chosen row and highlights it

### 4.4.2 Benchmark

To know Naive change detector performance, we need to see how it works on larger texts. To test it, let us take two texts:

- User Manual in the English language, which consists of 1244 rows
- A document mutation of the first text in the Czech language with 1004 rows of text

Via LinGit, we will change the English version of the manual in different parts of it. Let us make eleven changes, located at every ten percent of the text measured in rows. We will make a change at the beginning of the text, which corresponds to a change made at 0% of the English text. Then we will make a change at 10% of the text, which corresponds to the 124th row, and will repeat the process for 20%, 30%, and so on until 100% of the text.

By having these changes in the English language version of the User Manual, we create a new version in LinGit to be able to “translate” such changes into the Czech language. Then we open the translation interface which is described in Section 5.2.6.

For every change made in the English version, the Naive change detector scrolls to some row in the editor on the left side. For example, for the change made in 30% of the English language document mutation, the Naive change detector predicts that translation must be done in the 301st row of the Czech language document mutation, but in fact the change should be done in the 312th row. The difference is 11 rows, as shown in Figure 4.5.

[1] Ace editor (`https://github.com/ajaxorg/ace`) is used by Martor editor described in 4.2

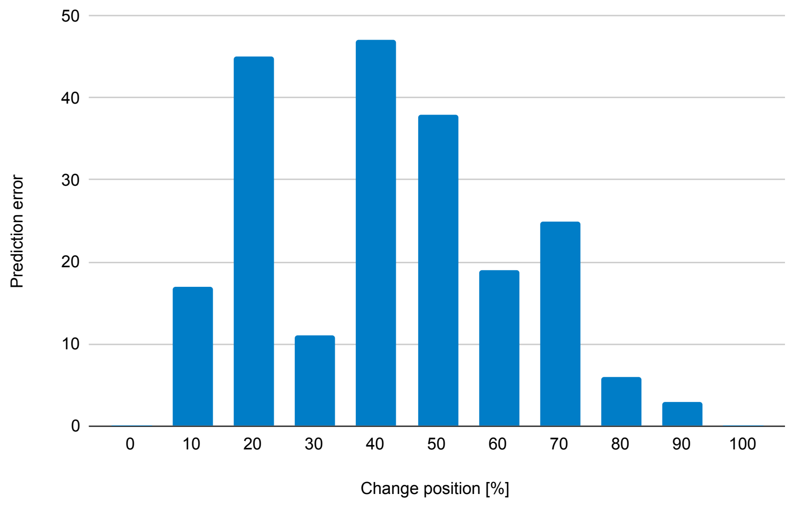


**Figure 4.5.** Naive predictor error

In Figure 4.5, the vertical axis shows prediction error, counted in rows. On the horizontal axis, there is a change position in the original mutation. For example, when the change was made in 70% of original document mutation, the prediction error was 25 rows, meaning that a translator needs to go 25 rows far from the predicted place.

To better understand the predictor performance, let us compute some common statistics measurements, such as mean and standard deviation.

```
Mean: 19.18 rows
Standard deviation: 17.53 rows
Minimum error: 0 rows
Maximum error: 47 rows
```

We can see that most of the time, the error is 19±18 rows. For example, the editor window size contains 52 rows of text for the 1920 x 1200 screen resolution, which means that a translator will need to scroll only a little bit from the predicted place. The prediction error is smaller than the screen height.

The main reason for bigger deviations in 20, 40, and 50% of the text, are the changes that were not reflected in the translated document. For example, one of the changes was the feature relevant to the source language but missing in the translation because of different accessories delivered with phones based on regional legislation.

### 4.4.3 Possible improvements

During pre-implementation, we created the tool, which can parse a Markdown document into a parse tree. This tool allows us to find a path to a change, i.e., we may determine where a change was made in the document structure, which we may see in Figure 3.2. For example, a description of a change might look like this: "The change was made in a *list* which is located *Section 3* which is located in *Chapter A*". This approach may be combined with Naive Change Detector. The tool is not built into the application editor yet, because of the discursiveness of its contribution, which should be further examined.

Based on a change detector's prediction, we may find a tree node where the change was made. One of the main reasons to do that is to find a place for a translator from which it might start looking for a change. For example, instead of pointing to the middle of some paragraph inside a section, we may show a translator the first section line to get the translator an idea of what text part he is currently in.

Let us describe how the tool works. We use the Python-Markdown project[19] to generate a parsing tree as shown in Figure 3.2. Tree nodes are abstractions over text parts (sections, paragraphs, lists, etc.). The Python-Markdown project was chosen because of its high popularity among developers and high extensibility. Our tool's core part is Tree Processor from Extension API, which facilitates modifying of internal Python-Markdown ElementTree object[20]. Python-Markdown converts a Markdown document to HTML and then provides us with a simple HTML tree structure. We enrich this structure with additional information about the nodes, such as the node's position in the whole document and its level. This information will help to detect the place where the change was made. Finally, a node of the parsing tree has the following structure:

- **Level**. The root node has level 0.
- **Text**. This item contains node real text. For example, if we are in the header node, we will see the header's text.
- **Tag**. The tree is generated by rendered HTML from the original Markdown text, so one of the useful properties is its tag, e.g., header or paragraph and even list tags. This helps to derive the node's prototype.
- **Children nodes**. List of the "Node" objects, which might be empty, meaning that we are working with the leaf.
- **Parent node**. For the root node, it is "None" object.
- **Start position**. The number indicating where the current node's text starts.
- **End position**. The number indicating where the current node's text ends. For example, the top-level root node will contain the whole text, starting at position 0 and finishing at a position equal to the text length.

Our parser is located in the "markdown_change_detector" module. To show extension possibilities, the module provides the class "MarkdownChangeDetector", which accepts two texts. The first text is the original text, and the second text is the same text with some changes. Let us call them "start document" and "final document". To generate the parse tree we need to use "run()" method on "MarkdownChangeDetector" instance. This module has two possibilities:

- Generate parsing trees for a start and final documents
- Generate the path where the change is located in a human-friendly manner

Inside the extension, we use Google's open-source library called Diff Match Patch[21]. It is used to generate an object with text differences. Previously it was used to generate an HTML version of the differences to the application's frontend, but it was replaced by the WikEd diff tool, described in detail in Section 4.3.

