Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

GIT based markdown online editor

Bc. Vojtěch Sajdl

Supervisor: Ing. Martin Komárek
Field of study: Open Informatics
Subfield: Software Engineering
May 2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Sajdl** Jméno: **Vojtěch** Osobní číslo: **457089**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Online Markdown editor postavený na GIT

Název diplomové práce anglicky:

GIT Based Markdown Online Editor

Pokyny pro vypracování:

Vytvořte online nástroj, který bude podporovat celý životní cyklus tvorby dokumentace software pomocí značkovacího jazyka Markdown a verzovacího nástroje GIT. Nástroj by měl být použitelný zejména pro práci se systémem Docusaurus.
Nástroj musí zejména zohledňovat:
- potřeby různých zúčastněných rolí (autor, korektor, copywriter,...)
- potřebu tvorby dokumentace pro různé verze dokumentovaného systému
Aplikace by měla umožnit zobrazení změněných souborů, zvýraznit jejich změněné části a umožnit soubory editovat. Klaďte důraz na jednoduchost použití, proveďte také kvalitativní uživatelské testování. Vývoj provádějte agilním způsobem.

Seznam doporučené literatury:

1. Facebook Inc.: Docusaurus, https://v2.docusaurus.io/docs/
2. John Gruber: Markdown: Syntax, https://daringfireball.net/projects/markdown/syntax
3. CommonMark: Commonmark: A strongly defined, highly compatiblespecification of Markdown, https://commonmark.org/
4. Scott Chacon a Ben Straub: Pro Git, ISBN: 1484200772
5. Yakov Fain a Anton Moiseev: Exploring Modern Web Development, ISBN: 9781617297809

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Martin Komárek, katedra informační bezpečnosti FIT

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

Datum zadání diplomové práce: **10.02.2021** Termín odevzdání diplomové práce: **21.05.2021**

Platnost zadání diplomové práce: **30.09.2022**

Ing. Martin Komárek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

.	
Datum převzetí zadání	Podpis studenta

Acknowledgements

I would like to thank my supervisor, Ing. Martin Komárek, for providing guidance and feedback throughout this thesis.

I would also like to express my gratitude to Justin Dalrymple for his help solving the mime type issue in Gitbeaker.

And finally, I would also like to greatly thank my girlfriend, family and friends who have all been very supportive.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

In Prague on May 21, 2021

Abstract

This thesis focuses on creating a comprehensive online tool that supports the entire life cycle of software documentation creation, using GIT and Markdown at its core. It aims to satisfy the needs of multiple user groups, especially authors, proof-readers and documentation managers. The system should also support creation of documentations for different versions of the documented system and most importantly be able to highlight changed parts of the text.

The developed system was aimed to be integrated with the existing documentation process used by Stratox Enterprises s.r.o. - the main source of information for this purpose was the thesis supervisor. The system was to be integrated with the technology that was already used - this means using the Gitlab repository hosting service and Docusaurus - the static page generator. Also, the system was to be deployed on the in-house developed CodeNow platform.

As Docusaurus supports all the requirements imposed upon the resulting documentation, the focus was to create a tool, that would allow for easy file edit, simplified proof-reading process and management of the documentations. After prototyping and finishing the minimal viable product, user testing was conducted. Found problems and their solutions are also discussed.

Keywords: Markdown, GIT, Editor, Docusaurus, Documentation management

Supervisor: Ing. Martin Komárek

Abstrakt

Tato práce se zabývá tvorbou komplexního online nástroje pro podporu celého životního cyklu tvorby dokumentace za použití nástroje Git a jazyku Markdown. Jejím cílem je uspokojit potřeby různých uživatelských skupin, zejména pak autorů, korektorů a manažerů dokumentací. Výsledný systém by měl podporovat tvorbu dokumentací pro různé verze dokumentovaného systému a funkci zvýrazňování změn souborů.

Vyvíjený systém byl určen pro integraci do stávajícího procesu dokumentace používaného firmou Stratox Enterprises s.r.o. - pro získávání informací o tomto procesu a použitých technologiích byl kontaktní osobou vedoucí práce. Systém bylo třeba integrovat s používanými technologiemi – zejména pak s hostingem GIT repositářů Gitlab a generátorem statických stránek Docusaurus. Vyvíjený systém měl být také nasazen na jejich vlastní platformě CodeNow.

Jelikož Docusaurus podporuje všechny požadavky na výslednou podobu dokumentace, práce se zabývala zejména vývojem nástroje, který by umožnil snadnou editaci, zjednodušený proces korektury a správy dokumentací. Po dokončení prototypování a vývoje minimálního použitelného produktu bylo provedeno uživatelské testování. Jeho výsledky – zejména pak nalezené problémy a jejich řešení jsou následně v práci diskutovány.

Klíčová slova: Markdown, GIT, Editor, Docusaurus, Documentation management

Překlad názvu: Online Markdown editor postavený na GIT

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

Software documentation is a thing that every programmer uses almost every day. Yet, this part of the job is often neglected, as it is often seen as a burden than a co-created artifact [1]. This is mainly true for internal code documentation, because more often than not, companies take a great care of user-facing documentations. This is mainly because these can have the power to decide whether their platform will be used or not. User-facing documentations can include architecture documentation, technical documentation and end user documentation [2]. These can take shape of API endpoint documentation to enable companies to integrate the software into their own apps or even tutorials for the end user on how to use their solution. It is these types of documentation that this thesis focuses on.

Nowadays there are many solutions for generating documentation websites, many of which are open source. That makes it easy for companies to cut costs of trying to develop their own documentation system. But while these tools work great for technically skilled users, there are still many other people included in the documentation process. Some of them might not have the technical knowledge to use the technologies these tools often utilize. Although Markdown aims to have a simple syntax that should be easy to read and write [3], especially older users might not be able to use it. And with GIT, which is often used to track the documentation changes, can be hard even for experienced users. This is something we need to keep in mind as many companies employ proof-readers and translators who might not have any prior experience with these technologies.

1.2 Goals

Because of the reasons mentioned above, the current process required the submitter to create a document from the markdown page they wanted to proofread. This document would then need to be sent to the proof-reader, who would make corrections and return the modified document back to the submitter. Submitter would then need to apply these changes manually and add them to the GIT repository. This makes for a long process when many pages are to be proofread.

Therefore, the focus of this thesis is to design and implement a system, that would solve these issues. The system should be a web application compatible with all modern web browsers. It should allow for the most common use cases such as editing files and requesting proofreading, without the burden of the previous processes. All changes should be automatically saved to the corresponding GIT repository and should not require user intervention unless absolutely necessary.

1.3 Thesis structure

The structure of this thesis is as follows:

- In this chapter we outlined what problem are we trying to solve and why.
- Chapter 2 contains a more in-depth look at the requirements and use cases of the application and researches the current state of technology.
- Chapter 3 shows the application design decisions and the reasoning behind them.
- Chapter 4 contains information about the finished application implementation.
- Chapter 5 shows the results of user testing and the consequent improvements along with how automatic testing is implemented.
- Chapter 6 contains the results of this thesis along with a list of improvements that can be made.

Chapter 2

Analysis

2.1 Requirements analysis

Throughout the entire thesis, the supervisor acted as the point of contact in regards of requirements gathering. Below you will find the requirements that were collected throughout working on the thesis.

2.1.1 Functional requirements

Login & permissions

The application should require users to log in and display only information the user should have access to. There should be at least 3 user levels - proof-reader, author and administrator. The administrator has full access to the documentation, while author can only edit the files contained in it or resolve proofreading requests. The proof-reader has access to own assigned proofreading requests.

Documentation management

The application should allow for a new documentation to be created, edited and deleted. Documentation administrator should be able to grant users access, so that more people can collaborate. Appropriate changes should be made in the hosted repository.

File diff

The application should allow for file differences to be displayed. This includes a list of changed files as well as highlighted changes in their content. These changes should be highlighted as green (in case of addition) or red (in case of deletion).

File edit

The application should allow file edit in such a manner that a WYSIWYG editor would be utilized. The changes should be written to the GIT repository on save and the changes should be autosaved so that in case of mistakenly closed tab the content could be recovered.

Proofreading

The application should facilitate the entire proofreading process. The process should start with selection of files that need to be proofread on a page with file differences displayed. The submitter should then create a proofreading request and assign a proof-reader to that request. Proof-reader will then edit the files and submit the finalized version. It is possible that the finalized version will contain no changes, the system should be able to handle such a situation. Then the submitter will either accept (merge) or reject the changes.

2.1.2 Non-functional requirements

File format support

The application should be able to work with the MD and MDX file formats. It does not need to support all features of the MDX format. If possible, the application should support displaying images hosted in the GIT repository or use placeholders otherwise.

Versioning system

The application should support the GIT versioning system. The GIT repositories will use one of the available hosting solutions such as GitHub or Gitlab.

Interface

The GUI should be available publicly from any web browser. The interface should also be responsive and usable on mobile devices.

Browser support

Only last two versions of browsers under active development need to be supported.

Language

The GUI language doesn't need to be switchable and should be primarily in English.

Deployment

The application is to be deployed using Docker images on the CodeNow platform.

2.2 Use cases & processes

In this section the two most important use cases are described more in-depth. This is done mainly to paint a better picture of the logic behind the process design. User roles and their use cases are described in the diagram below.

2.2.1 Use cases

When we consider only the app as a whole, there is just one user group. Each user can login, logout, create new documentation and view the documentations they have access to.

As the applications main purpose is documentation management, the use cases are documentation dependent. Therefore, without the user being assigned to a documentation or having created a documentation, the use of the app is limited. The use cases are illustrated in the diagram below.

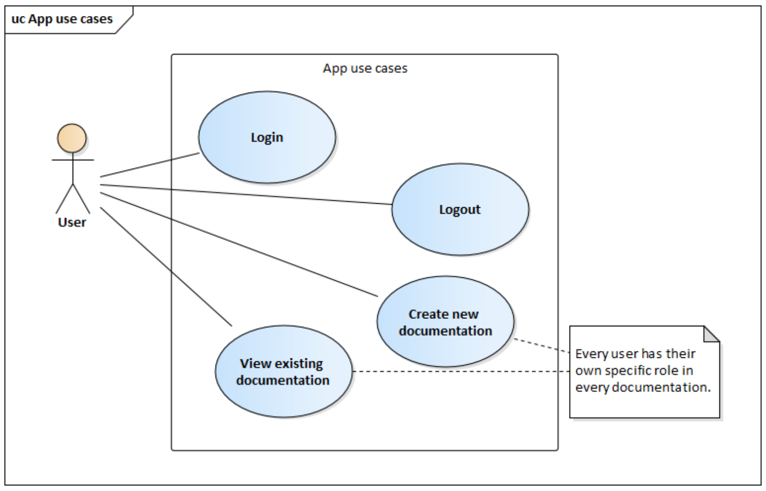

uc App use cases
App use cases
User
Login
Logout
Create new documentation
View existing documentation
Every user has their own specific role in every documentation.

Figure 2.1: App use cases

When we take a look at the use cases related to documentations, we can see 4 main user groups. The main idea between the split is, that the proof-reader should only be able to do proofreading related tasks, author should be able to to CRUD operations on any file, the documentation manager should be able to create proofreading requests and the admin should have full access to the documentation. You can see the details in figure 2.2.

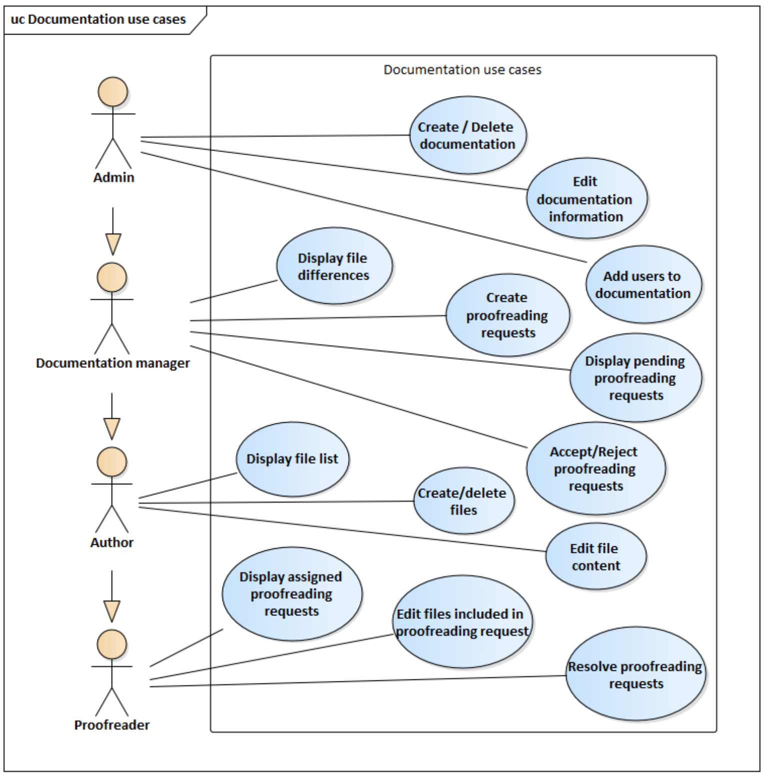

uc Documentation use cases
Documentation use cases
Admin
Documentation manager
Author
Proofreader
Create / Delete documentation
Edit documentation information
Add users to documentation
Display file differences
Create proofreading requests
Display pending proofreading requests
Accept/Reject proofreading requests
Display file list
Create/delete files
Edit file content
Display assigned proofreading requests
Edit files included in proofreading request
Resolve proofreading requests

Figure 2.2: Documentation use cases

2.2.2 Documentation management

The documentation management is one of the most important things to describe, mainly because of the underlying connection to the GIT. Usually, if we talk about documentation, we mean the representation in the application and if we talk about the repository, we mean the repository where the documentation files are stored.

The underlying repository is expected to be managed solely by the provided application, any user interaction with the underlying repository should not be required and is not recommended. This is because no synchronization mechanism will be implemented in this thesis. The application can create and delete the documentation, add and remove users, change their permissions or modify the title, slug or description of the documentation. All the required changes should be made to the repository automatically. If the user makes modifications to the repository, everything should still be able to work (unless the user does not have the required permissions), but the changes will not be reflected in the application.

User roles

Almost all the changes can be made in 1:1 manner apart from one: user permissions. The user should always get the minimal required access to the repository. To be able to push files into the repository, at least developer role is required [4]. This is needed even for our user role with the least permissions. Should further user access control be required, protected branches could be utilised.

Role	GitLab role	Gitlab permission summary
Admin	Maintainer	Has full control over the repository
Documentation manager	Developer	Can perform any action on unprotected branches
Author	Developer	Same as above
Proofreader	Developer	Same as above

Table 2.1: User role mapping

2.2.3 Proofreading process

The design of the proofreading process was important, as this was the main intended functionality of the application. The user should be able to select files they want included, and only those files should be visible to the proof-reader. It should be also possible to write a note to the proof-reader and either accept or reject the changes the proof-reader made. The proof-reader should be able to work on the request and submit it when the proofreading is finished. This led to the design of the simple process depicted below.

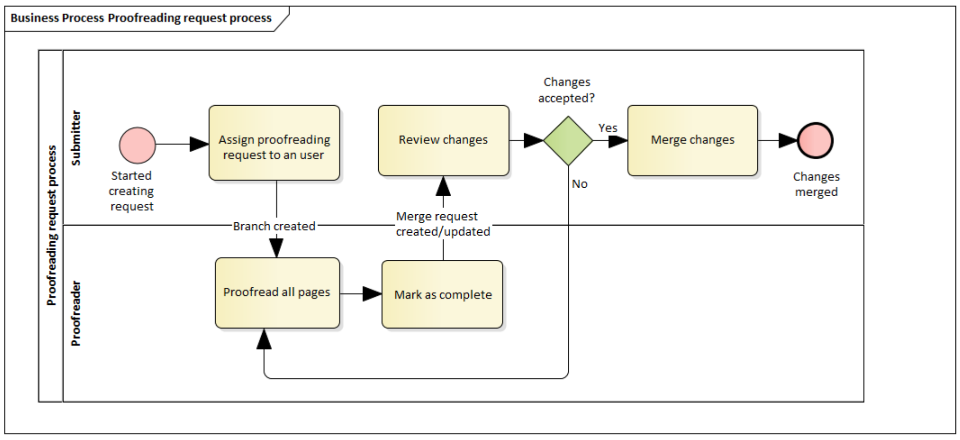


Figure 2.3: Proofreading process

2.3 Available technology

2.3.1 Technology used in current setup

Docusaurus

Docusaurus is Open-Source project made by Facebook that is often used to generate documentation. Lately the developers recommend using the alpha version of Docusaurus - Docusaurus 2. This version has been rebuilt from the ground up, so that it keeps the benefits of Docusaurus 1 (such as versioned documentation and i18n) while implementing many new features. Docusaurus 2 now can generate many common content-driven websites such as blogs, landing pages and many more [5]. The new version has also switched its page syntax from GitHub Flavoured Markdown (GFM) to MDX which brings a lot of new features.