As we mentioned before, this module generates a `path to the change`, which is stored in the "changes_with_description" attribute of the "MarkdownChangeDetector" instance. It may look as follows:

```
There is a change under the header 'Aer deus deiectam auferat'
=> 'Mihi hanc iussit timorem te mille' in ordered list
```

## 4.5 Unit tests

To test vital application parts, we use the Python built-in `unittest` module[22]. Tests are stored with **test_*** filenames in **test** module and in each application part folder (in Django terminology, they are applications).

Because we test the web application, many tests might need to use some data from a database. To do that, we need to isolate test environment data and our production data. The Django framework creates a temporary test database before tests run and destroys it after, regardless of whether the tests were passed or failed.

To fill up the temporary database `setUp()` method is used inside a test object. The ORM is used to populate the database, as we may see on the code snippet below.

```
class TestDocumentServices(TestCase):
    def setUp(self):
        self.user = create_stuff_user()

        self.v1 = Version.objects.create(
            name='v1',
            description='First test version',
            moderator=self.user,
            create_date=datetime(2020, 7, 23, 15, 0, 0, 0,
                                 pytz.timezone("Europe/Prague"))
        )
```

The objects **user** and **v1** might be in all the tests which belong to the DocumentServices tests. Another possibility is to get every document by using ORM.

```
v1 = Version.objects.get(name='v1')
```

# Chapter 5
# Usage

In this Chapter we talk about application usage. In Section 5.1, we show how to run the application. In Section 5.2, we describe the application's main parts.

## 5.1 Get started

Since our application is web-based, we need to create a server that will respond to requests. To do so, perform the following steps.

1. Open terminal at the project root (lingit folder)
2. Create and activate the virtual environment

```
virtualenv venv -p python3 # creates new virtual environment
source venv/bin/activate # activates virtual environment
```

3. Install requirements

```
pip3 install -r requirements.txt
```

4. Make database migrations

```
python3 manage.py makemigrations
```

5. Apply database migrations

```
python3 manage.py migrate
```

6. Create superuser

```
python3 manage.py createsuperuser
```

   Enter login, email, and password.
7. Run server

```
python3 manage.py runserver
```

8. Open login page

```
http://127.0.0.1:8000
```

## 5.2 Application structure

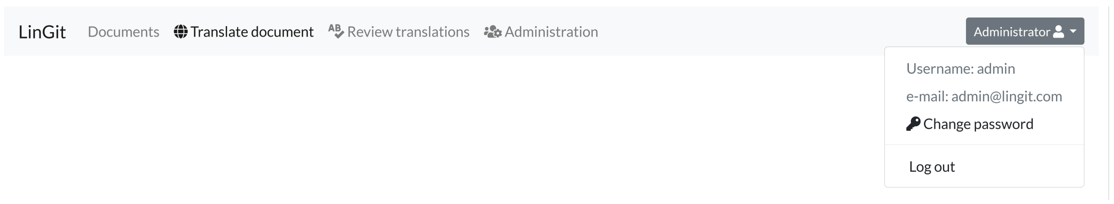
This Section shows the application's main parts. This should help to understand the application's logic better.

### 5.2.1 Administrator panel

The Django framework generates this part of the application automatically. The page is protected from being accessed by users who are not in the administrators' user group. In Section 5.1, we ran a webserver to run the application. One of the main steps was to create a superuser. Superuser[15] is the first application user. It does have administrator access, so we may use it to access the administrator panel.

### 5.2.2 Main menu

The main menu is shared by every page with two exceptions: login page and Administration page, which is generated by the Django Admin application by registered data models[23].



Username: admin
e-mail: admin@lingit.com

**Figure 5.1.** Main menu

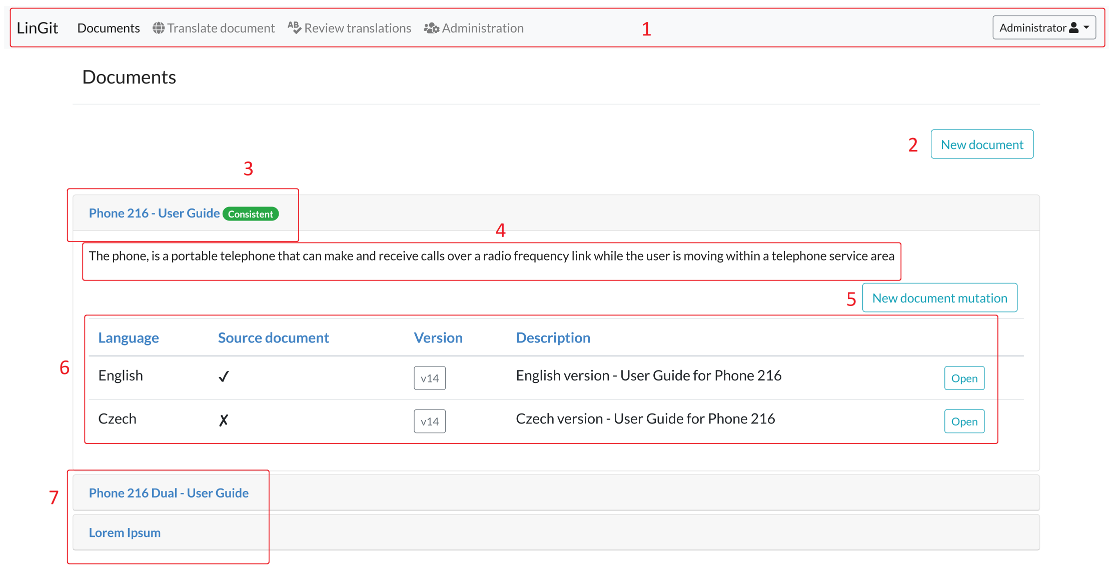
As we may see in Figure 5.1, the menu comprises the application logo, navigation panel, and user menu. The navigation panel is dynamic. Depending on the user group, elements are filtered to show relevant application parts. Default menu items visibility is shown in Table 5.1.

| Page | Translator | Moderator | Super Admin |
|---|---|---|---|
| Documents | No | Yes | Yes |
| Translate document | Yes | No | Yes |
| Review translations | No | Yes | Yes |
| Administration | No | No | Yes |
| Change password | Yes | Yes | Yes |

**Table 5.1.** Page visibility for different user groups

### 5.2.3 Document list

This page (in Figure 5.2) is the main application page. Here we may find all documents and their mutations, which are stored in the application.



The phone, is a portable telephone that can make and receive calls over a radio frequency link while the user is moving within a telephone service area

| Language | Source document | Version | Description | |
|---|---|---|---|---|
| English | ✓ | v14 | English version - User Guide for Phone 216 | Open |
| Czech | ✗ | v14 | Czech version - User Guide for Phone 216 | Open |

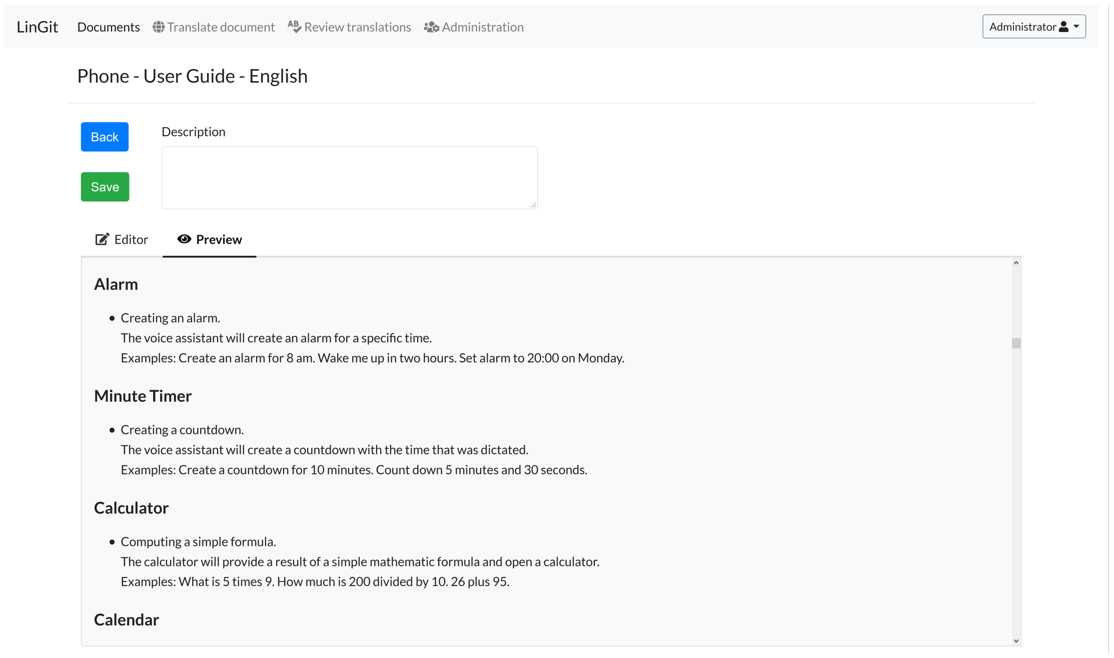
**Figure 5.2.** Documents list page

Let us describe every marked zone of the document list page:

1. Main menu
2. New document button redirects a user to the page where it is possible to create a new document. After the document is created, it will appear on the document list page.
3. Document header and document consistency tag. Headers of all documents are clickable. For example, the user guide for the `Phone 216` is expanded, but after the header is clicked, it will be in the compact form, same as `Phone 216 Dual` and `Lorem Ipsum` documents. The tag shows if the document is in a consistent state. If the tag is hidden, some document mutations have older versions that still need to be translated. As we may see on the highlighted areas #3 and #7 in Figure 5.2, the first document has a consistency tag because all its mutations have the same `v14` version.
4. Document description
5. New document mutation button redirects a user to the page for creation of a new document mutation, e.g., to create German mutation of the `Phone 216 - User Guide` document.
6. Document mutations table consists of 5 columns. The first column is the mutation language. The second column shows whether a mutation is a source document, e.g., changes from the English version will be translated to other `Phone 216 - User Guide` documents because it is a source mutation. The version column shows the current document mutation version. There is a detailed document mutation change list under the version tag described in detail in Subsection 5.2.5.
7. Collapsed documents

All tables on the page are generated by the `django-tables2` library. More information might be found in [24].

### 5.2.4 Edit mutation



Phone - User Guide - English

Alarm

- Creating an alarm.
  The voice assistant will create an alarm for a specific time.
  Examples: Create an alarm for 8 am. Wake me up in two hours. Set alarm to 20:00 on Monday.

Minute Timer

- Creating a countdown.
  The voice assistant will create a countdown with the time that was dictated.
  Examples: Create a countdown for 10 minutes. Count down 5 minutes and 30 seconds.

Calculator

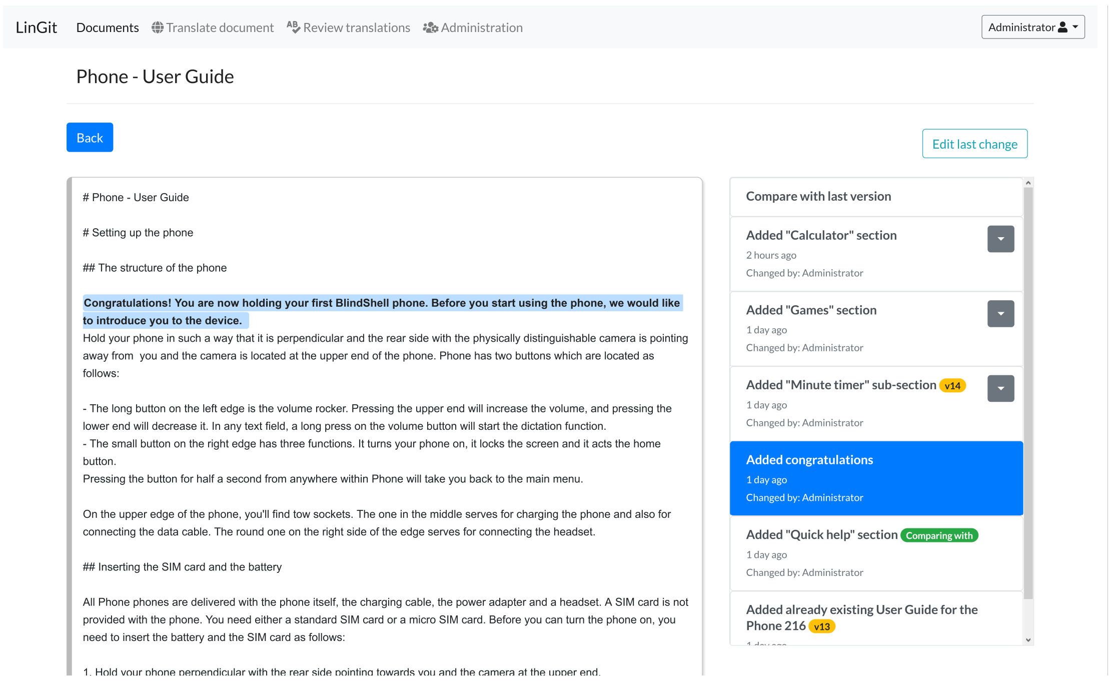
- Computing a simple formula.
  The calculator will provide a result of a simple mathematic formula and open a calculator.
  Examples: What is 5 times 9. How much is 200 divided by 10. 26 plus 95.