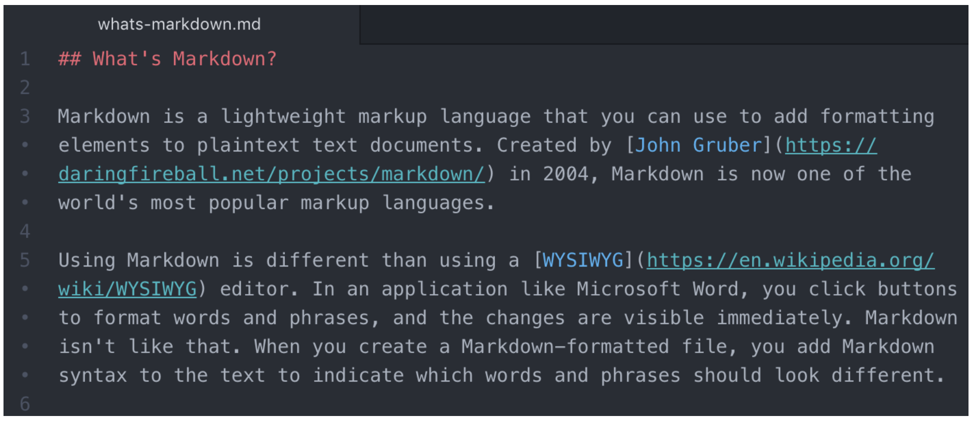
GitLab

As their GIT hosting service, the Stratox Enterprises s.r.o. uses a self-hosted GitLab instance. This is important because it means that we should support configuring the instance URL. Moreover, each hosting service has their own specific APIs and libraries. And even with the libraries provided, there are still many ways to implement such API access - this will be explained in the Chapter 3.

2.3.2 Markdown

Markdown history

The original Markdown specification was written by John Gruber in the year of 2004 [6]. The main advantage of Markdown is that its use does not require any WYSIWYG editor, because of its easy syntax. This syntax is designed in such a way, that it should be very easy to read even though the text is not rendered. For illustration, you can see a Markdown file example below.



whats-markdown.md

```
## What's Markdown?

Markdown is a lightweight markup language that you can use to add formatting
elements to plaintext text documents. Created by [John Gruber](https://
daringfireball.net/projects/markdown/) in 2004, Markdown is now one of the
world's most popular markup languages.

Using Markdown is different than using a [WYSIWYG](https://en.wikipedia.org/
wiki/WYSIWYG) editor. In an application like Microsoft Word, you click buttons
to format words and phrases, and the changes are visible immediately. Markdown
isn't like that. When you create a Markdown-formatted file, you add Markdown
syntax to the text to indicate which words and phrases should look different.
```

Figure 2.4: Markdown syntax example [6]

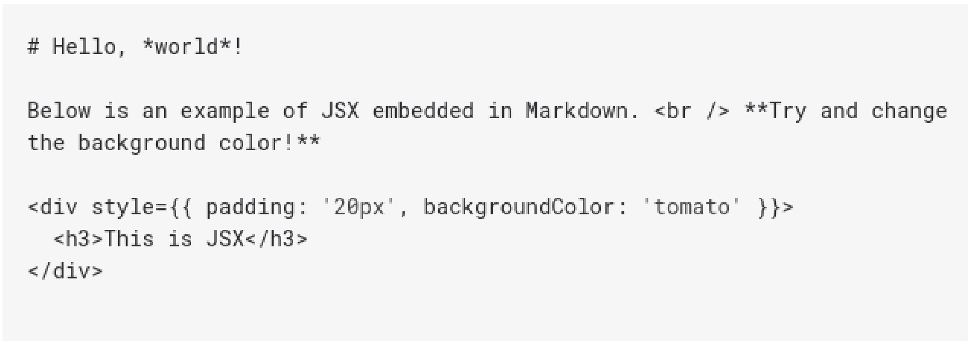
Thanks to its simplicity it spread and is nowadays used in many different applications. Along with Docusaurus, many popular services such as GitHub, Slack, Trello and even the Reddit social network have implemented it as their text markup syntax [7]. But the original specification was vague and that left room for different interpretations. As a result, if you take a GitHub Wiki page and run it through a different system (such as Pandoc), the output can be very different [8]. Moreover, the original Markdown specification does not specify any conditions that would lead to syntax error; therefore, these implementation differences might not be noticeable at the first sight.

To try to solve this issue, the CommonMark project was founded, creating a comprehensive test suite and an unambiguous Markdown specification [8]. The project even provides reference implementations of the specification in various languages. Therefore, CommonMark is the specification most flavours are based on. Why are there still flavours if CommonMark created an unambiguous Markdown specification?

There are still some problems that persist - mainly the absence of some common elements used in text processors - such as tables. But even though this is a problem, most flavours are mainly platform specific extensions. For example, GitHub has its own specification that supports referencing issues, pull requests, navigation inside wiki pages and even the beforementioned tables. Other flavours include GitLab Flavoured Markdown, StackOverflow's Markdown and many more [9]. Frontmatter is one of the most added features, mainly for services that generate websites using Markdown. It allows for specification of additional information including, but not limited to author, page title, page id and edit link.

Markdown vs MDX

Now that we know more about the history of Markdown, let us look at what MDX is trying to solve. Even though CommonMark is a new specification, the core concept remains unchanged - readability above all. MDX takes a slightly different approach. MDX sacrifices some of its readability, so that JSX code can be incorporated. An example of the syntax can be seen in figure 2.5. This allows users to create dynamic pages using JavaScript - more specifically React components. This can be anything from interactive tables, dynamic alerts or even charts [10]. MDX also provides not just the specification, but the toolkit to make generation of such interactive pages possible.



```
# Hello, *world*!

Below is an example of JSX embedded in Markdown. <br /> **Try and change
the background color!**

<div style={{ padding: '20px', backgroundColor: 'tomato' }}>
  <h3>This is JSX</h3>
</div>
```

Figure 2.5: MDX syntax example [10]

2.4 Alternative markdown parsers

Because in the application requirements MDX parity was not specifically requested, we would be able to use a different Markdown parser in case the MDX toolkit was hard to implement or slow. Below you will find a list of other well-known Markdown parsers.

2.4.1 Remark

Remark is the world's most popular Markdown parser. It can do much more than just parsing - it can inspect and transform the parsed Markdown too. Using plugins, it can do anything from linting the code to even transforming the code into valid React markup. It uses the micromark parser internally - this parser has 100% CommonMark compatibility, and it can be extended to support different flavors using plugins [11]. It is used by beforementioned MDX, Gatsby or Netlify CMS.

2.4.2 Marked

Marked is second most popular markdown parser. Its main advantage is that it is built with speed in mind. Although it is fast, it implements features from many common markdown specifications, although it does not aim for a strict compliance to any specification. It works in the browser as well as in the NodeJS environment. The extension system was designed with simplicity in mind and allows us to extend the renderer and tokenizer [12].

2.4.3 markdown-to-jsx

Markdown-to-jsx is an easy-to-use markdown component that takes GitHub flavoured Markdown (GFM) and makes native JSX without dangerous hacks. It uses a heavily modified fork of simple-markdown as its parsing engine [13]. This parser is notable thanks to its JSX support and seamless integration with React. Below you will find a popularity chart taken from npmtrends.

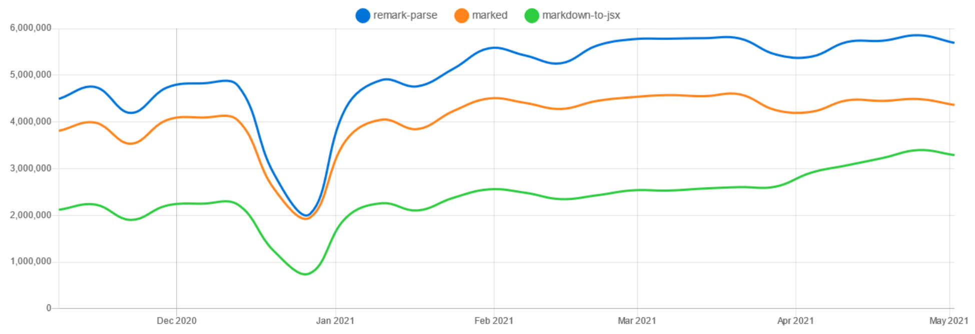


Figure 2.6: Parser popularity chart [14]

2.5 React

2.5.1 Javascript

Before we talk about React, it would be appropriate to make a quick introduction to JavaScript and some of its problems. JavaScript is an interpreted programming language mainly known for its use in web browsers, though it is also used in other environments - for example NodeJS [15]. As the browsers improve, the web technology specifications are bound to change. Most often these changes bring along new features, make the specifications less vague or deprecate some less known or experimental APIs.

In a perfect world, we would not have to care and could just use the latest and greatest features of the JavaScript language. But as it happens, some users are reluctant to change. Even to this date, Internet Explorer, for which Microsoft is releasing bug fixes and security patches only[1], has over 2.5% of desktop market share worldwide [16]. Looking at modern browsers, even Safari made by Apple lacks some of the features specified in ES6+ [17]. Therefore, many libraries have either limited browser support, or require polyfills to add it (as is the case with React) - this mainly applies to the beforementioned Internet Explorer.

2.5.2 React and JSX

React is a JavaScript library for building user interfaces across a variety of platforms [18]. It allows programmers to design simple views while React will efficiently update and render only the components that have changed. Declarative views make the code more predictable and easier to debug. React can be used on the server side using NodeJS, in the browser or it can even be used to build mobile applications using React Native [19].

And what is the relation between React and JSX? JSX looks similar to a templating language but is actually an extension of the JavaScript language, that was designed specifically for React. JSX is used in React to describe what the user interface should look like, but it is not the only way to write React. JSX embraces the fact that the rendering and other UI logic are inherently coupled together - including event handling, state changes or data preparation [20].

Instead of artificially separating the code by splitting files into UI and logic parts, React separates concerns with loosely coupled units called “components”

[1]Applies only to Internet Explorer 11, all other versions are unsupported.

that contain both. Although React doesn't require JSX to be used, many people find it useful, because they can see the user interface structure while working with the corresponding logic [20].

2.5.3 Comparison of React to other frameworks

When talking about React, usually two other big frameworks come up too. These are Angular and Vue.js. Each has their own benefits, which will be quickly summarized below.

	React	Angular	Vue
Watchers	6.7k	3.2k	6.3k
Stars	168k	73.1k	183k
Forks	33.8k	19.2k	29k
Contributors	1542	1407	399
Used by	6.4M	1.8M	145k

Table 2.2: Popularity of frontend frameworks based on data from GitHub on May 11, 2021 from the respective GIT repositories

From the popularity of each framework, we can draw conclusions about the maturity and community. As we can see, Vue is the most popular based on the number of stars. But it is used by the least number of projects. Therefore, there might be a lack of resources in case of issues. It also has the least number of contributors, although React and Angular being backed by Facebook and Google respectively might be the reason those two have much larger numbers [21]. Main highlights of each library:

React	Angular	Vue
Flexible	Steep learning curve	Flexible
Relatively mature	MVC architecture	Relatively new
Widely used	Uses Typescript	Easy to learn
Easy to learn	Mature	

Table 2.3: Highlights of frontend frameworks

2.6 Redux

Redux is a predictable state container for JavaScript apps [22]. What does this mean? Every application has its state. It might contain response from a server, the page user is on or maybe some user data. When the application grows, it is easy to introduce errors, if state is not managed correctly. This is because the state of the application can often be mutated by many different actions asynchronously [23].

Debugging these problems then becomes increasingly hard and sometimes always impossible. There also might be many sources of truth if the state is not centralized. This is what Redux is trying to solve. It aims to provide single, centralized source of truth. This allows for an easier state management or even "time travel" debugging. It is extensible and has a wide array of addons, including React bindings [22].

2.7 Diff libraries

Before starting the development of this application, it was necessary to get to know the state of Markdown diff libraries. From a quick research it became apparent, that these libraries use plain-text diff libraries at their core, which might cause some problems.

2.7.1 Plain text diff libraries

There are two important plaintext libraries - the most widely used JSDiff and diff-match-patch library made by Google. Both libraries use the algorithm proposed in "An O(ND) Difference Algorithm and its Variations" by Eugen W. Myers. The diff-match-patch library then applies various filters, that aim to make the output cleaner and more human readable. In one of the Markdown diff libraries the `difflib` library was also used, but this library was last updated in the year of 2012 and the respective GIT repository does not show any signs of activity.

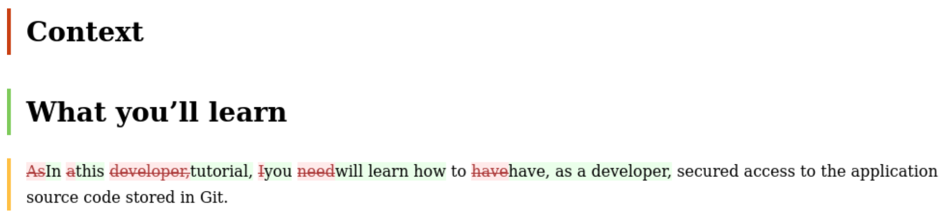
2.7.2 Markdown diff libraries

Markdown diff libraries are mostly based on beforementioned libraries, except for `react-rich-diff`. Each library does things a little bit differently - let it be the flavour support or how they handle various markdown elements. Below you will find a quick summary of flavour support and underlying implementation of each library.

Name	Markdown flavor	Implementation
markdown-diff	GitHub Flavored Markdown	JSDiff
@ads-vdh/md-diff	CommonMark	difflib
rich-text-diff	GitHub Flavored Markdown	JSDiff
react-rich-diff	GitHub Flavored Markdown	Own implementation

Table 2.4: Diff library overview

Notice, that we do not see any library that would support MDX. Moreover the `rich-text-diff` and `react-rich-diff` libraries are already 5 and 3 years without any activity respectively and the other libraries do not seem to be doing much better, thus any help from the developers is unlikely. The MDX support and eventual problems will have to be tested and fixed by the application developer. But the differences do not end there, the outputs also differed in quality and representation. Below you can see an image of `react-rich-diff` output, which differed from the other libraries:



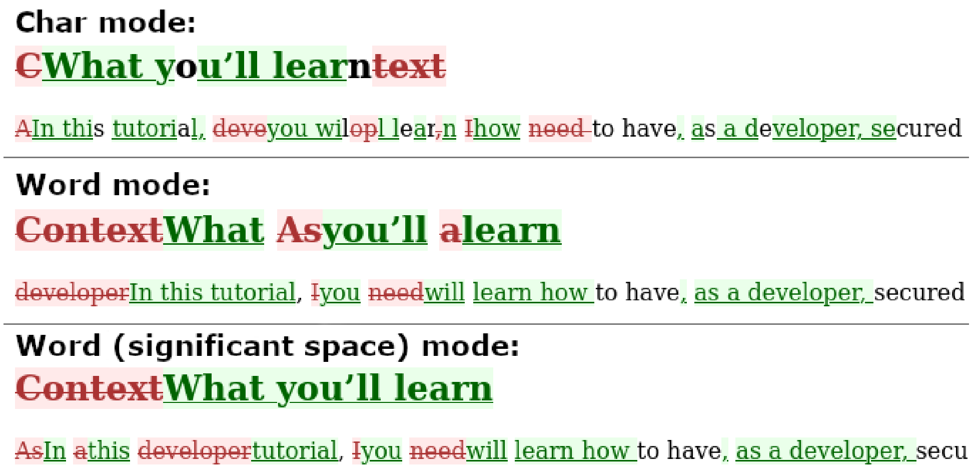
Context

What you'll learn

~~As~~In ~~a~~this ~~developer,~~tutorial, ~~I~~you ~~need~~will learn how to ~~have~~have, as a developer, secured access to the application source code stored in Git.

Figure 2.7: Output of the react-rich-diff library

The different quality of the results was caused mainly by the used diff mode. For example, `markdown-diff` uses char mode, which means it highlighted every different letter. That is not ideal, because when used by humans, the diff is not easy to read and even more so to understand. Just by changing this mode we achieved a big difference in readability without sacrificing the correctness of the output. Google Docs output was added for comparison.