Calendar

**Figure 5.3.** Mutation change page

This page is one of the most used pages in the application. It designed to do changes to the source mutation (e.g., English), but may be used to make some changes into non-source document mutations if needed. In Figure 5.3, we may see that the page header with a document name combined with its language. Below them, there is a commit description window and Markdown editor in preview mode.

### 5.2.5 Mutation changes list page



Phone - User Guide

# Phone - User Guide

# Setting up the phone

## The structure of the phone

**Congratulations! You are now holding your first BlindShell phone. Before you start using the phone, we would like to introduce you to the device.**
Hold your phone in such a way that it is perpendicular and the rear side with the physically distinguishable camera is pointing away from you and the camera is located at the upper end of the phone. Phone has two buttons which are located as follows:

- The long button on the left edge is the volume rocker. Pressing the upper end will increase the volume, and pressing the lower end will decrease it. In any text field, a long press on the volume button will start the dictation function.
- The small button on the right edge has three functions. It turns your phone on, it locks the screen and it acts the home button.
Pressing the button for half a second from anywhere within Phone will take you back to the main menu.

On the upper edge of the phone, you'll find tow sockets. The one in the middle serves for charging the phone and also for connecting the data cable. The round one on the right side of the edge serves for connecting the headset.

## Inserting the SIM card and the battery

All Phone phones are delivered with the phone itself, the charging cable, the power adapter and a headset. A SIM card is not provided with the phone. You need either a standard SIM card or a micro SIM card. Before you can turn the phone on, you need to insert the battery and the SIM card as follows:

1. Hold your phone perpendicular with the rear side pointing towards you and the camera at the upper end.

Compare with last version

Added "Calculator" section
2 hours ago
Changed by: Administrator

Added "Games" section
1 day ago
Changed by: Administrator

Added "Minute timer" sub-section v14
1 day ago
Changed by: Administrator

Added congratulations
1 day ago
Changed by: Administrator

Added "Quick help" section Comparing with
1 day ago
Changed by: Administrator

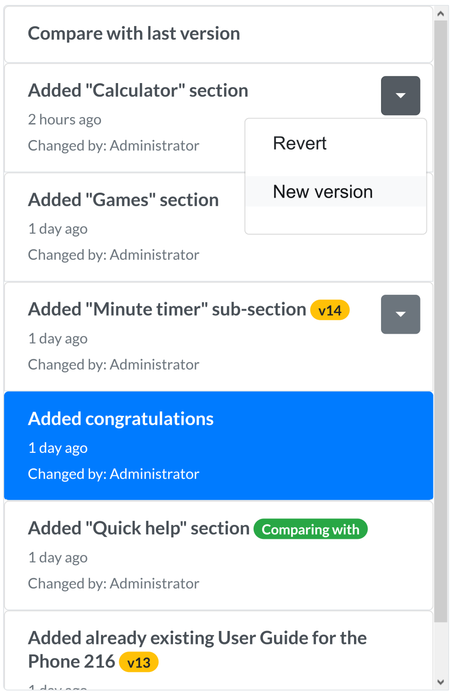
Added already existing User Guide for the Phone 216 v13

**Figure 5.4.** Interface of Mutation changes page

In Figure 5.4 we may see the page with the mutation changelog. On the left side, there is WikEd editor (described in Section 4.3) with the new text, which is colored in blue. On the right side, there is a button `Edit last change` and the list of changes, which are called `commits`.

The change list consists of a commits list and the special option «Compare with last version». When this special option is active, the application compares the most actual commit of document mutation with the last commit, which has a version tag.

The button «Edit last change» allows a user to change the top commit without creating a new one.



Compare with last version

Added "Calculator" section
2 hours ago
Changed by: Administrator

Revert
New version

Added "Games" section
1 day ago
Changed by: Administrator

Added "Minute timer" sub-section v14
1 day ago
Changed by: Administrator

Added congratulations
1 day ago
Changed by: Administrator

Added "Quick help" section Comparing with
1 day ago
Changed by: Administrator

Added already existing User Guide for the Phone 216 v13

**Figure 5.5.** Mutation change list

Every commit from the change list (could be seen in more detail in Figure 5.5) contains a commit description when the commit was created (e.g., 2 hours ago, 1 day ago) and the commit author. Some commits may have special tags and a menu button.

Commits might have two types of tags: version tag and «Comparing with» tag. As we may see, in the Figure 5.5 there are only two versions tags (yellow ones) «v13» and «v14», from that we may infer that the version «v14» contains this tree commits:

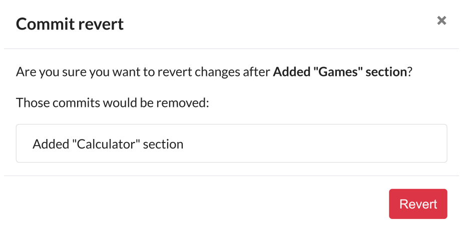
- Added ”Minute timer” sub-section
- Added congratulations
- Added ”Quick help” section

Version tag «v13» on the last commit means that it is the end of the 13th version.

Commit with «Comparing with» tag shows the commit with which the currently chosen commit is compared.

The menu is available for commits without version, that is version under development. There are two options here: Revert and New version.

The Revert option helps to remove commits that are newer than the chosen one. As we may see in Figure 5.6, where we want to revert all changes to the second commit from Figure 5.5.



Commit revert ×

Are you sure you want to revert changes after **Added "Games" section**?

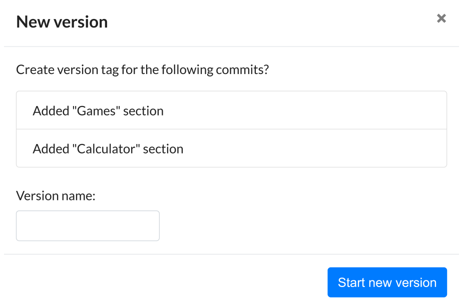
Those commits would be removed:

Added "Calculator" section

Revert

**Figure 5.6.** Commit revert window

The logic of adding a new version is opposite to the Revert option. On the chosen (e.g., Added "Calculator" section from Figure 5.5) commit, we will add a new version tag, meaning that the new version will end on the chosen commit. Result of choosing New version option is in Figure 5.7. In order to create a new version, you need to fill out the "Version name" field, e.g., v13.


New version

Create version tag for the following commits?

Added "Games" section

Added "Calculator" section

Version name:

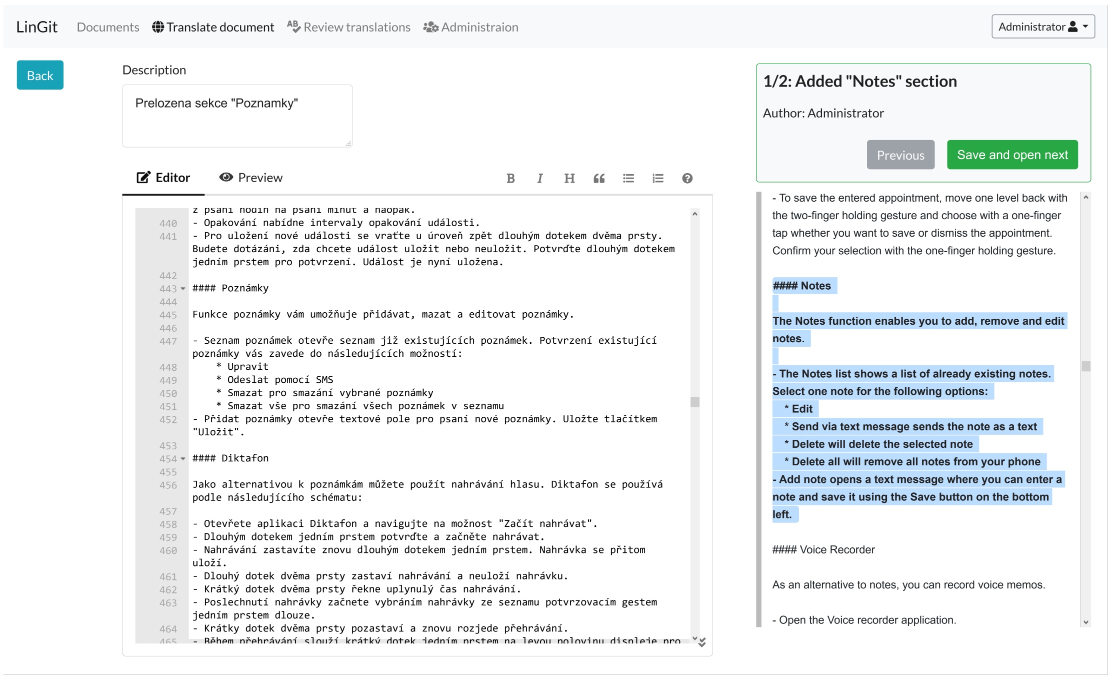
**Figure 5.7.** New version window

### 5.2.6 Translations page

This page is created to be used by translators. It consists of three main parts:

- Text editor which is described in Section 4.2
- Navigation window
- Commit to translate

The whole page overview may be found in Figure 5.8.



Description

Prelozena sekce "Poznamky"

```
z psaní hodin na psaní minut a naopak.
- Opakování nabídne intervaly opakování události.
- Pro uložení nové události se vraťte u úroveň zpět dlouhým dotekem dvěma prsty.
Budete dotázáni, zda chcete událost uložit nebo neuložit. Potvrďte dlouhým dotekem
jedním prstem pro potvrzení. Událost je nyní uložena.

#### Poznámky

Funkce poznámky vám umožňuje přidávat, mazat a editovat poznámky.

- Seznam poznámek otevře seznam již existujících poznámek. Potvrzení existující
poznámky vás zavede do následujících možností:
    * Upravit
    * Odeslat pomocí SMS
    * Smazat pro smazání vybrané poznámky
    * Smazat vše pro smazání všech poznámek v seznamu
- Přidat poznámky otevře textové pole pro psaní nové poznámky. Uložte tlačítkem
"Uložit".

#### Diktafon

Jako alternativou k poznámkám můžete použít nahrávání hlasu. Diktafon se používá
podle následujícího schématu:

- Otevřete aplikaci Diktafon a navigujte na možnost "Začít nahrávat".
- Dlouhým dotekem jedním prstem potvrďte a začněte nahrávat.
- Nahrávání zastavíte znovu dlouhým dotekem jedním prstem. Nahrávka se přitom
uloží.
- Dlouhý dotek dvěma prsty zastaví nahrávání a neuloží nahrávku.
- Krátký dotek dvěma prsty řekne uplynulý čas nahrávání.
- Poslechnutí nahrávky začnete vybráním nahrávky ze seznamu potvrzovacím gestem
jedním prstem dlouze.
- Krátky dotek dvěma prsty pozastaví a znovu rozjede přehrávání.
- Během přehrávání slouží krátký dotek jedním prstem na levou polovinu displeje pro
```

1/2: Added "Notes" section

Author: Administrator

```
- To save the entered appointment, move one level back with
the two-finger holding gesture and choose with a one-finger
tap whether you want to save or dismiss the appointment.
Confirm your selection with the one-finger holding gesture.

#### Notes

The Notes function enables you to add, remove and edit
notes.

- The Notes list shows a list of already existing notes.
Select one note for the following options:
   * Edit
   * Send via text message sends the note as a text
   * Delete will delete the selected note
   * Delete all will remove all notes from your phone
- Add note opens a text message where you can enter a
note and save it using the Save button on the bottom
left.

#### Voice Recorder

As an alternative to notes, you can record voice memos.

- Open the Voice recorder application.
```