Char mode:

~~C~~What you'll lear n~~text~~

~~A~~In this tutorial, ~~deve~~you wil~~op~~l lear~~,~~n ~~I~~how ~~need~~ to have, as a developer, secured

Word mode:

~~Context~~What ~~As~~you'll ~~a~~learn

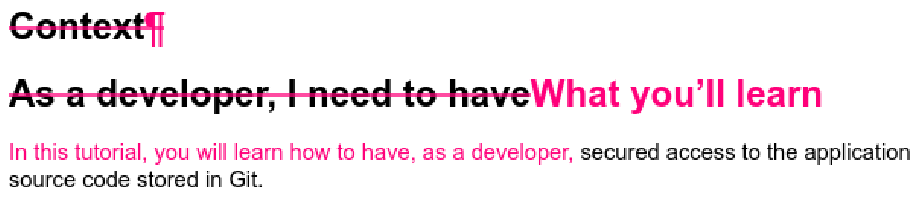
~~developer~~In this tutorial, ~~I~~you ~~need~~will learn how to have, as a developer, secured

Word (significant space) mode:

~~Context~~What you'll learn

~~As~~In ~~a~~this ~~developer~~tutorial, ~~I~~you ~~need~~will learn how to have, as a developer, secu

Figure 2.8: Different JSDiff settings



~~Context~~¶

~~As a developer, I need to have~~What you'll learn

In this tutorial, you will learn how to have, as a developer, secured access to the application source code stored in Git.

Figure 2.9: Google Docs output

2.8 Markdown editors

Although Markdown is a widespread language, there are not many Open-Source editors to choose from. Even more so if you require the editor to support WYSIWYG mode. There are some editors that seem well made, but the developer stopped its support a long time ago. This would mean that the burden of bugfixes and improvements would be on the application developer. Some of the most notable popular editors are:

- **SimpleMDE**
  - no active development
- **EasyMDE**
  - Fork of SimpleMDE, under active development
  - has side-by-side mode allowing seamless write & preview
  - has autosave built in
  - provides option to change the built-in markdown renderer
  - easy React setup using react-simplemde-editor wrapper
- **TOAST UI Editor**
  - under active development, most active community
  - works well with frontmatter
  - provides option to disable the built-in markdown renderer
  - has side-by-side mode allowing seamless write & preview
  - easy React setup
  - biggest package size of all mentioned editors
- **React MDE**
  - under active development
  - provides option to change the built-in markdown renderer
  - easy React setup

Same as with the diff libraries, there are not any WYSIWYG editors for the MDX format. The only available MDX editor found was **ok-mdx**, but it does not fit the needs of our application, as it doesn't offer WYSIWYG mode and is made to work with local files only.

2.9 Application access to GIT

As one of the most used parts of code would be the GIT connection, the first step was to find out what are the recommended and most common ways to implement such a system. From research, some common GIT access implementation approaches emerged. The implementation and their implications are described below.

2.9.1 Credentials

There were two main approaches when it comes to credentials. One approach is to create a service user. This user might be a regular user or if the GIT hosting allows, it can be a bot. This approach has one big flaw - if the user account or application is compromised, all the repositories it has access to are compromised too.

This might not be a big issue if the access was read-only or if it was a bot that automatically closes issues and cannot do much more. And although having company secrets leaked is a big deal, malicious code inserted into a repository with production code could do more harm. For the purpose of this thesis, we need full read write access on the repository - meaning the attacker could do a lot of damage and a malicious code would be hard to find, as all the commits would be made by one account.

The other implementation utilizes the user's own account. This works much better for our purposes - each user has access only to the repositories, that they require for their work. In case of unauthorized access, the commits can be easily located and reverted, and the access revoked, and the application will still work for other users.

2.9.2 Local access vs API

Local access

One of the common approaches is accessing locally cloned GIT. This poses many challenges and would work best combined with the service user approach. Again, there are two main implementations of this kind of access - utilizing an installed GIT application or the `ligbit2` library. The first approach was used in the prototype app and did not work very well with blob files - the problem was that it utilized the output from terminal and as such, it expected to be string.

The conversion of the binary file to string is not lossless - should the parser encounter unrecognized characters; it will replace it with a `U+FFFD` character [24]. Therefore, the issue couldn't be resolved, and the approach was deemed as unviable.

The second approach would utilize the `libgit2` library. This library implements a small set of GIT functions and accesses the repository directly. Therefore, this approach would solve the issues with blob files, but the limited set of functions could cause problems. Still there are other issues that this second approach does not solve, such that we would need to solve conflicts when updating the local repository and more.

API access

Using the API to communicate with the GIT repository host is by far the easiest way to work with the GIT repository. The access can be obtained via OAuth2 when the user logs in into the application, although the service user approach can still be used. As most of the providers have official or unofficial libraries that provide a wrapper around their APIs, the implementation is also fairly simple.

2.9.3 Authentication

After a consultation with the supervisor, it was decided, that authentication using GitLab OAuth2 API would be sufficient. For proof-readers that do not have a Gitlab account, the account would be created by the company. This also gives us the opportunity to ask Gitlab for the API keys required to access the repositories. These keys will then be stored inside the database for reasons described in section 3.4.1.

As the HTTP protocol is stateless, we also need a mechanism to keep the user logged in. There are many approaches to solving this problem. The most prevalent are session based auth and JWT.

Session based auth

Session based auth keeps the session data stored on the server. When first contacted, the server sends a cookie with a session id to the browser. The browser then sends this cookie every time it accesses the server again [25]. This way the server "remembers" who the user is. The main benefit of this approach is that it is easy to implement, but as the identifier is specific to the server, it causes problems with scalability. This approach also caused problems[2] during prototyping, therefore JWT based auth was introduced.

JWT

JWT tries to solve the main problems of the session based auth - it uses tokens to create a universal identifier that can be used to authenticate the user. This token is then stored on the client side and does not have any meaning for the client. This approach allows for a great scalability but adds another layer of complexity. All JWTs are constructed from three different elements: the header, the payload, and the signature/encryption data [26]. This token has no meaning to the client and is meant solely for the server. But still, some new problems are introduced.

The main problem is that JWT tokens are not easily revokable as they are meant to be stateless. If they were, then each service would need to check a database of active/revoked tokens before serving the request and that means scalability issues. Some exploits for this approach were also discovered [27]. To solve some of these issues long lived revokable refresh tokens tend to be implemented. These are used to request new tokens once the short-lived access tokens expire.

2.10 Other related software

2.10.1 StackEdit

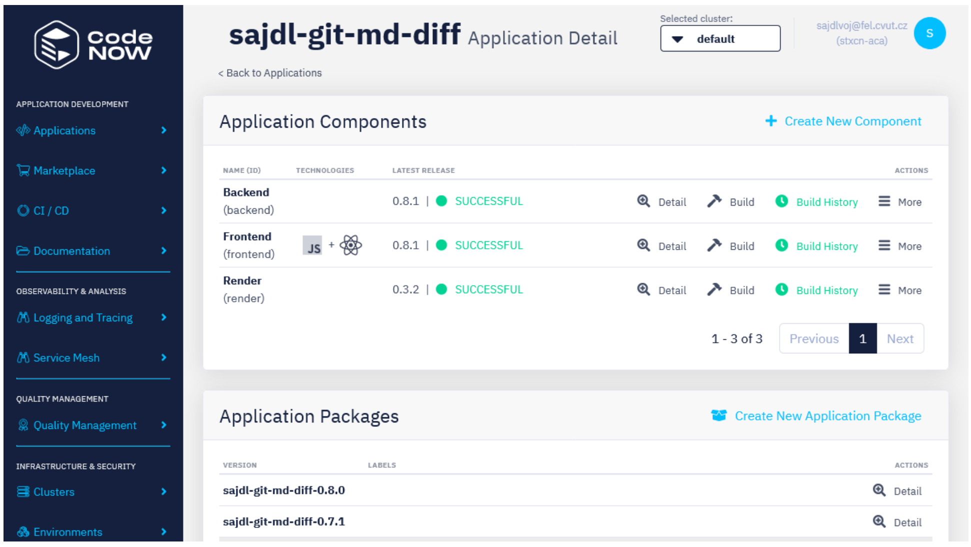
Stack edit is an in-browser Markdown editor. It supports WYSIWYG editing, simultaneous preview with scroll synchronization, comments, collaboration and much more. It can even connect to GitHub and GitLab to edit files stored there. This application seems to be a good candidate to use for documentation management, but it has its own problems.

[2]If the server was restarted, the developer would need to log in again. Even with code hot swap, this could occur relatively often over the course of several minutes of development.

While this application works well for basic use cases such as file edit, the setup is somewhat complicated. It requires user to create a new application in their GitLab account, give it correct permissions and then configure which project they want to use. This would be hard for non-technical users such as proof-readers. Also, it does not support creating or deleting the repository and the user can only edit one branch at a time (branches need to be imported separately).

2.10.2 CodeNow platform

For the purpose of this thesis, access to the CodeNow Software Factory, made by Stratox Enterprises s.r.o. was provided by the supervisor free of charge. CodeNow aims to provide an all-in-one solution that would cover the entire software delivery lifecycle. Everything is done automatically - application building, deployment, testing or even logging and monitoring. To deliver this solution, it uses Open-Source software, so the danger of vendor lock-in is not present [28]. The applications are deployed using docker images, which can be built in the platform. Configuration can also be managed there, but some constraints apply - such as that it all must be present in one specific folder. Below, you can see a view of a successfully built version of the application on the CodeNow platform.



sajdl-git-md-diff Application Detail

sajdlvoj@fel.cvut.cz (stxcn-aca)

Application Components

NAME (ID)	TECHNOLOGIES	LATEST RELEASE	ACTIONS
Backend (backend)		0.8.1 | ● SUCCESSFUL	Detail Build Build History More
Frontend (frontend)	JS +	0.8.1 | ● SUCCESSFUL	Detail Build Build History More
Render (render)		0.3.2 | ● SUCCESSFUL	Detail Build Build History More

1 - 3 of 3

Application Packages

VERSION	LABELS	ACTIONS
sajdl-git-md-diff-0.8.0		Detail
sajdl-git-md-diff-0.7.1		Detail

Figure 2.10: Codenow user interface

Chapter 3

Design

3.1 Architecture

The first important decision was to select an architecture, that would allow for a good scalability and speed of development. The architectural section will focus mainly on the high-level design, thus describing technologies used and not the internal implementation.

3.1.1 Monolithic architecture

Although monolithic applications are usually easier to develop, debug, test and deploy, the negatives come to light at a later time. As the monolithic architecture has inherent scalability issues, the architecture becomes a problem when a large user base starts using the application. Furthermore, as the application grows the codebase becomes harder to understand and manage. Should the need to switch to a different technology arise, the changes would need to be made to the entire code [29].

3.1.2 Microservice architecture

On the other hand, there is the microservice architecture. This architecture is based on independent components, providing much more flexibility. Moreover, if one service fails or has a bug in it, the rest of the system remains operational. Additional benefit of this architecture is that there is less code in a service for the programmer to understand and scaling is also much easier unless the application is implemented incorrectly. Technologies can be switched relatively easily, as the change can affect only one microservice at a time.

The downsides of this approach are that it adds complexity. You need to set up logging systems, health monitoring and handle the communication across all the microservices and deployment alone becomes harder. Testing also becomes more complex, as each microservice must be tested separately.

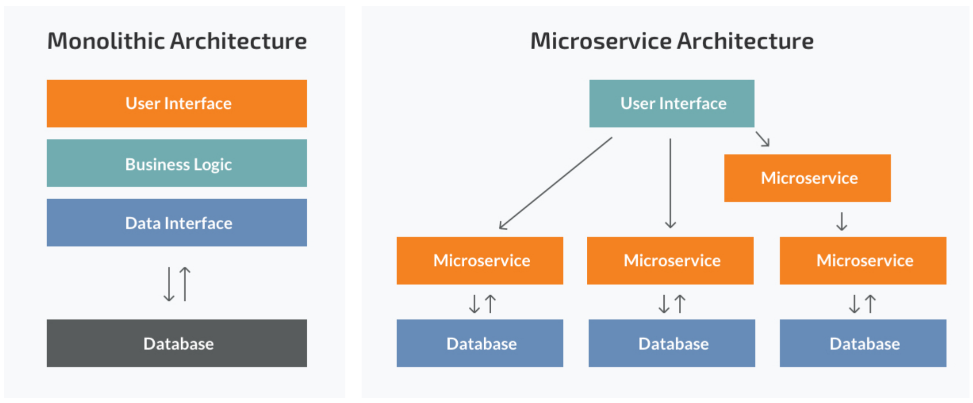


Figure 3.1: Microservice and Monolithic architecture side by side [29]

3.2 Markdown diff and render

During prototyping all the markdown diff and render libraries were evaluated. Ideally the entire diff and rendering process should be offloaded to the browser so as to not require an expensive infrastructure. However, at the early stages of prototyping it has become clear, that the MDX library had problems running in a browser environment. This meant that either we would need to sacrifice render feature parity to Docusaurus and use a different parser, or that a render server would need to be deployed.

3.3 Final architecture

The decision was made to utilize a hybrid approach between monolithic and microservice architecture. The basic idea was that there would be one API server and one frontend server, serving a single page application.

The API server could be further separated into an auth and documentation/proofreading microservices, but for the sake of simplicity of development, all of it stayed on the same server in the current implementation. The Auth part can be separated at any time, as it should not be coupled with any other part of the application.

The frontend library was selected to be React, because it is the most widely used and we are unlikely to encounter any major problems. The server language was selected to be JavaScript, using NodeJS as the runtime environment, because if some code would be found to be slow in the browser, porting it to the server would be simpler than with other languages.

There are also many toolkits that allow to generate the basic application structure - for example React has its own create-react-app, and there is the Neutrino project, which can generate multiple different presets. CodeNow has their own scaffolders for their supported languages too, but access to the CodeNow platform was obtained later in the development, therefore their scaffolders could not be used from the start. In the figure 3.2 you can see the proposed deployment of the application.

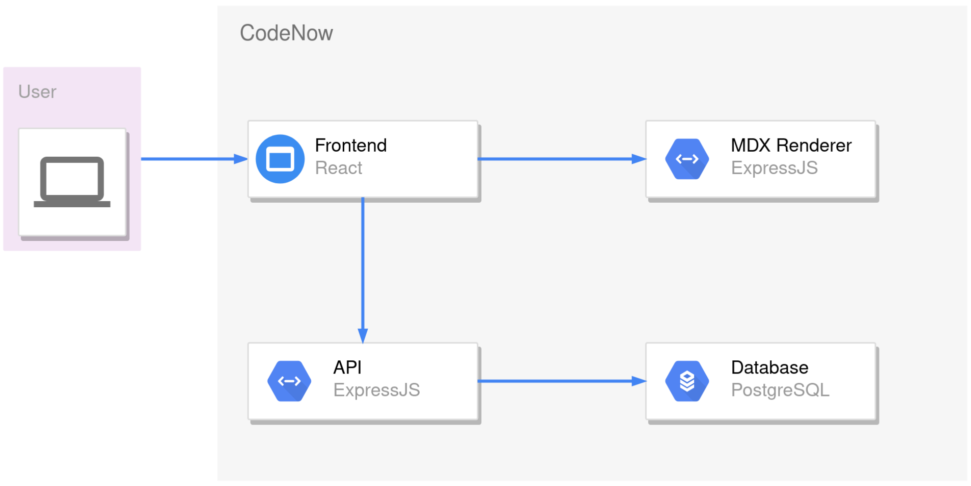


Figure 3.2: Application architecture

3.3.1 NodeJS & Express

Before we finish this section, we should also quickly introduce the beforementioned NodeJS and its most prevalent web framework library - ExpressJS.

Node.js is a JavaScript runtime environment built on Chrome's V8 JavaScript engine, designed to build scalable network applications with the help of its asynchronous, event-driven nature [30], [31]. It is open source, cross-platform, and since its introduction in 2009, it got hugely popular [32].

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications [33]. It builds on top of its features to provide easy to use functionality that satisfy the needs of the Web Server use case [34]. The combination of NodeJS and Express is widely used and already very mature. As of writing this thesis, ExpressJS was used by over 9.7M projects based on data from the Gitlab repository.

3.4 Database

The database choice was mainly constrained by the CodeNow platform support. A relational database is best suited for this application; therefore, we are left with two options - the PostgreSQL and CockroachDB. The database for this application was chosen to be PostgreSQL mainly because of its better NodeJS support.

3.4.1 Database model

The database model was designed in such a way, so that we could minimize API communication, primarily in the proofreading case. Also, the possibility to add more GIT hosting providers was considered. To achieve this, each documentation has the provider where it is hosted specified. However, in the current implementation, this will only be Gitlab. Linking user account to multiple providers is also possible, thanks to the JSONB columns `linked` (links account with provider account ID) and `tokens` (keeps access tokens stored for our use). The structure of the linked and tokens fields is as follows:

```
Linked account:
{
    <provider>: {
        <number>
    }
}

Tokens:
{
    <provider>: {
        access: <token>,
        refresh: <token>
    }
}
```

Note that in the current implementation the tokens could easily be stored in the JWT payload, when obtained from the provider on login. This would not work when the support for more providers would be added - the user would always need to login through the provider they wanted to use. This would not be very user friendly, so the approach described above was utilized.

The tokens table contains JWT refresh tokens, and the app checks whether the token is present (if it is, it was not revoked) before it generates new access token for the user. The rest of the tables and their columns is self-explanatory, apart from proofreading request. The columns `sourceBranch`, `targetBranch`, `revFrom` and `revTo` are all GIT-related. But why are they there, when we could simply open a merge request that would store this information for us?

The main reason is that we need to access this data often. It is used in the file diff part of the proofreading request as well as to indicate which branch the file should be saved to and various other features. This way we know all the information needed and the merge request can be created once the proofreading is finished, minimizing the need to access the Gitlab API. The modified column exists for the same reason - so that we know which files were modified by the proof-reader without having to request a diff from Gitlab.

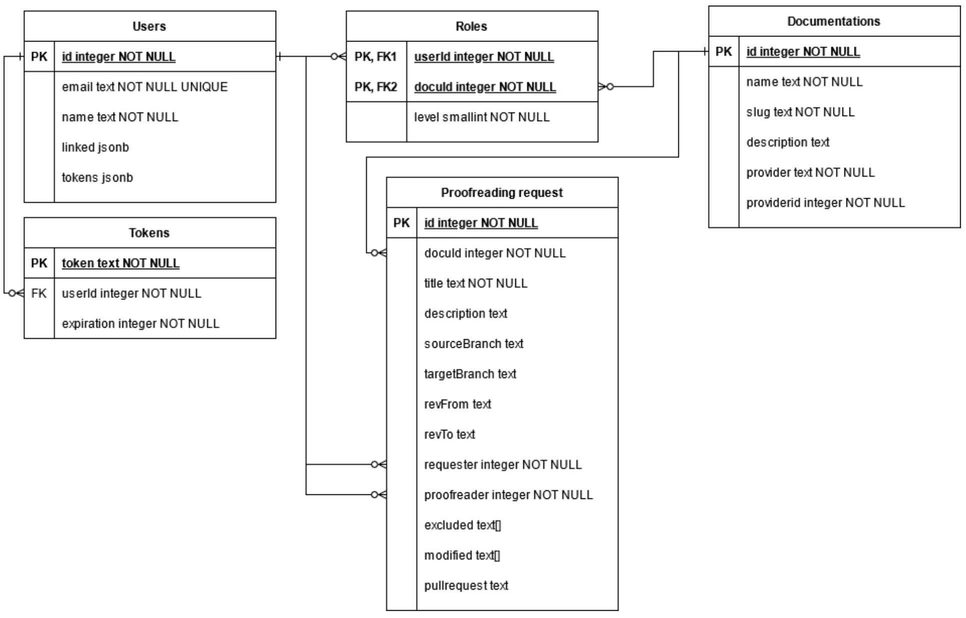
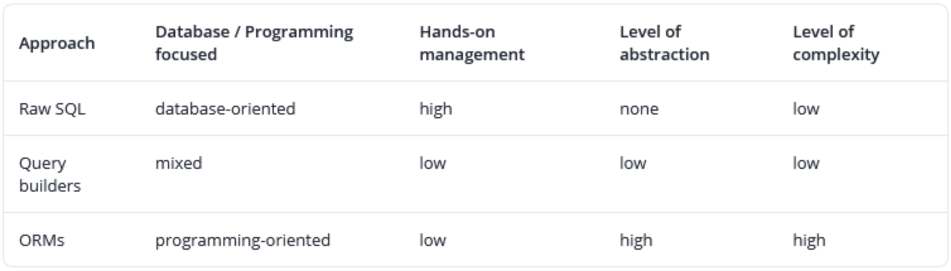


Figure 3.3: Database diagram

3.4.2 Database access

There are various approaches to database access - raw SQL, query builders and ORMs can be used. Each has their own benefits and drawbacks, some of which can be seen in the figure below.



Approach	Database / Programming focused	Hands-on management	Level of abstraction	Level of complexity
Raw SQL	database-oriented	high	none	low
Query builders	mixed	low	low	low
ORMs	programming-oriented	low	high	high

Figure 3.4: SQL vs Query builders vs ORM [35]

The choice of the approach is often dependent on the skills and preferences of the developer. NodeJS has a great collection of tools for all of these approaches. Although the other options may be easier for some people to use, I personally like to know what exactly the application is querying, therefore I focused on the Raw SQL approach.

Database access libraries

As of writing there were multiple libraries to choose from, namely: `pg`, `postgres` and `slonik`. As of writing the `pg` library was the most popular one and `postgres` was by far the fastest one. As the `postgres` library had a good documentation and an active developer, it was used in the final application.

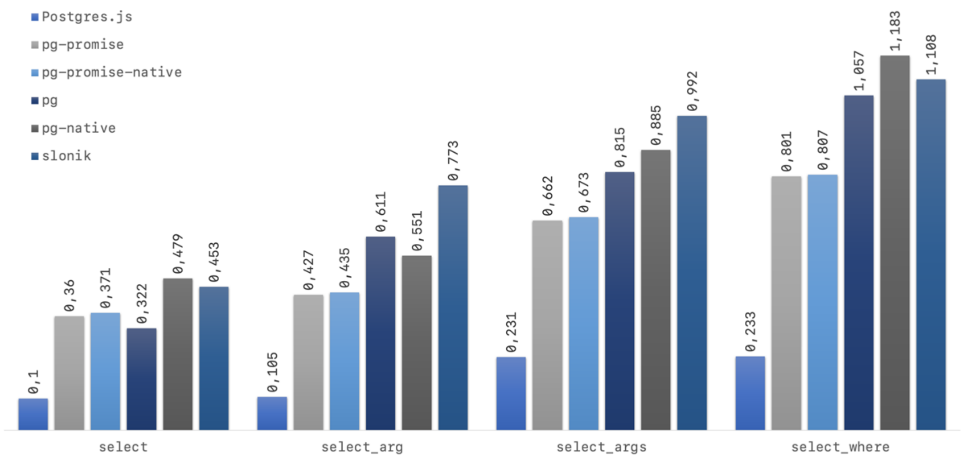


Figure 3.5: Postgres database libraries benchmark measured in *ms* [36]

3.5 OAuth2 login process

The OAuth2 login process works great when used with a monolithic, session auth based application, but brings another layer of complexity while creating a microservice app. This is mainly because the process of frontend and backend communication needed is not well described. A library for PassportJS was found, but the implementation was not straightforward, as basically every example assumed a monolithic architecture. Note that a simpler implementation using an auth provider such as Auth0 could be used, but it would also mean that we might need to use a paid plan in the future.

First, we need the access tokens to be sent to the NodeJS API server, so that we can store them in our database. Second, the API server needs to somehow let the frontend know that the user logged in successfully and transfer the appropriate JWT. After a while of research, the process was designed as follows:

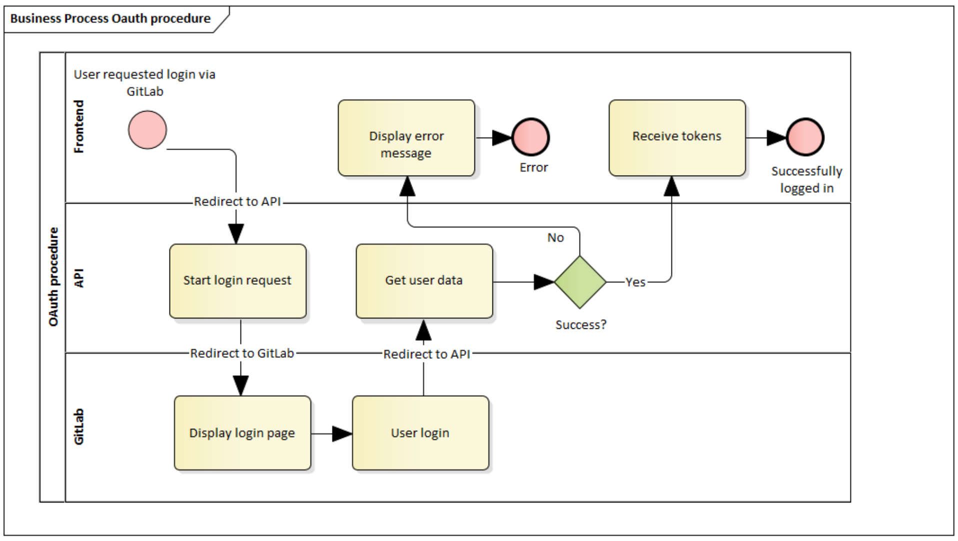


Figure 3.6: GitLab Oauth2 login procedure diagram

As can be seen from the diagram, the resulting JWT are sent from the backend to the frontend. This is done via a redirect to a special URL, in which these tokens are contained. This is not a security risk, as the HTTPS protocol encrypts the entire request. This basically means that a hacker may be able to figure out which host name the user is connected to but not the rest of the URL, keeping the tokens safe [37].

3.6 User Interface

The user interface design was an important focus of this thesis. The aim was to provide the users with an UI that would be easy to navigate in and more importantly fast to get things done.

3.6.1 User interface design

A good user interface design is fundamental for any applications success as there often are a lot of competitors. Therefore, a user centred design is nowadays a requirement, not an advantage over the competition.

Properly analysing the collected requirements and understanding their meaning is essential for a good UI and UX design [38]. User centred design has to also account for many problems that are not mentioned in any requirements - for example if the application has a long-running task, the design should be created in such a way, that the user will know that something is happening, and the application didn't just freeze.

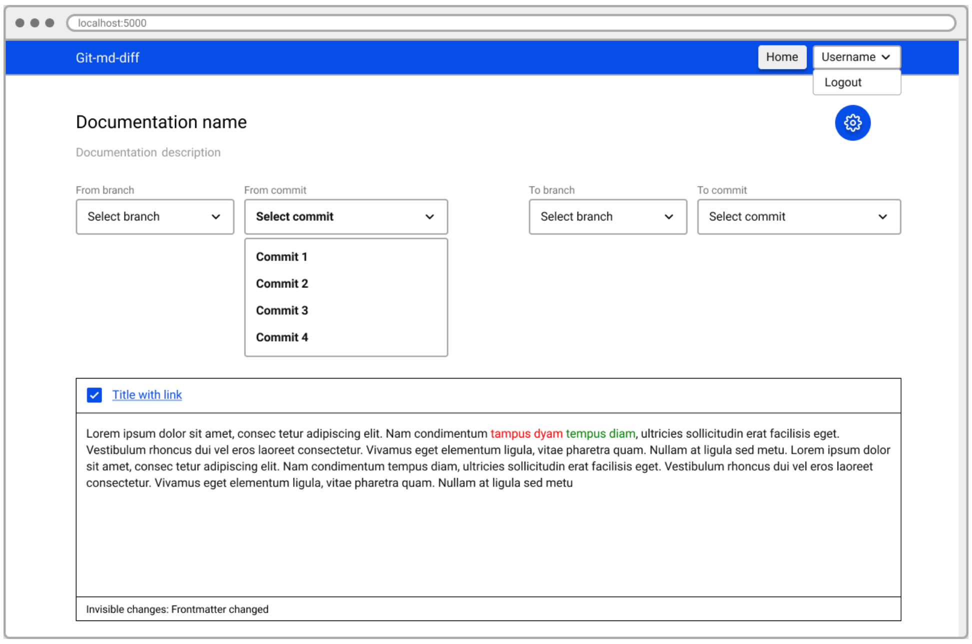
Although it would seem so, to create a good UI design, you do not have to consult with users all the time. There are some general rules, that if utilized properly will help us create a good UI/UX. These include:

- Proximity - items close together seem related, therefore they should be
- Visibility - size of elements and their contrast can make user notice them first
- Hierarchy - if a hierarchy exists, it should be discernible in the UI
- Mental Models - confirmation dialogs, utilization of Fitt's Law and consistent UI are just some examples of how we can help improve the user experience

Another good technique to use is creating wireframes before an actual implementation. Wireframes are used to represent the user interface and describe the hierarchy of components in the final product. These wireframes then can be consulted and validated with users even before implementing the design [39]. In the next sections we will focus on the low fidelity wireframes created before the main implementation.

3.6.2 Early wireframe prototyping

The UI prototyping went through two phases - first the prototypes for the main documentation view designated for the documentation manager (the diff view) was designed along with the edit page. Most of those wireframes stayed the same during the development, although the wireframe of the diff view was updated, because the branch selection was not present in the initial design. Other changes such as the user menu and cog wheel for documentation settings were added in the next design iteration to more reflect the actual design that would be used.

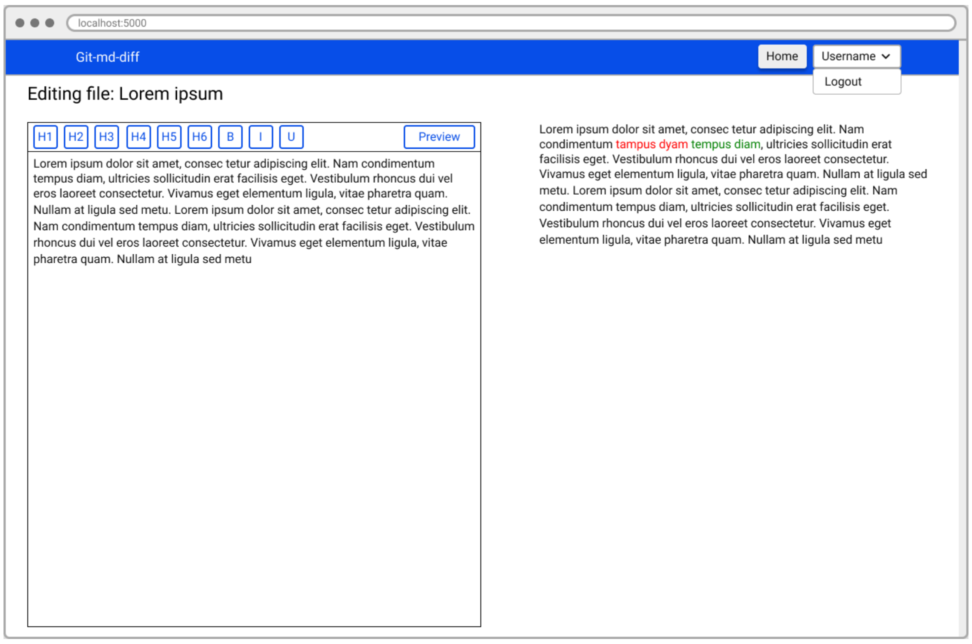


Lorem ipsum dolor sit amet, consec tetur adipiscing elit. Nam condimentum tampus dyam tempus diam, ultricies sollicitudin erat facilisis eget. Vestibulum rhoncus dui vel eros laoreet consectetur. Vivamus eget elementum ligula, vitae pharetra quam. Nullam at ligula sed metu. Lorem ipsum dolor sit amet, consec tetur adipiscing elit. Nam condimentum tempus diam, ultricies sollicitudin erat facilisis eget. Vestibulum rhoncus dui vel eros laoreet consectetur. Vivamus eget elementum ligula, vitae pharetra quam. Nullam at ligula sed metu

Invisible changes: Frontmatter changed

Figure 3.7: Wireframe of the diff view

At this point the design of the edit page assumed, that the preview would be implemented as an in-editor preview. However later due to the editor's limitations, this was found to be impossible, and the design had to be changed. This change is not reflected in the wireframe below but can be seen in the figure 5.4.



Lorem ipsum dolor sit amet, consec tetur adipiscing elit. Nam condimentum tempus diam, ultricies sollicitudin erat facilisis eget. Vestibulum rhoncus dui vel eros laoreet consectetur. Vivamus eget elementum ligula, vitae pharetra quam. Nullam at ligula sed metu. Lorem ipsum dolor sit amet, consec tetur adipiscing elit. Nam condimentum tempus diam, ultricies sollicitudin erat facilisis eget. Vestibulum rhoncus dui vel eros laoreet consectetur. Vivamus eget elementum ligula, vitae pharetra quam. Nullam at ligula sed metu

Lorem ipsum dolor sit amet, consec tetur adipiscing elit. Nam condimentum tampus dyam tempus diam, ultricies sollicitudin erat facilisis eget. Vestibulum rhoncus dui vel eros laoreet consectetur. Vivamus eget elementum ligula, vitae pharetra quam. Nullam at ligula sed metu. Lorem ipsum dolor sit amet, consec tetur adipiscing elit. Nam condimentum tempus diam, ultricies sollicitudin erat facilisis eget. Vestibulum rhoncus dui vel eros laoreet consectetur. Vivamus eget elementum ligula, vitae pharetra quam. Nullam at ligula sed metu

Figure 3.8: Wireframe of the edit page

Along with the first prototypes, the UI was to be implemented. It was at this time that the question of which UI framework to use arose. There is a wide variety of frameworks that aim to help the coders to create and prototype user interfaces quickly, but not every one of them has a complete React implementation. Why is that important? More often than not, these frameworks enable us to create dropdowns, modals and various other elements that require JavaScript to work properly. Having those elements implemented in React makes it easier to use them as they behave as a React developer would expect. Due to previous experience, I preferred the Bootstrap framework. It even had ready-made React bindings library `react-bootstrap`.