**Figure 5.8.** Translations page - Overview

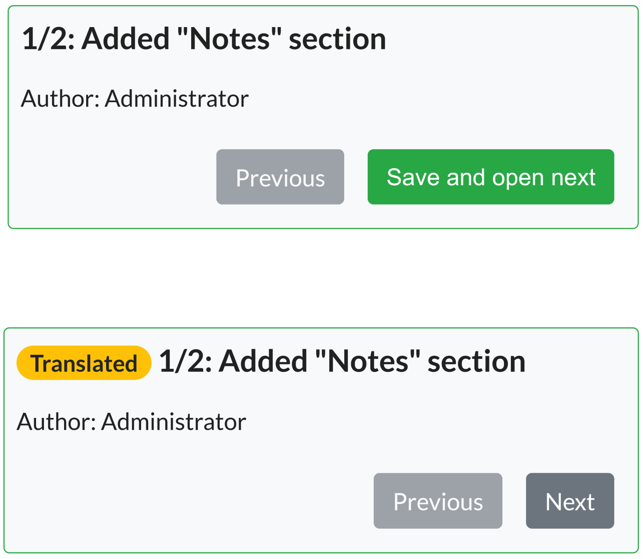


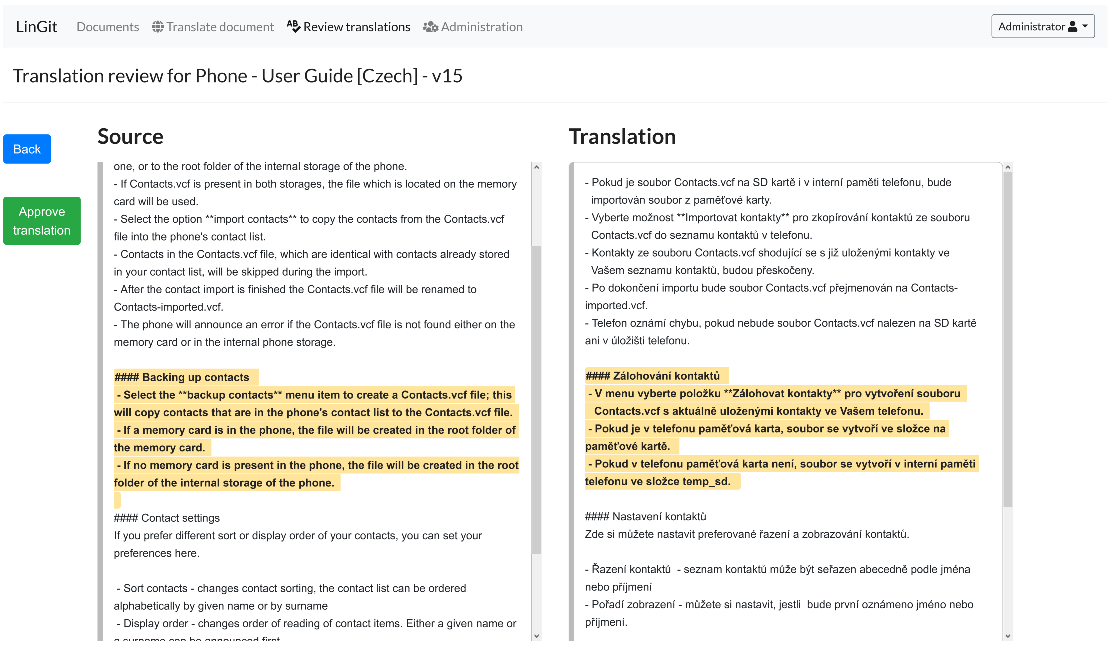
**Figure 5.9.** Navigation window - commit before and after translation

In Figure 5.9, we may see the navigation window in more detail. There are two images; the first one shows how the navigation window looks before translation, and the second one is after the translation was done.

The navigation window consists of four parts:

1. On the left upper corner commit number and overall commits count to translate are displayed. In Figure 5.9, the first commit out of two is opened. After the commit is translated, near the commit number appears a yellow `Translated` tag.
2. The next part is a commit description required when any document mutation is changed (more in Subsection 5.2.4). This description might help a translator understand what change was done in the main mutation and react accordingly.
3. The author section shows the name of the person who created the currently opened commit. In case the translator has some questions about the current change, he might want to contact this person.
4. The last part is the button section, where you may navigate between commits to translate. The `Previous` button is always presented but may be disabled when the first commit is open. The second button for non-translated commit saves changes. This button appears like the `Next` button for an already translated commit, which moves forward to the next commit.

### 5.2.7 Translation review page



Translation review for Phone - User Guide [Czech] - v15

Source

```
one, or to the root folder of the internal storage of the phone.
- If Contacts.vcf is present in both storages, the file which is located on the memory
card will be used.
- Select the option **import contacts** to copy the contacts from the Contacts.vcf
file into the phone's contact list.
- Contacts in the Contacts.vcf file, which are identical with contacts already stored
in your contact list, will be skipped during the import.
- After the contact import is finished the Contacts.vcf file will be renamed to
Contacts-imported.vcf.
- The phone will announce an error if the Contacts.vcf file is not found either on the
memory card or in the internal phone storage.

#### Backing up contacts
 - Select the **backup contacts** menu item to create a Contacts.vcf file; this
will copy contacts that are in the phone's contact list to the Contacts.vcf file.
 - If a memory card is in the phone, the file will be created in the root folder of
the memory card.
 - If no memory card is present in the phone, the file will be created in the root
folder of the internal storage of the phone.

#### Contact settings
If you prefer different sort or display order of your contacts, you can set your
preferences here.

 - Sort contacts - changes contact sorting, the contact list can be ordered
alphabetically by given name or by surname
 - Display order - changes order of reading of contact items. Either a given name or
```

Translation

```
- Pokud je soubor Contacts.vcf na SD kartě i v interní paměti telefonu, bude
  importován soubor z paměťové karty.
- Vyberte možnost **Importovat kontakty** pro zkopírování kontaktů ze souboru
  Contacts.vcf do seznamu kontaktů v telefonu.
- Kontakty ze souboru Contacts.vcf shodující se s již uloženými kontakty ve
  Vašem seznamu kontaktů, budou přeskočeny.
- Po dokončení importu bude soubor Contacts.vcf přejmenován na Contacts-
imported.vcf.
- Telefon oznámí chybu, pokud nebude soubor Contacts.vcf nalezen na SD kartě
ani v úložišti telefonu.

#### Zálohování kontaktů
 - V menu vyberte položku **Zálohovat kontakty** pro vytvoření souboru
  Contacts.vcf s aktuálně uloženými kontakty ve Vašem telefonu.
 - Pokud je v telefonu paměťová karta, soubor se vytvoří ve složce na
paměťové kartě.
 - Pokud v telefonu paměťová karta není, soubor se vytvoří v interní paměti
telefonu ve složce temp_sd.

#### Nastavení kontaktů
Zde si můžete nastavit preferované řazení a zobrazování kontaktů.

- Řazení kontaktů  - seznam kontaktů může být seřazen abecedně podle jména
nebo příjmení
- Pořadí zobrazení - můžete si nastavit, jestli  bude první oznámeno jméno nebo
příjmení.
```

**Figure 5.10.** Translation review page

Translation review page title consists of four parts corresponding to following pattern: Translation review for «document-name» [«translated-document-language»] - «reached-version».