Bootstrap UI framework

Originally created by a designer and a developer at Twitter, Bootstrap has become one of the most popular front-end frameworks and open-source projects in the world. It went through a major rewrite in each version, incorporating new technologies such as media queries and flexboxes [40]. As of writing, Bootstrap is used by over 2.5M projects worldwide.

3.6.3 Final product wireframe prototyping

After implementing the initial app version, it was time to design more of the UI. There were additional features to be added that were not included in the original version: Dashboard, Documentation settings, Documentation creation, File view and the entire proofreading process.

Some of the features would use the same or very similar wireframes as in the previous iteration - mainly the proofreading page and file edit while proofreading. Also, the creation of a new documentation and the proofreading request creation are just simple forms, therefore these wireframes were not created.

First off, we have the Dashboard wireframe. In this wireframe the main focus was on simplicity. The proposed tile layout of the proofreading requests and documentation was assumed to make it easier for users to select the correct item, as the clickable area is rather large. It also gives us the space to fit in more information in the proofreading request tile, such as the proof-reader and submitter name.

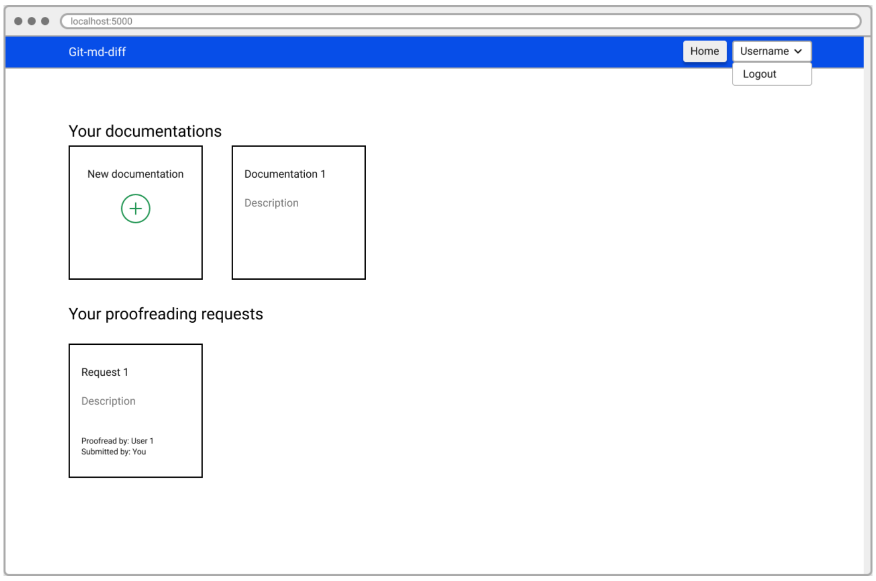


Figure 3.9: Wireframe of the dashboard page

As for the files page, the main focus was easily distinguishable hierarchy as well as making it easy to create or delete a new file. The file deletion should also have confirmation dialog not displayed in this wireframe. At this stage, the problem that is not solved is that we need to indicate which files can be edited, as the application supports only Markdown files.

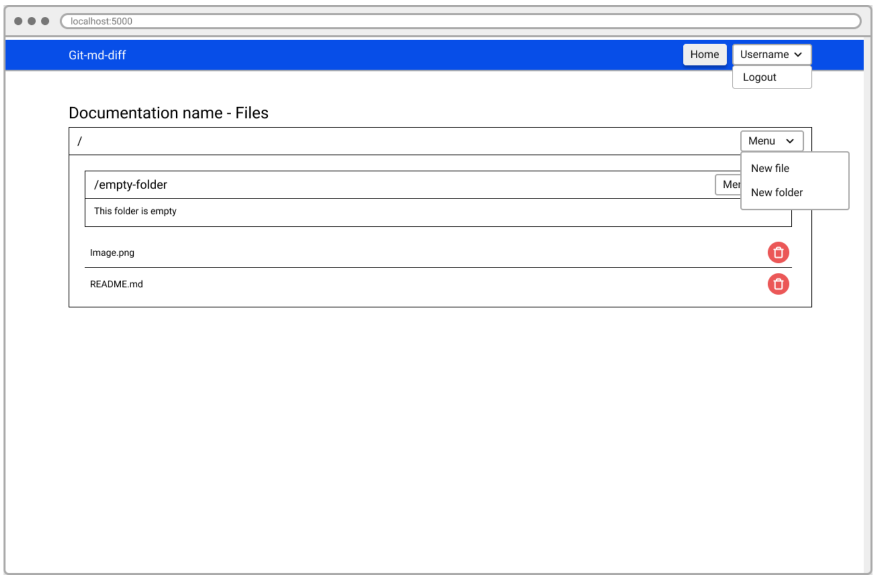
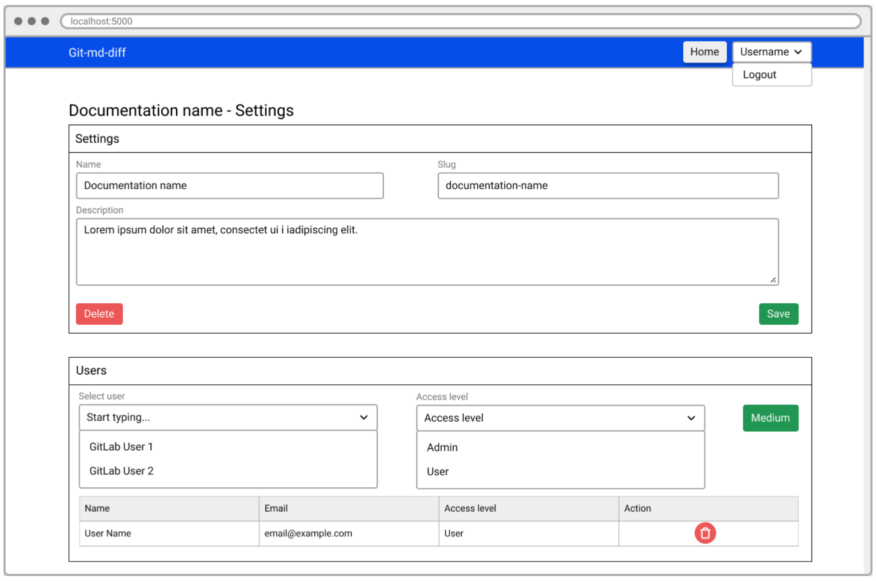


Figure 3.10: Wireframe of the files page

Finally, the documentation settings page design. The main point of this wireframe is to illustrate, how the user management will work, as all other documentation settings can be solved by a simple form. The user dropdown is meant to be searchable, searching all users registered on the Gitlab instance. The information that could be obtained about each user was not known at this point, therefore the design of the dropdown is taking available information into account. Also, the access level list was shortened for convenience.



Name	Email	Access level	Action
User Name	email@example.com	User	

Figure 3.11: Wireframe of the settings page

Chapter 4

Implementation

The implementation could be split into two phases - prototyping (as mentioned in previous sections) and the final implementation. Only the important parts of the prototyping will be described.

4.1 Prototyping

The prototyping iterated over a few solutions. The first two prototype versions were CLI based and meant to solve the immediate problem at hand - generating Markdown diffs that could be at least copy-pasted into a word processor to help facilitate proofreading. The CLI based versions utilized Bash and NodeJS.

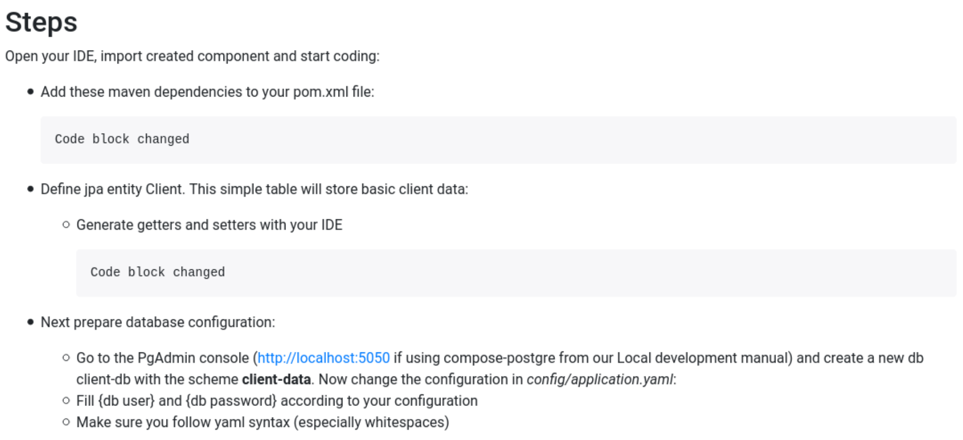
The difference between them was that the first version could generate Markdown diffs locally, but they were not very usable yet. The second version uploaded them directly to Google Drive to allow to utilize its diff algorithm. This script utilized the ability to convert HTML to a Google Document, but the diff could not be done automatically, as Google does not provide such an API endpoint.

After finishing this implementation that was requested by the supervisor, we moved onto the planned web version of this application. During this phase, a simple React application was created, and the viability of a web solution was tested. The first point was whether it is possible to generate diffs and render them in the browser. Also, the viability of different GIT access methods was explored. Immediately problems were encountered while trying to render the Markdown.

When trying to use the MDX toolkit, it was found, that it had problems running in a browser environment. To render the markdown with at least partial feature support, Remark was then tried. When we implemented Remark as the main parser, it slowed the browser down noticeably. Later, the marked and `markdown-to-jsx` parsers were tried. Both worked fairly well, but although the feature parity to Docusaurus was not specifically required, it would be a great help, because the end user could easily preview what they were doing.

Therefore, more effort to get the MDX parser working was made, trying to get it working via server-side rendering. Including it on the same server was deemed as a bad idea (mainly due to performance concerns), therefore a new service was made. Deployed in such a manner, the MDX library started working and rendering similar to the Docusaurus could be done.

The diff libraries were compared at this time, and the markdown-diff library was later chosen as the primary diff library, due to its readability and thus ease of modification. Some diff enhancements were also introduced at this time, mainly detecting changes inside the code blocks and shortening them using a placeholder, as they are not important to the proof-reader. Other work included image placeholders and eventually image loading. This was all solved using regex expressions.



Steps

Open your IDE, import created component and start coding:

- Add these maven dependencies to your pom.xml file:

  ```
  Code block changed
  ```

- Define jpa entity Client. This simple table will store basic client data:
  - Generate getters and setters with your IDE

    ```
    Code block changed
    ```

- Next prepare database configuration:
  - Go to the PgAdmin console (http://localhost:5050 if using compose-postgre from our Local development manual) and create a new db client-db with the scheme **client-data**. Now change the configuration in *config/application.yaml*:
  - Fill {db user} and {db password} according to your configuration
  - Make sure you follow yaml syntax (especially whitespaces)

Figure 4.1: Code block placeholder example

During prototyping phase, the application was used as a local app, therefore prototyping started by implementing local GIT access using the installed GIT application. This implementation worked fairly well, until the image display feature started to be implemented. During its implementation problem with the string conversion described in section 2.9.2 was discovered, making any blob file corrupted. But at this point, the proof-of-concept implementation was good enough and we moved onto the final product, where the approach would be changed.

4.2 Final product

During the final implementation it was important, to have the API server ready to be extensible. Due to the choice of programming language, some limitations were present - mainly the inability to create interfaces and enums. Also, the structure of either a React or NodeJS project is not fixed, it is up to the programmer to choose their preferred way to write the application.

4.2.1 API server

The prototyping stack was designed, to make it easy for me to work with. At this point, the problem with interfaces and enums needed to be addressed to move forward. As I have a good experience with plain JavaScript, I chose to change my approach to designing the application structure to account for the missing interfaces. The technique to emulate enums was easy - for a basic implementation we can use an object containing explicitly stated key-value pairs and then use the widely supported `Object.freeze()` API to prevent any object modification. These problems could be solved with the use of Typescript too, but at the time CodeNow did not support generic Docker images and Typescript was not explicitly listed as supported.

The implementation was built still tried to account for the future possibility to add more GIT hosting providers. Although the API server was not split into several microservices, it was designed to keep the services separate and clearly distinguish the API endpoints of each service. Moreover, both the documentation and proofreading services need to access the GIT hosting provider. The API endpoints were designed using the REST API approach.

The GIT hosting provider is accessed through a wrapper class, that selects the appropriate implementation based on the provider requested. At the moment only Gitlab client is implemented. This approach should ensure that the application does not need to be entirely rewritten when a new provider would be added. Although Gitlab promotes `GitLab-Yaac` on their website, it has not been updated for over 6 years. Therefore, to implement the Gitlab client, the library `Gitbeaker` was used, as it was by far the most popular and feature-complete.

4.2.2 Frontend

The `React` and `Redux` combination was used to implement the frontend, with the addition of `wouter` to enable us to use multiple pages. Several other libraries, such as `react-select` used to implement searchable dropdowns and `slugify`, used to automatically create slugs from documentation names were also introduced. The most notable addition that sped up the development process was the addition of the `ky` library, which is a tiny and elegant HTTP client based on the browser Fetch API [41]. The aim of this library is to provide an easy-to-use wrapper to the browsers fetch API. Thanks to this library, the requests became much easier to implement and are much more readable - especially those containing a JSON body.

The application structure aimed to mainly separate the components and pages as some of the components would be reusable. Redux was also utilized to make the state management easier, mitigating problems with passing the values through many components. In the end the frontend manages the file diffing to lessen the load on the API and the diff is then sent to the Render service.

Before sending the content to the service, it is modified using regex expressions that replace the url of the images contained in the page with our own. This ensures that the images are working once the content is rendered. When the image is rendered a request to the API server is made and the API server in turn requests the file from Gitlab. The tokens need to be passed along with the requested file to prevent unauthorized access; therefore, it has to be included as a part of the URL.

4.3 Problems

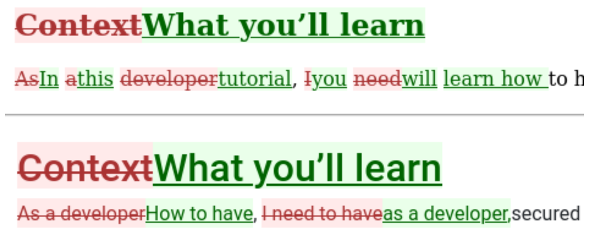
Over the course of the final implementation, many problems were discovered and resolved. The most important ones are described below.

4.3.1 Diff quality

The diff quality problems were already discovered during the prototyping phase. In the final implementation, these problems had to be fixed. The main two problems were readability (as mentioned in analysis) and validity (the MDX library refused to render them). To solve this issue, the original library was forked and modified.

Readability

The readability issues were explored previously, but some still remained. The main improvement was seen in implementing a way that multiple consequent additions and deletions could be merged, to make it more readable. A solution utilizing a lookahead approach, that would stop should any non-modified or special block be encountered, was designed.



~~Context~~What you'll learn

~~As~~In ~~a~~this ~~developer~~tutorial, ~~I~~you ~~need~~will learn how to h

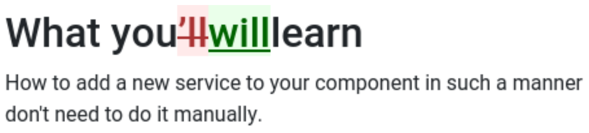
~~Context~~What you'll learn

~~As a developer~~How to have, ~~I need to have~~as a developer,secured

Figure 4.2: Comparison of output from the significant space mode (top) and the modified markdown-diff library (bottom)

Validity

The validity problem consisted of two parts: the <ins> and <del> html tags were either placed inside links, images or other HTML tags or caused the renderer to crash due to syntax errors. The MDX toolkit is especially susceptible to syntax errors - even having these tags placed across multiple lines of text causes errors. The first part of the problem was solved by utilizing regex expressions to fix the affected parts of the Markdown file, but some minor problems are effectively unsolvable by just these expressions. Below you will find an image of how a change of heading to paragraph is incorrectly rendered.