The page is intended to review translation changes. On the left side, there is a text with changes from the source mutation version which was translated. On the right side of the window, there is a translation itself.

After the "Approve translation" button is clicked, a non-source document mutation receives the desired version tag.

# Chapter 6
# Conclusion

In this thesis, we developed the version control system for multilingual documents. It provides a whole workflow for the companies which need to employ multilingual documents in their work. This application covers all requirements to make translations consistent. It generates a list of changes that they must finish to align document versions for translators. It provides a review tool for moderators, where it is possible to see all changes for the translated version. Our application supports multiple constantly changing multilingual documents, providing a whole system where all the documents are stored and changed. This way, it covers the whole translations' workflow, so no other tool is needed.

Our application's main advantage is a VCS-like structure of changes, where instead of one huge change is used commits model to divide work into atomic pieces naturally.

For such a system, there is always a lot of possible changes. More information about planned changes might be found in Section 6.1.

## 6.1 Future work

In this Section we will describe all planned changes to be done in the LinGit system. In Subsection 6.1.1, we introduce the task generation, where multiple commits might be joined. In Section 6.1.2, we show unchanged text hiding to display changes more clearly. In Section 6.1.3, we introduce the Cypress framework, which should provide a test environment for our application's frontend.

### 6.1.1 Commits joining, task generation

In this Subsection, we will discuss the task generation. The idea behind task generation is when the source-mutation is changed rapidly, some commits are changing the text nearly in the same place. This means that we may join such commits.

Comparing Figures 3.1 and 6.1 we see, that task generation potentially reduces number of translation commits. It will probably be reflected in commit sizes but should not interfere with the translator's work because the changes from one task should be near one place in the text.

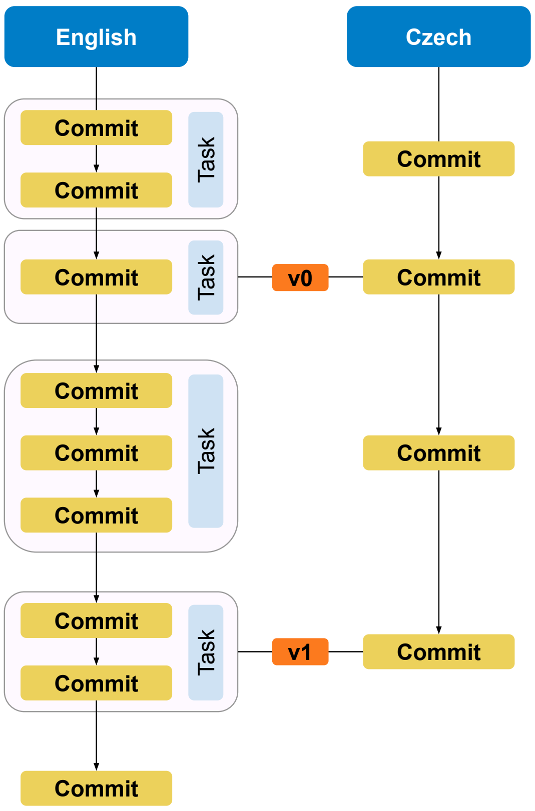
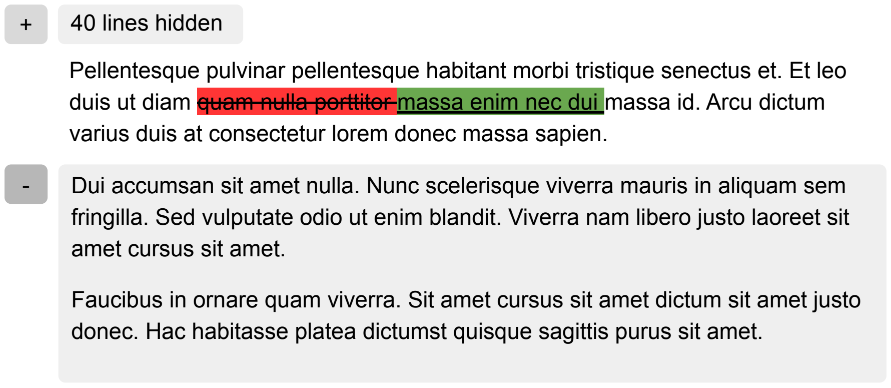


**Figure 6.1.** Tasks model

### 6.1.2 Unchanged text hiding

One of the possible improvements is to hide parts of the text without changes. This change will help to work with large texts where there is a small change only. It is immediately apparent how many text blocks were changed.



+ 40 lines hidden

Pellentesque pulvinar pellentesque habitant morbi tristique senectus et. Et leo duis ut diam ~~quam nulla porttitor~~ massa enim nec dui massa id. Arcu dictum varius duis at consectetur lorem donec massa sapien.

- Dui accumsan sit amet nulla. Nunc scelerisque viverra mauris in aliquam sem fringilla. Sed vulputate odio ut enim blandit. Viverra nam libero justo laoreet sit amet cursus sit amet.

Faucibus in ornare quam viverra. Sit amet cursus sit amet dictum sit amet justo donec. Hac habitasse platea dictumst quisque sagittis purus sit amet.

**Figure 6.2.** Change collapse

As we may see in Figure 6.2, there are tree text parts. The first one is hidden and consists of 40 lines of text. There is a plus sign button on the left side of the text block, which expands the hidden text block. The second is a difference between the texts themselves. The last block of text is another unchanged piece of text. The button's text is changed on the minus sign, which means that this button may hide unchanged text. By default, all unchanged text blocks must be hidden.