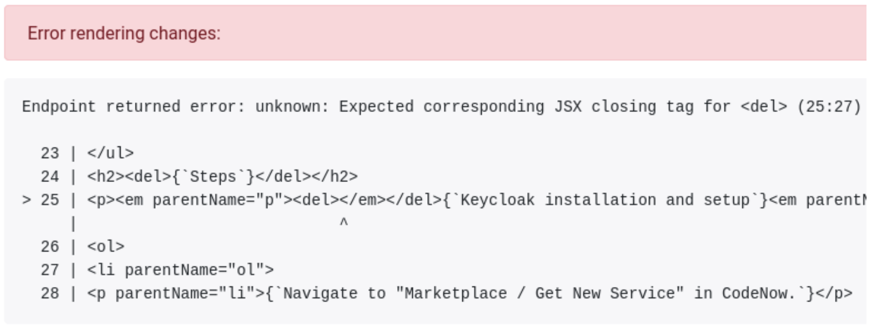


What you~~'ll~~willlearn

How to add a new service to your component in such a manner
don't need to do it manually.

Figure 4.3: Wrong output - the heading should be a paragraph as described above

The issue of the MDX syntax errors getting triggered by wrongly placed html tags is a bigger issue. Again, regex expressions were utilized, along with splitting tag content when a newline is encountered, but still, some of the issues remained unsolved as some are hard to debug or a very specific issue to the MDX renderer. Also, even though efforts were made to fix any broken HTML syntax, some invalid syntax is hard to detect and/or fix.



Error rendering changes:

```
Endpoint returned error: unknown: Expected corresponding JSX closing tag for <del> (25:27)

  23 | </ul>
  24 | <h2><del>{`Steps`}</del></h2>
> 25 | <p><em parentName="p"><del></em></del>{`Keycloak installation and setup`}<em parentN
     |                            ^
  26 | <ol>
  27 | <li parentName="ol">
  28 | <p parentName="li">{`Navigate to "Marketplace / Get New Service" in CodeNow.`}</p>
```

Figure 4.4: Hard to fix syntax error example

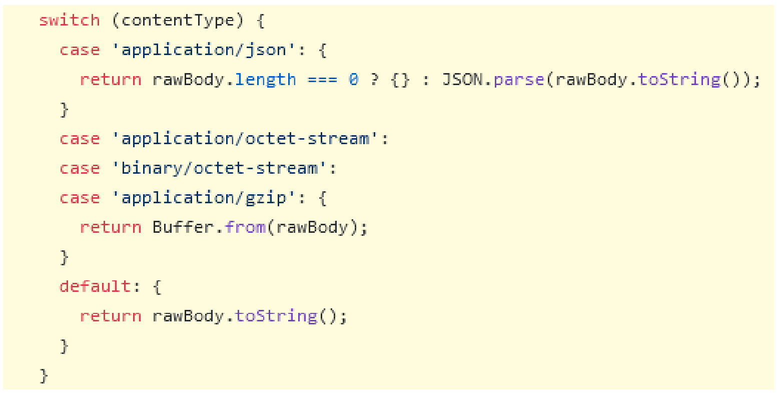
4.3.2 Gitbeaker

The Gitbeaker library had two issues, one of which I managed to fix. This includes blob file and error handling.

Blob handling

This was the same issue I encountered while prototyping. I was very curious why the issue was present, although the Gitlab API was utilized, therefore I searched for a bug in my implementation. The issue persisted when I tried writing the file directly to the hard drive, therefore I assumed the problem has to lie elsewhere and begun searching in the library source code.

Soon enough, the source of the problem was found. The blob file was being treated as a UTF string. Again. This time though, this was deliberate.



```
switch (contentType) {
  case 'application/json': {
    return rawBody.length === 0 ? {} : JSON.parse(rawBody.toString());
  }
  case 'application/octet-stream':
  case 'binary/octet-stream':
  case 'application/gzip': {
    return Buffer.from(rawBody);
  }
  default: {
    return rawBody.toString();
  }
}
```

Figure 4.5: The block of problematic code in Gitbeaker

The code above actually expects only 3 different mime types to be handled as raw binary data. I encountered this problem because I was using the library to fetch an `image/png` file type. The fix was rather obvious - that is to treat only `text` type as a string, regardless of its subtype and every other file type as raw binary file.

After I reported the issue, the developer of the library - Justin Dalrymple was quick to respond. After I created a fix for the issue and added appropriate tests, he quickly merged the pull request I made and published a new version of the library.

Error handling

The other problem encountered while working with the library was error handling. I am currently not sure why, but the HTTP status code returned by the Gitlab instance is inaccessible and the error codes provided by Gitbeaker are not very descriptive. If the HTTP status code was included, it would be a lot easier to provide meaningful messages to the user and more importantly - handle the errors appropriately.

4.3.3 Gitlab issues

Performance

Another interesting problem was encountering multiple strange performance issues while using the Gitlab API. Both of them were encountered, when requesting all data from a particular endpoint - more precisely the file structure and all commits in the repository. The issue with getting the full file tree was expected and an implementation using lazy loading of folders was created. The load time of the entire file tree was a lot shorter, when the folders were eagerly loaded, too.

But getting all commits from Gitlab was exceptionally slow - and that was only with around 250 commits at that! One would expect that this process would have been optimized a lot. But this request caused browser to timeout the fetch - with ky as the fetching library this is 10s in the default setting. When tested from my PC the commit information retrieval request alone took around 3.5 s. On a hunch, I tried to limit the number of returned commits to 100. Almost magically, the response time went down to 120 ms. Returning 100 commits takes 120ms and returning around 250 takes 3.5 s? Something does not add up. Therefore, setting the limit to 1000 was tried next and the 250 commits were returned in around 300 ms.

Repository API

Another problem with the API that was encountered during the development was that the Documentation changes could not be saved, which was only true for newly added admins, not the owner. The api actually returned the 403 Forbidden error code. There was no information about this issue on the internet and there was no obvious issue with the implementation, as it works for the repository owner. Also, there was no problem when tested in the Gitlab UI. Therefore, it is assumed that this is a bug in the API implementation.

4.3.4 Deployment

As the development was agile, it was split into smaller sprints. The application was deployed on the CodeNow platform at least once a sprint was finished, after the access to the platform was obtained around halfway through the development. This meant that the application needed to be modified to accommodate the workflow utilized by CodeNow.

Docker

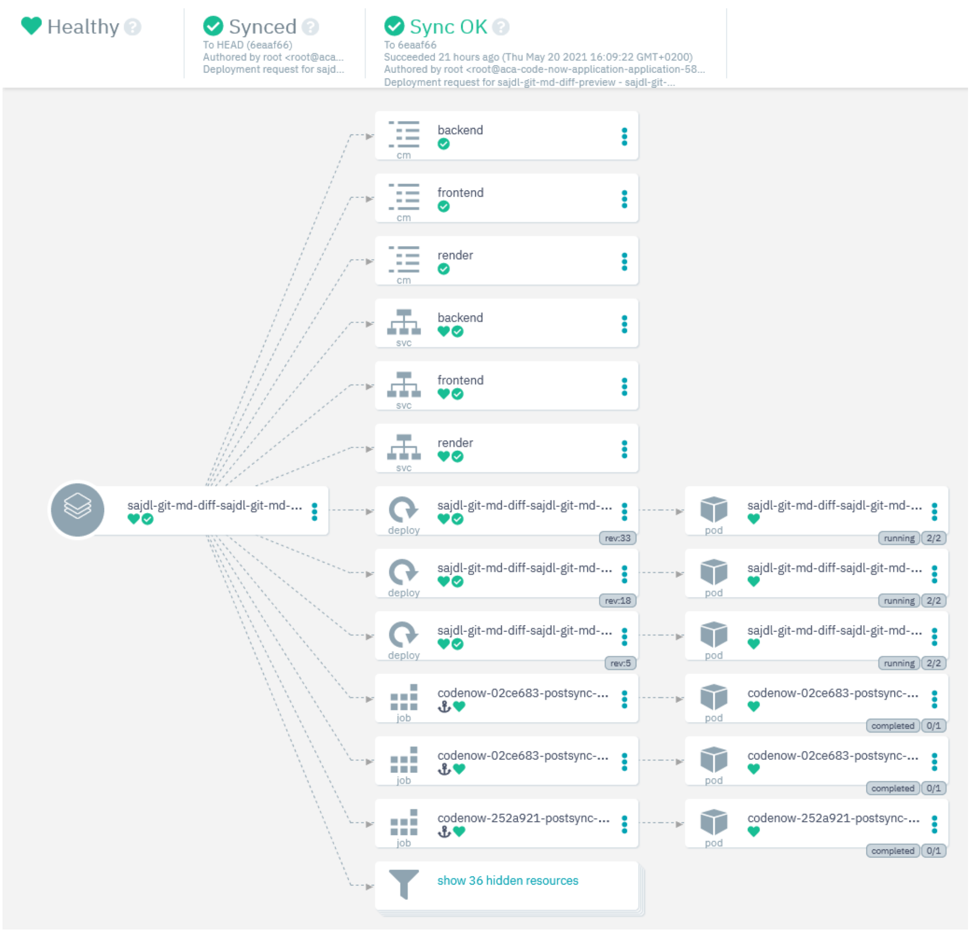
Docker is a widespread virtualization tool which utilizes containers to run application in their own environments. The environment is isolated from all other processes on the host machine. The approach Docker utilizes leverages kernel namespaces and cgroups, which are Linux kernel features [42].

Docker compose is a tool for defining and running multi-container Docker applications. With Docker compose, YAML files are used to configure the application services and connections between them. After writing this configuration, the application can be started up using a single command [43].

The implementation uses different Docker images for the local deployment and the deployment on CodeNow. Local deployment also utilizes the Docker compose tool to enable users to easily run all services, including the app services, database, pgAdmin and even the API documentation.

CodeNow

There were various problems when first deploying the application to the CodeNow platform. First, it is expected that the code will be hosted in separate repositories in the Gitlab instance provided by Stratox Enterprises s.r.o. This was an issue, because the development was fully in progress and the project was already hosted at GitHub. But still, this could be easily solved by utilizing the subfolders in the repository as separate GIT repositories, which made it relatively easy to use. The second problem was that the platform expects use of the provided scaffolders, which is something that was not expected, therefore the folder structure had to be modified a bit. At the time of writing, this was already remedied, and generic Docker images are now allowed.


Healthy

Synced
To HEAD (6eaaf66)
Authored by root <root@aca...
Deployment request for sajd...

Sync OK
To 6eaaf66
Succeeded 21 hours ago (Thu May 20 2021 16:09:22 GMT+0200)
Authored by root <root@aca-code-now-application-application-58...
Deployment request for sajdl-git-md-diff-preview - sajdl-git-...

backend
cm
frontend
cm
render
cm
backend
svc
frontend
svc
render
svc
sajdl-git-md-diff-sajdl-git-md-...
sajdl-git-md-diff-sajdl-git-md-...
deploy
rev:33
sajdl-git-md-diff-sajdl-git-md-...
pod
running
2/2
sajdl-git-md-diff-sajdl-git-md-...
deploy
rev:18
sajdl-git-md-diff-sajdl-git-md-...
pod
running
2/2
sajdl-git-md-diff-sajdl-git-md-...
deploy
rev:5
sajdl-git-md-diff-sajdl-git-md-...
pod
running
2/2
codenow-02ce683-postsync-...
job
codenow-02ce683-postsync-...
pod
completed
0/1
codenow-02ce683-postsync-...
job
codenow-02ce683-postsync-...
pod
completed
0/1
codenow-252a921-postsync-...
job
codenow-252a921-postsync-...
pod
completed
0/1
show 36 hidden resources

Figure 4.6: Codenow deployment overview

Chapter 5

Testing

5.1 User testing

During the writing of this thesis, the Coronavirus was rampant in the Czech Republic. So much so, that the government decided to impose district lockdown. This posed a problem in two areas: firstly, people were less willing to participate in the user testing and secondly, remote user testing tends to be more time consuming and for older, less technically able users often hard to do. It is also harder to assess the (dis)satisfaction of the user. For the beforementioned reasons, it was decided to include user behaviour tracking service in the app.

5.1.1 User behaviour tracking

User behaviour tracking tools, unlike normal analytics, track how the user interacts with the page. Several tools exist on the market with various features ranging from basic heatmap tracking to recordings and some even allow users to report bugs via a form embedded in the page. For this thesis, we considered two services - Hotjar and Smartlook.

Hotjar

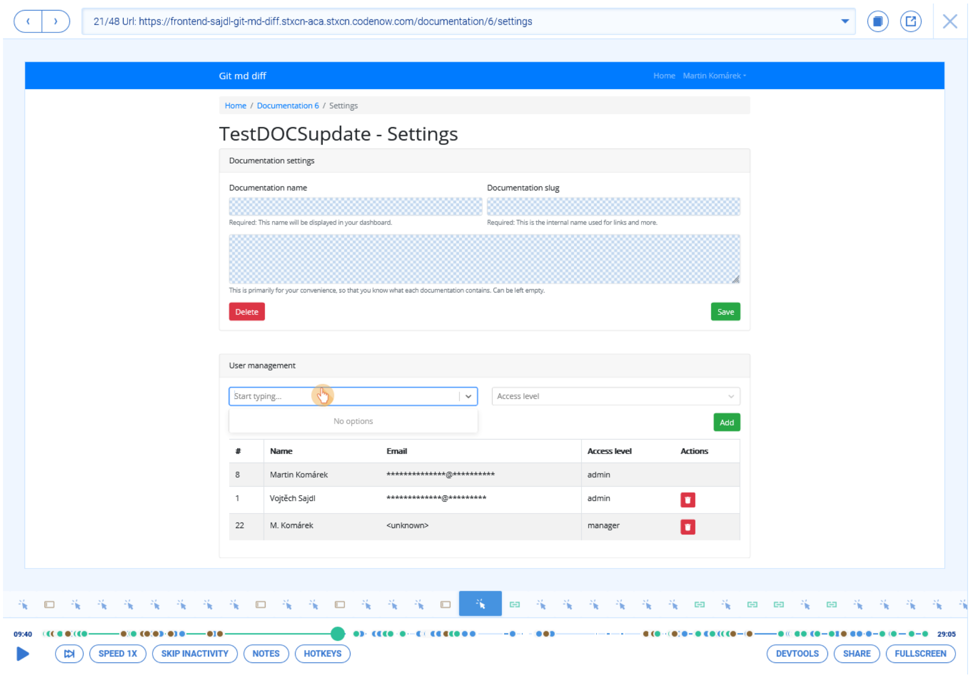
Hotjar is a behaviour analytics and user feedback service that helps companies understand the behaviour of your website users and get their feedback [44]. There are many success stories, including even some large companies such as Tomtom and Ryanair [45]. Hotjar is GDPR-ready and provides multiple useful features - from the standard heatmaps, and recordings to collecting and managing user feedback. It also has the option to survey users [44]. The implementation in React is simple with the `react-hotjar` npm package.

Smartlook

Smartlook is a direct competitor to Hotjar. It offers many of the features Hotjar supports and adds some of their own [46]. There are many success stories too - including the biggest Czech online store Alza.cz or the Kiwi.com online travel agency [47]. It is also GDPR ready but does not offer any features to collect user feedback. For React integration the `smartlook-client` package is provided.

Implementation & UI overview

Although both options were evaluated, due to its better pricing options, it has been decided to use Smartlook. In both cases the implementation is very simple as the basic implementation takes just about one line of code. Below you will find an example of Smartlook recording with automatic sensitive information censoring which is a part of their GDPR-ready design.


21/48 Url: https://frontend-sajdl-git-md-diff.stxcn-aca.stxcn.codenow.com/documentation/6/settings
Git md diff
Home
Martin Komárek
09:40
29:05
SPEED 1X
SKIP INACTIVITY
NOTES
HOTKEYS
DEVTOOLS
SHARE
FULLSCREEN

Home / Documentation 6 / Settings

TestDOCSupdate - Settings

Documentation settings

Documentation name

Required: This name will be displayed in your dashboard.

Documentation slug

Required: This is the internal name used for links and more.

This is primarily for your convenience, so that you know what each documentation contains. Can be left empty.

Delete Save

User management

Start typing... | Access level

No options

Add

#	Name	Email	Access level	Actions
8	Martin Komárek	**************@**********	admin	
1	Vojtěch Sajdl	*************@*********	admin	🟥
22	M. Komárek	<unknown>	manager	🟥

Figure 5.1: Smartlook recording

5.1.2 Methodology

According to Jakob Nielsen[1], there is no need to test usability with no more than 5 users. Furthermore, he goes as far as to claim that as soon data from a single test user are collected, it contains about $\frac{1}{3}$ of the information that there is to know about the usability of the design.