It is possible to use HTML5 “details-summary” tags pair, which creates a small title which hides larger piece of text:

```
<details>
    <summary>TEXT SUMMARY</summary>
    <p>TEXT TO HIDE</p>
</details>
```

### 6.1.3 Frontend testing

In Section 4.5, we described how the backend application part is tested. The LinGit frontend is not tested at all, and that is a problem during development because it is easy to break something up without noticing it.

One of the possibilities to test the frontend side of the application is to use the Cypress, which is a JavaScript testing framework. It is independent of the web application development framework. This means that there is no difference if the application’s frontend framework will be changed to another, such as React, Angular, or Vue.js. Cypress tests will remain the same.

# References

[1] Pariyar, A., D. Lin, and T. Ishida. Tracking Inconsistencies in Parallel Multilingual Documents. In: *2013 International Conference on Culture and Computing*. 2013. pp. 15-20.

[2] Ruparelia, Nayan B. The History of Version Control. *SIGSOFT Softw. Eng. Notes*. New York, NY, USA: Association for Computing Machinery, jan, 2010, Vol. 35, No. 1, pp. 5–9. ISSN 0163-5948. Available from DOI 10.1145/1668862.1668876.
`https://doi.org/10.1145/1668862.1668876`.

[3] Somers, Harold. Translation memory systems. *Benjamins Translation Library*. JOHN BENJAMINS BV, 2003, Vol. 35, pp. 31–48.

[4] Lagoudaki, Elina. Translation memories survey 2006: Users' perceptions around TM use. *Proceedings of the ASLIB International Conference Translating & the Computer*. London: Imperial College London, 2006, Vol. 28, pp. 12–13.

[5] Xia, Ying. Research on statistical machine translation model based on deep neural network. *Computing*. Springer, 2020, Vol. 102, No. 3, pp. 643–661.

[6] Kituku, Benson, Lawrence Muchemi, and Wanjiku Nganga. A Review on Machine Translation Approaches. *Indonesian Journal of Electrical Engineering and Computer Science*. 01, 2016, Vol. 1, pp. 182. Available from DOI 10.11591/ijeecs.v1.i1.pp182-190.

[7] Google. *Terms of Service* [Online]. [cit. 2020-12-25].
`https://policies.google.com/terms?hl=en-US#toc-what-you-expect`.

[8] Lakritz, David. *Translation management system*. US Patent 8,489,980.

[9] Schmid Stevenson, Sarah. *Pairaphrase Pushes Smart, Secure Language Translation Software* [Online]. [cit. 2020-12-25].
`https://xconomy.com/detroit-ann-arbor/2016/07/07/pairaphrase/`.

[10] Chawathe, Sudarshan S., and Hector Garcia-Molina. Meaningful Change Detection in Structured Data. *SIGMOD Rec.* New York, NY, USA: Association for Computing Machinery, jun, 1997, Vol. 26, No. 2, pp. 26–37. ISSN 0163-5808. Available from DOI 10.1145/253262.253266.
`https://doi.org/10.1145/253262.253266`.

[11] *SQLite Is Serverless* [Online]. [cit. 2020-12-26].
`https://www.sqlite.org/serverless.html`.

[12] *Write-Ahead Logging* [Online]. [cit. 2021-1-2].
`https://sqlite.org/wal.html`.

[13] *Developer Survey Results - 2019* [Online]. [cit. 2020-12-19].
`https://insights.stackoverflow.com/survey/2019#technology-_-web-frameworks`.

[14] Rossum, Guido van, and Python Development Team. *Python Tutorial: Release 3.6.4*. 2018. ISBN 1680921606.

[15] Django Software Foundation. *Django documentation* [Online]. [cit. 2020-08-05].
https://docs.djangoproject.com/en/3.1/.

[16] agusmakmun. *Django Markdown Editor* [Online]. [cit. 2020-08-04].
https://github.com/agusmakmun/django-markdown-editor.

[17] Cacycle. *WikEd diff* [Online]. [cit. 2020-08-04].
https://en.wikipedia.org/wiki/User:Cacycle/diff.

[18] Wikimedia Foundation. *List of Wikipedias* [Online]. [cit. 2020-08-03].
https://en.wikipedia.org/wiki/List_of_Wikipedias#Grand_total.

[19] The Python Markdown Project. *A Python implementation of John Gruber's Markdown.* [Online]. [cit. 2020-12-13].
https://github.com/Python-Markdown/markdown.

[20] The Python Markdown Project. *Writing Extensions for Python-Markdown* [Online]. [cit. 2020-11-29].
https://python-markdown.github.io/extensions/api/.

[21] Fraser, Neil. *Diff Match Patch* [Online]. [cit. 2020-12-16].
https://opensource.google/projects/diff-match-patch.

[22] Python Software Foundation. *Python documentation. Unit testing framework* [Online]. [cit. 2020-12-19].
https://docs.python.org/3/library/unittest.html.

[23] Django Software Foundation. *Documentation. The Django admin site* [Online]. [cit. 2020-12-13].
https://docs.djangoproject.com/en/3.1/ref/contrib/admin/.

[24] Waagmeester, Jan Pieter. *Django-tables2 - An app for creating HTML tables* [Online]. [cit. 2020-12-28].
https://django-tables2.readthedocs.io/en/latest/index.html.

# Appendix A
# List of abbreviations

| | | |
|---|---|---|
| API | ■ | Application Programming Interface |
| CRUD | ■ | Create, Read, Update and Delete operations |
| EBMT | ■ | Example-based machine translation |
| HTML | ■ | Hypertext Markup Language |
| MT | ■ | Machine translation |
| ORM | ■ | Object-Relational Mapping |
| PBKDF2 | ■ | Password-Based Key Derivation Function 2 |
| RBMT | ■ | Rule-based machine translation |
| REST | ■ | Representational State Transfer |
| SBMT | ■ | Statistical-based machine translation |
| SQL | ■ | Structured Query Language |
| TMS | ■ | Translation Management Systems |
| VCS | ■ | Version Control Systems |
| WAL | ■ | Write-Ahead Log |

# Appendix B
# Project files

| | |
|---|---|
| `vcs-for-translations.pdf` | Contains this bachelor’s thesis in PDF format. |
| `lingit` | Folder with the LinGit application files. |