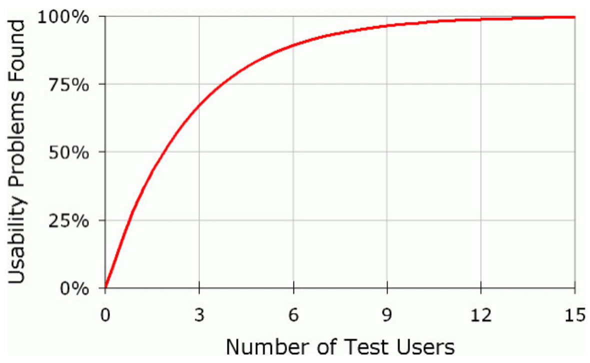


Figure 5.2: User testing and amount of total errors found [48]

Because of the coronavirus limitations there was only a limited number of users that wanted to participate and time to evaluate the application. Therefore 2 rounds of testing with 3 users were conducted.

Target audience

As the target audience a group of people between 20 and 60 years without any major disabilities were selected. The users were to be at least moderately technically inclined - the main factors included their knowledge of markdown or word processors and git knowledge. Users with little to no knowledge of GIT were preferred.

Tested scenarios

When the testing was planned, it was expected that there would be at least 5 participants split into 2 different user groups. First (the less technically inclined one) was meant to test the proof-reader use cases and the second one was meant to test the administration interface. But because of the unexpectedly long government regulation of the freedom of movement, the

[1] Jakob Nielsen is an web usability consultant and human-computer interaction researcher, has invented some of the usability methods that are widely used today, including heuristic evaluation.

test group was missing 2 people who were meant to mainly test the usability of the editor. Therefore, it was decided to include all the remaining users in both types of tests. In the end the scenario was split into three parts - basic documentation administration testing, file view/edit testing and proofreading process testing. Therefore, the user was first presented with the scenario, outlined in points below:

- Login using Gitlab
- Create a new documentation
- Add a specific user to the documentation
- Change the documentation information (any of the name, slug or description)
- Remove a user from the documentation

After completing the first test the user was added to an existing documentation which was prepared with existing snapshot of CodeNow documentation. The testing then continued as outlined below:

- Edit a file (e.g., make some text bolder or change the heading level) and save it
- Edit a file and "accidentally" close the tab. Then try to edit the file again.
- Create a folder or file
- Delete a file

Now the testing continued with the proofreading process evaluation:

- Compare two revisions of the documentation
- Create a proofreading request from this comparison
  - user was asked to exclude one specific file
  - while user was completing this step, a proofreading request was assigned to them
- Check whether any new requests were assigned to you
- Complete the assigned request

  - while user was completing this step, the proofreading request they created was completed
- If any requests are finished, accept them

The average time to complete the entire testing scenario was about 65 minutes, with the second test being significantly shorter, as it was primarily focused on reviewing the changes made to the application.

5.1.3 Results

During the testing, various problems were encountered. Some of them were bugs, but there were also some major oversights in the original design, although the fixes were usually fairly easy to implement. Below you will find summary of some of the most important problem that were discovered, that were not caused by bugs.

User 1

User 1 was currently studying Open Informatics bachelor programme. They had some basic understanding of GIT and a good understanding of the Markdown language. They had their own Gitlab account.

- Large priority problems
  - Cursor does not indicate that a documentation on the dashboard is clickable
  - Cannot easily tell which revision is earlier, date would help
  - Proofreading request - warning alert "Merge not yet possible" confusing when shown right after creating the request
  - Buttons not hiding when merged/submitted changes
  - Option to go back should be added, user feared the changes would disappear should they press the back button
- Medium priority problems
  - State of the request should be more easily distinguishable
  - Proofreading request description - not sure where is displayed, what to write
  - Allow switching "My changes" and "Diff changes" in proofreading
  - No invisible changes badge is confusing

- Low priority problems
  - Link to users GitLab account
  - Indicate a folder has editable elements

User 2

User 2 was currently studying Open Informatics master's programme. They had good understanding of GIT and a good understanding of the Markdown language. They had their own Gitlab account.

- Large priority problems
  - A documentation of the application functions should be available
  - Access levels should be described
  - Proofreading requests should contain information about which documentation they belong to
  - It should be possible to change user access level without their removal
  - Proofreading request should include contact for the proof-reader & requester
  - Why some files cannot be opened?
- Medium priority problems
  - Highlight required fields
  - Unknown email - unclear what it means
  - File badges are confusing
  - Commit message in commit select is too short
- Low priority problems
  - User search requires diacritics
  - Some elements change position when alert disappears

We can see that some problems start repeating at this point. Also, some problems that were identified are constrained by API limitations and therefore cannot be fixed, for example the user search User 2 mentioned.

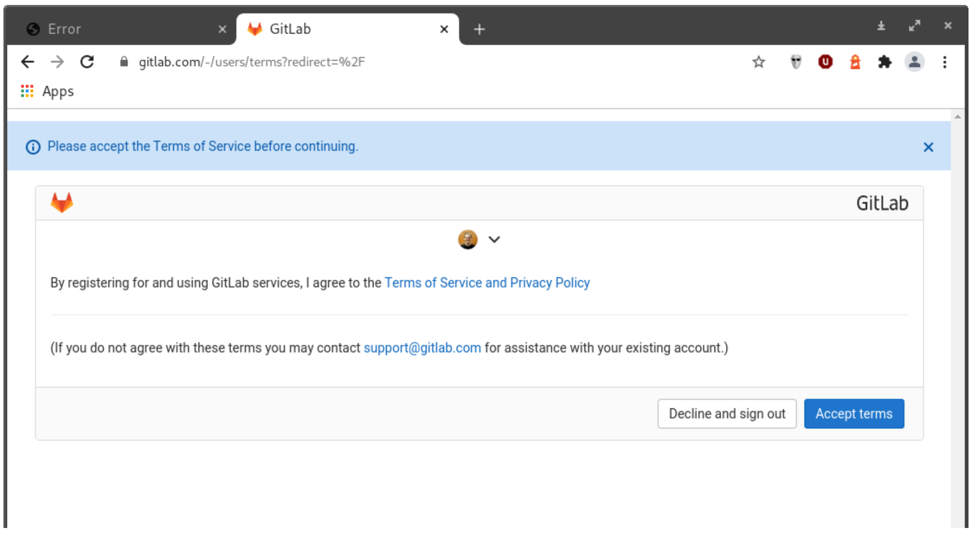
User 3

User 3 was an employee of Stratox Enterprises s.r.o. They had good understanding of GIT and a good understanding of the Markdown language. They had their own Gitlab account, but it was unusable due to a problem that was encountered. This will be described below.

- Blocker problems
  - Login not working
- Large priority problems
  - We should be able to write commit message
  - Commits are not limited and it is not easy to know what is "from" and what is "to"
  - We should have some info about the user levels present
  - We should send out emails when user is added / proofreading req. state changed
  - Edit button should be present, not only clickable title
  - We shouldn't allow user to navigate away from the editor if changes were made
- Medium priority problems
  - If file did not change, do not allow saving
  - Detect whether branch is protected and we don't have access
  - Select all files / deselect all files
  - File badges confusing - add tooltips
- Low priority problems
  - Proof-reader should see differences by default, not "Preview"
  - Mark as complete button could be named Proofreading finished
  - Start typing when searching user might not be clear
  - Duplicate file could be added - useful so that we do not have to start from a blank file
  - Internal links should work
  - Green colour of newly added file did not seem correct
  - Autosave diff would be nice to have

This user was the first one to encounter any problems while logging in, which was confusing, because it worked for all previous users with me included. Therefore, it seemed unlikely to be problem with the implementation and a bug in the underlying library was assumed. I tried to reproduce the issue with existing accounts without success, therefore the same procedure used by the user to log into their account was used.

As the user was not sure whether the account has previously existed under the same Google account login method, new account using the same method was created, then the bug was successfully reproduced. After some investigation along with forking the library and updating its dependencies, it seemed like the library was in a good working order. The next step was to check the account settings to look for any abnormalities. It was then when the root cause was discovered.



Please accept the Terms of Service before continuing.

GitLab

By registering for and using GitLab services, I agree to the Terms of Service and Privacy Policy

(If you do not agree with these terms you may contact support@gitlab.com for assistance with your existing account.)

Figure 5.3: The root cause of the login error

It was found out, that although the OAuth process continued successfully, the account was in fact not yet active. That caused the application to not be able to retrieve the user's data. Unfortunately, there is no way for us to fix or detect this problem.

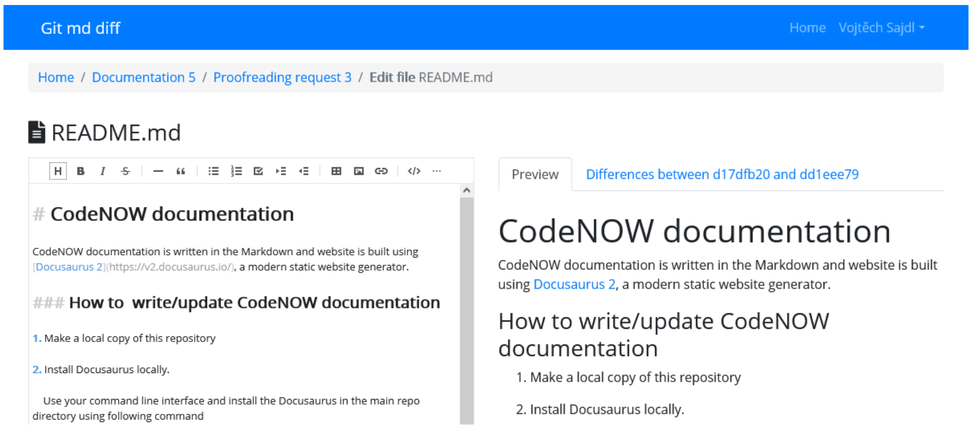
5.1.4 Implementation changes after user testing

As the result, many changes to the user interface were implemented. Some of the more interesting ones are highlighted below.

Back navigation

After the first user testing, the need for a non-browser based back navigation was identified. This might have been implemented in many different ways - for example as a back button. Using heuristic evaluation, it was recognized that this implementation might not be ideal as the problem of remembering the navigation sequence would be necessary.

This might be a problem when the user focuses on writing a new documentation page or proofreading an existing one. Therefore, it was decided to implement this in the way of breadcrumbs. This approach is also beneficial when a large navigation sequence led to the resulting page, the user can then decide to skip a few steps. This case is depicted below.


Git md diff
Home
Vojtěch Sajdl
Home / Documentation 5 / Proofreading request 3 / Edit file README.md
README.md
Preview
Differences between d17dfb20 and dd1eee79

```
# CodeNOW documentation

CodeNOW documentation is written in the Markdown and website is built using
[Docusaurus 2](https://v2.docusaurus.io/), a modern static website generator.

### How to  write/update CodeNOW documentation

1. Make a local copy of this repository

2. Install Docusaurus locally.

  Use your command line interface and install the Docusaurus in the main repo
directory using following command
```

CodeNOW documentation

CodeNOW documentation is written in the Markdown and website is built using Docusaurus 2, a modern static website generator.

How to write/update CodeNOW documentation

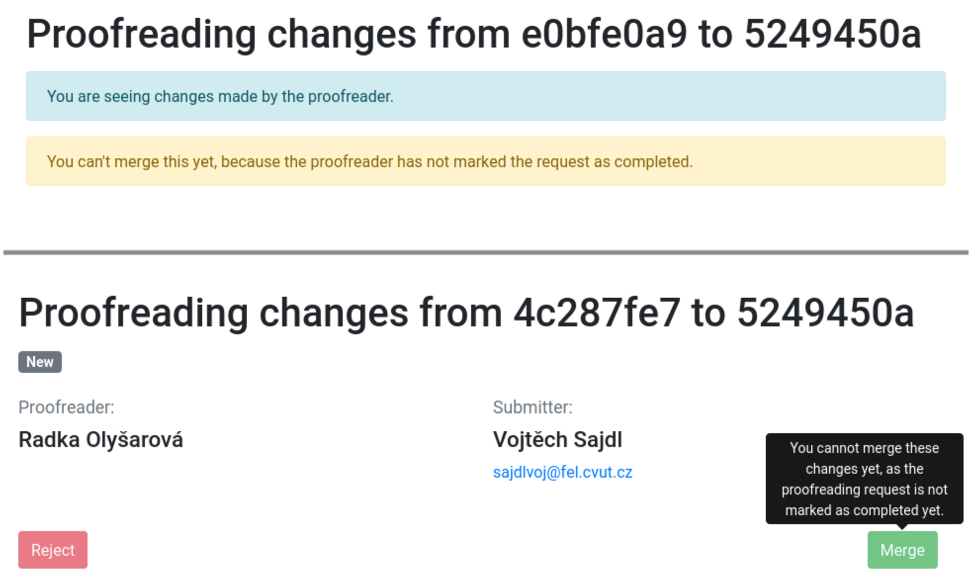
1. Make a local copy of this repository
2. Install Docusaurus locally.

Figure 5.4: Breadcrumbs implementation on the proofreading edit page

Still one problem with the implementation was discovered - which was that the inclusion of the IDs instead of the proofreading request or documentation name were confusing to one of the users. This problem was fixed only partially due to time constraints.

Information display

In a few cases, the user testing revealed a problem with how the information is being displayed. This ranged from "I am not sure what x means" to "I felt I've done something wrong". This was the main case when the proofreading request was created. To solve this, a tooltip was introduced on the element that it was referring to. Below you can see a comparison of a new and previously used implementation.



Proofreading changes from e0bfe0a9 to 5249450a

You are seeing changes made by the proofreader.

You can't merge this yet, because the proofreader has not marked the request as completed.

Proofreading changes from 4c287fe7 to 5249450a

New

Proofreader:
Radka Olyšarová

Submitter:
Vojtěch Sajdl
sajdlvoj@fel.cvut.cz

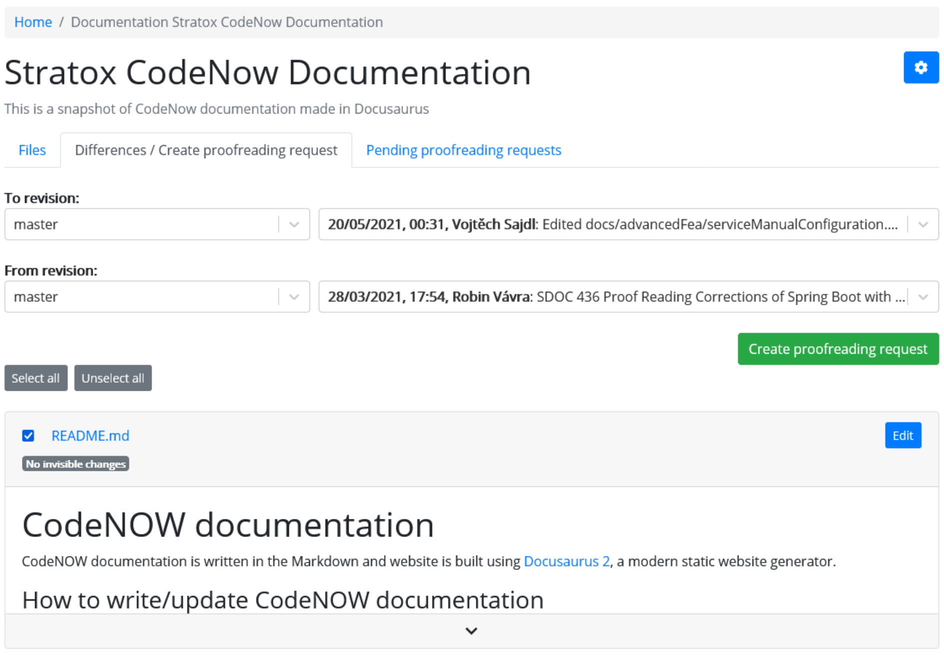
You cannot merge these changes yet, as the proofreading request is not marked as completed yet.

Figure 5.5: Comparison of information display in proofreading request - old at the top, new at the bottom

Notice that more information was added to the proofreading request too. Originally this information was omitted, which was found to be a bad idea from the user testing.

Documentation display

The documentation page underwent a complete overhaul, as problems with the original design were discovered. The original design was implemented to match the original wireframe almost exactly. The new implementation can be seen below.


Home / Documentation Stratox CodeNow Documentation

Stratox CodeNow Documentation

This is a snapshot of CodeNow documentation made in Docusaurus

Files
Differences / Create proofreading request
Pending proofreading requests

To revision: master — 20/05/2021, 00:31, Vojtěch Sajdl: Edited docs/advancedFea/serviceManualConfiguration....

From revision: master — 28/03/2021, 17:54, Robin Vávra: SDOC 436 Proof Reading Corrections of Spring Boot with ...

Create proofreading request
Select all
Unselect all
README.md
No invisible changes
Edit

CodeNOW documentation

CodeNOW documentation is written in the Markdown and website is built using Docusaurus 2, a modern static website generator.

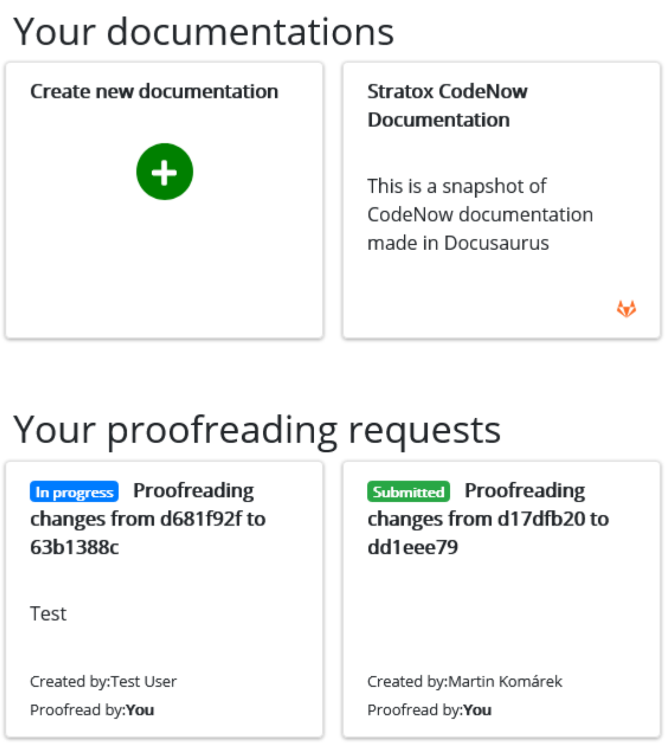
How to write/update CodeNOW documentation

Figure 5.6: The overhauled documentation page

This implementation added the edit button to each document, the badges moved to the top of the document card and now the document is collapsed by default and can be expanded by clicking at the arrow at the bottom. This solves the problem where, if many files were modified at once, the page would be very long and cluttered.

Dashboard

The dashboard was implemented almost exactly as the original wireframe intended. The most significant change is the inclusion of the state badges, as the states were written in the header previously and were not very recognizable. This can be seen on the image below.


Your documentations

Create new documentation

Stratox CodeNow Documentation

This is a snapshot of CodeNow documentation made in Docusaurus

Your proofreading requests

In progress Proofreading changes from d681f92f to 63b1388c

Test

Created by:Test User
Proofread by:**You**

Submitted Proofreading changes from d17dfb20 to dd1eee79

Created by:Martin Komárek
Proofread by:**You**

Figure 5.7: Dashboard implementation

5.2 Automated tests

Automated software testing is an important part of any production-ready application. It helps in preventing introduction of bugs or security vulnerabilities in the code. It is even more important, when the codebase becomes large and when more developers are working at the same project as the tests often check the code behaviour. Automated tests can be split into two parts: static and dynamic tests, both of which will be described in this section.

5.2.1 Static analysis

Static analysis, as the name suggests, does not actually run the code. The methods utilize semantic analysis methods such as type checking and vulnerability signature matching. These tools can often reveal a bug in the code even before any tests are written.

Lint tools

Lint tools specialize on code style enforcement, but they often go beyond that. Most of the tools are code-aware and therefore can detect bad syntax, use of undefined variables and much more. Some even offer the option to automatically fix the style issues. These tools mostly eliminate problems that for humans would be hard to spot, such as writing `=` instead of `==` in a condition.

Vulnerability detection

Another important part of static code analysis is vulnerability detection. These tools have a great database of code patterns that introduce vulnerabilities in the code or can cause the application to overload or behave unexpectedly. They can detect pretty much anything from Cross Site Scripting, to missing rate limiting or even the use of vulnerable version of packages.

5.2.2 Dynamic testing

Dynamic testing actually runs the code and can check its results. The dynamic testing methods can be split into multiple categories. Each of them has their own use-cases and the complexity of implementation and runtime length also differs a lot. Moreover, different test types can discover different category of bugs. Below you will find a quick summary of the dynamic testing techniques.

Unit testing

Unit tests are the most used type of tests. The aim is to test every unit separately - therefore if it depends on a different unit, it needs to be mocked. The unit tests should be minimal and therefore fast and will always identify the exact point of failure in the application. Unit testing also allows for refactoring and even re-implementing parts of code with confidence, because the output of the modified component is tested [49], [50].

Integration testing

Integration testing is a level of software testing where individual units are combined and tested as a group [51]. It is performed to expose defects in the interactions between individual units. As this kind of testing often takes up more time than unit testing, test cases and data should be more carefully chosen.

End to end testing

End to end testing is the most complex one as it simulates the end user. It tests the application as a whole, utilizing a defined scenario. They rely on the whole app; therefore, they cannot identify the point of failure and can be flaky - for example, a 3rd party service might fail temporarily during testing and therefore yielding wrong test results.

The recommended split

From the quick summary above, it is apparent that each of the methods has its advantages and disadvantages. According to the Google Testing blog article written by Mike Wacker, end to end tests are overly complex and often do not provide enough insight [52]. Google often suggests a 70/20/10 split: 70% unit tests, 20% integration tests, and 10% end-to-end tests [52] - this roughly translates into the testing pyramid depicted below.

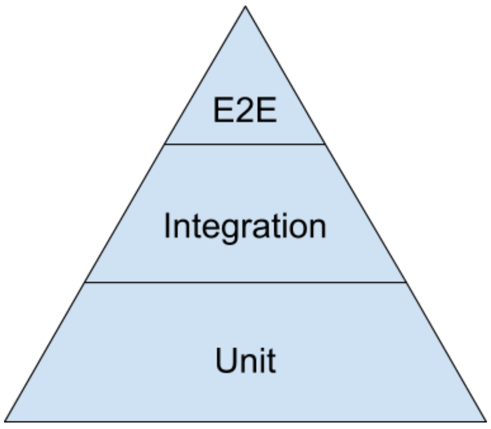


Figure 5.8: Testing pyramid [52]

5.2.3 Implementation

In this thesis, we implemented our own solution as well as modified some of the libraries it uses. For those libraries, most modifications warranted writing new unit tests. For example, for the `markdown-diff` library, unit tests trying to capture minimal test cases encountered in the wild were written. In the `Gitbeaker` library, minimal test cases for the encountered bug were written.

The API relies mainly on static analysis and basic unit tests right now, as due to setbacks during the development, there was not enough time to write meaningful integration tests. The implemented unit tests were written to cover the most basic cases - such as verifying that a new object is created properly using the constructor parameters. More importantly the frontend part remains completely untested from React component standpoint and relies on static analysis only. The Redux reducers and actions have some basic test cases written. Also, in both the frontend and backend, the immutability of the enums is tested, as well as their values.

Chapter 6

Conclusion

6.1 Achieved results

The result of this thesis is a fully working, user tested application that is deployed in a CodeNow academy environment. As the application supports custom Gitlab instances, it is currently planned to start real-world testing of the application by utilizing it in the CodeNow documentation management process at Stratox Enterprises s.r.o.

User testing discovered some major usability problems, most of which are already fixed. The application should be mostly production-ready, but due to the smaller than expected user testing group, some usability issues might still arise. There are also some known areas that would deserve further improvements.

6.2 Further improvements

There are some areas that are known to require additional improvements, that unfortunately were not incorporated during this thesis. These will be described below.

6.2.1 Diff algorithm

The diff algorithm underwent a lot of changes during the development. Over the time it became clear, that even though for most cases, regex fixing is enough, there are some unfixable problems as mentioned before. These problems would most likely require a different approach to the entire diff algorithm. One of those approaches could be using an AST diff algorithm.

Gumtree algorithm

Gumtree algorithm is an AST diff algorithm, inspired by the way programmers search in the code, aims to create diffs that are close to capturing the developer's intent [53]. Using such algorithm would resolve all the syntax problems caused by the existing libraries. Libraries such as JSDiff could still be utilized to highlight changes inside the AST blocks.

6.2.2 Various UI and UX improvements

There also were some good UX improvements that unfortunately could not be included. Some of them are relatively easy, such as excluding users that have already access to the documentation. Some are minor annoyances like the one when on reload, the active tab is not remembered. And there were also some functionality improvements - mainly file duplication or making the internal links in markdown files work.

Also something many might not consider an UX improvement, but improvement nonetheless - the quality of the emails. Right now, the emailing is relatively barebones and a better worded emails including more information would be great for users.

6.2.3 Better test coverage

As of now, the API server is partially covered by the test suite and the frontend is not covered at all, except for the Redux tests. This is a big room for improvement, because right now a bug in the React app could cause users to be unable to complete some tasks.

6.2.4 Database migration

Another important improvement to implement would be database migrations. Until now all the migrations were either done manually or by dropping the table and recreating it, as the application was not running in a production environment. This will not be possible when the application is deployed and used by real users. Thus, implementing a database migration library such as `ley` would be highly beneficial.

Bibliography

[1] C. J. Stettina and W. Heijstek, "Necessary and neglected?: An empirical study of internal documentation in agile software development teams", in *Proceedings of the 29th ACM international conference on Design of communication - SIGDOC '11*, ACM Press, 2011, p. 159, ISBN: 9781450309363. DOI: 10.1145/2038476.2038509. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2038476.2038509.

[2] ClickHelp LLC, *What is software documentation?* [Online]. Available: https://clickhelp.com/software-documentation-glossary/software-documentation/.

[3] J. Gruber, *Markdown: Syntax*. [Online]. Available: https://daringfireball.net/projects/markdown/syntax.

[4] GitLab, *Permissions*. [Online]. Available: https://docs.gitlab.com/ee/user/permissions.html.

[5] Facebook Inc., *Docusaurus: Introduction*. [Online]. Available: https://v2.docusaurus.io/docs/.

[6] M. Cone, *An overview of markdown, how it works, and what you can do with it*. [Online]. Available: https://www.markdownguide.org/getting-started/.

[7] ——, *Applications and components that support markdown*. [Online]. Available: https://www.markdownguide.org/tools/.

[8] CommonMark, *Commonmark: A strongly defined, highly compatible specification of markdown*. [Online]. Available: https://commonmark.org/.

[9] ——, *Markdown flavors*. [Online]. Available: https://github.com/commonmark/commonmark-spec/wiki/Markdown-Flavors.

[10] MDX, *Mdx*. [Online]. Available: https://mdxjs.com/.

[11] Remark, *Readme*. [Online]. Available: https://github.com/remarkjs/remark/blob/main/readme.md.

[12] Marked, *Marked documentation.* [Online]. Available: https://marked.js.org.

[13] E. Jacobs, *Markdown-to-jsx.* [Online]. Available: https://github.com/probablyup/markdown-to-jsx.

[14] npm trends, *Remark-parse vs marked vs markdown-to-jsx.* [Online]. Available: https://www.npmtrends.com/remark-parse-vs-marked-vs-markdown-to-jsx.

[15] *Javascript | mdn.* [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[16] *Desktop browser market share worldwide.* [Online]. Available: https://gs.statcounter.com/browser-market-share/desktop/worldwide/2020.

[17] J. Zaytsev, *Ecmascript 2016+ compatibility table.* [Online]. Available: https://kangax.github.io/compat-table/es2016plus/.

[18] M. T. Thomas, *React in action.* Manning Publications, 2018, ISBN: 9781617293856.

[19] Facebook Inc., *React.* [Online]. Available: https://reactjs.org/.

[20] ——, *Introducing jsx.* [Online]. Available: https://reactjs.org/docs/introducing-jsx.html.

[21] S. Daityari, *Angular vs react vs vue: Which framework to choose in 2021.* [Online]. Available: https://www.codeinwp.com/blog/angular-vs-vue-vs-react.

[22] *Redux - a predictable state container for javascript apps.* [Online]. Available: https://redux.js.org/.

[23] D. Bugl, *Learning Redux: write maintainable, consistent, and easy to-test web applications.* O'Reilly Media, 2017, ISBN: 9781786469533. [Online]. Available: https://proquest.safaribooksmisc.com/9781786462398.

[24] T. U. Consortium, *The unicode standard, version 12.1.0*, 2019. [Online]. Available: http://www.unicode.org/versions/Unicode12.1.0/.

[25] S. Hsu, *Session vs token based authentication*, Jul. 2018. [Online]. Available: https://sherryhsu.medium.com/session-vs-token-based-authentication-11a6c5ac45e4.

[26] S. Peyrott, *JWT Handbook*, 0.14.1. Auth0® Inc.

[27] B. Pontarelli, *Pros and cons of jwts.* [Online]. Available: https://fusionauth.io/learn/expert-advice/tokens/pros-and-cons-of-jwts/.

[28] S. E. s.r.o., *Codenow.* [Online]. Available: https://www.codenow.com/.

[29] *Microservices vs monolith.* [Online]. Available: https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/.

[30] Node.js, *Node.js.* [Online]. Available: https://nodejs.org/en/.

[31] ——, *About.* [Online]. Available: https://nodejs.org/en/about/.

[32] F. Copes, *The NodeJS handbook.*

[33] *Express - node.js web application framework.* [Online]. Available: https://expressjs.com/.

[34] ——, *The Express handbook.*

[35] *Sql vs orms vs query builders | compare.* [Online]. Available: https://www.prisma.io/dataguide/types/relational/comparing-sql-query-builders-and-orms.

[36] R. Porsager, *porsager/postgres-benchmarks.* Apr. 2021. [Online]. Available: https://github.com/porsager/postgres-benchmarks.

[37] *What is ssl, tls and https? | digicert.* [Online]. Available: https://www.websecurity.digicert.com/security-topics/what-is-ssl-tls-https.

[38] T. Lowdermilk, *User-centered design: a developer's guide to building user-friendly applications*, First edition. O'Reilly, 2013, ISBN: 9781449359805.

[39] E. Canziba, *Hands-on UX design for developers: design, prototype, and implement compelling user experiences from scratch.* Packt Publishing, 2018, ISBN: 9781788626699.

[40] J. T. Mark Otto and B. contributors, *About.* [Online]. Available: https://getbootstrap.com/docs/4.1/about/overview/.

[41] S. Sorhus, *sindresorhus/ky.* May 2021. [Online]. Available: https://github.com/sindresorhus/ky.

[42] Docker Inc., *Orientation and setup*, May 2021. [Online]. Available: https://docs.docker.com/get-started/.

[43] ——, *Overview of docker compose*, May 2021. [Online]. Available: https://docs.docker.com/compose/.

[44] Hotjar, *What is hotjar?* [Online]. Available: https://www.hotjar.com/blog/what-is-hotjar/.

[45] ——, *Customer stories and case studies.* [Online]. Available: https://www.hotjar.com/customers/.

[46] Smartlook, *Free website analytics tool.* [Online]. Available: https://www.smartlook.com/website-analytics/.

[47] ——, *Customer stories archives.* [Online]. Available: https://www.smartlook.com/blog/category/customer-stories/.

[48] J. Nielsen, *Why you only need to test with 5 users.* [Online]. Available: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

[49] *Unit testing*, Feb. 2011. [Online]. Available: https://softwaretestingfundamentals.com/unit-testing/.

[50] umer, *Unit, integration and end-to-end tests - finding the right balance*, Jul. 2016. [Online]. Available: `https://codeahoy.com/2016/07/05/unit-integration-and-end-to-end-tests-finding-the-right-balance/`.

[51] *Integration testing*, Mar. 2011. [Online]. Available: `https://softwaretestingfundamentals.com/integration-testing/`.

[52] *Just say no to more end-to-end tests*. [Online]. Available: `https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html`.

[53] J.-R. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus, "Fine-grained and accurate source code differencing", in *ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014*, 2014, pp. 313–324. DOI: `10.1145/2642937.2642982`. [Online]. Available: `http://doi.acm.org/10.1145/2642937.2642982`.

Appendix A

CD

The attached CD contains:

- This document in the PDF file format
- The application source codes
  - The `backend` folder contains the API microservice
  - The `frontend` folder contains the React frontend
  - The `render` folder contains the MXD render microservice
  - The `api.yaml` file contains the API endpoint documentation
  - The `docker-compose.yml` contains configuration for running the application locally using docker-compose
- The modified `markdown-diff` library

The source codes of the main application are also available at `https://github.com/Pryx/git-md-diff` and of the modified markdown-diff library at `https://github.com/Pryx/markdown-diff`.

Not included are the modifications made to the `Gitbeaker` library, which are already present upstream.