



Assignment of master's thesis

| | |
|---------------------------------|--|
| Title: | Redmine Helpdesk Web Application |
| Student: | Bc. Jakub Lukačín |
| Supervisor: | Ing. Oldřich Malec |
| Study program: | Informatics |
| Branch / specialization: | Software Engineering |
| Department: | Department of Software Engineering |
| Validity: | until the end of summer semester 2025/2026 |

Instructions

The goal of this thesis is to design and implement a web application that will serve as a helpdesk portal for Redmine, a highly customizable project management and issue-tracking tool. The primary objective of the application is to deliver a simplified user interface optimized for users without technical expertise.

Tailor your solution towards the assignee's organization's requirements, but consider interoperability in the sense of possible future distribution to third-party Redmine instances.

Undertake the following steps:

- Gather the user requirements.
- Analyze existing competition software.
- Design your solution.
- Analyze possible technical approaches and choose the most suitable one.
- Implement a functioning prototype of the application.
- Deploy the prototype, perform its user testing, and evaluate the testing.
- Remark on possible future improvements.



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Master's thesis

Redmine Helpdesk Web Application

Bc. Jakub Lukačín

Department of Software Engineering
Supervisor: Ing. Oldřich Malec

May 9, 2024

Acknowledgements

I would like to thank my supervisor, Ing. Oldřich Malec, for his time and valuable advice. I would also like to thank my colleagues who assisted me with the thesis. Last but not least, I'd like to thank my family and my friends for their support during my studies.

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.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 9, 2024

Czech Technical University in Prague

Faculty of Information Technology

© 2024 Jakub Lukačín. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Lukačín, Jakub. *Redmine Helpdesk Web Application*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Abstrakt

Táto práca sa zaoberá designom a implementáciou helpdesku. Tento helpdesk má pracovať so systémom Redmine, ktorý sa zameriava na riadenie projektov. Začiatok práce predstavuje software Redmine a termín helpdesk. Táto časť je nasledovaná popisom užívateľských požiadavkov a technológií použitých pri implementácii. Medzi použité technológie patrí Express.JS, PostgreSQL, framework Vue a knižnica komponent Vuetify. Kapitola Design je nasledovaná kapitolou Implementácia, kde je priblížených viacero zaujímavých oblastí implementácie, ako napríklad autentifikácia, práca so súbormi a internacionalizácia aplikácie. Nasadenie prototypu aplikácie je diskutované pri konci práce a využíva technológiu kontajnerizácie. Predposledná kapitola práce približuje prevedené užívateľské testovanie a posledná kapitola je venovaná budúcnosti projektu.

Kľúčová slova Redmine, systém pro řízení projektů, helpdesk, Vue, Vuetify

Abstract

This thesis focuses on the design and implementation of a help desk for a project management tool called Redmine. The beginning of the thesis introduces Redmine software and state-of-the-art in the help desk area. Afterward, the user requirements and technological stack of the application are presented. The main technologies are the Vue framework, Vuetify component library, Express.JS, and PostgreSQL. The design chapter is followed by the implementation, where several interesting areas, such as authentication, working with files, and internationalization are discussed. The deployment of the prototype is done with the help of containerization and is talked about near the end of the thesis. The deployment is followed by a chapter on user testing and the last chapter of the thesis talks about the possible future of the project.

Keywords Redmine, project management tool, helpdesk, Vue, Vuetify

Contents

| | |
|---|-----------|
| Introduction | 1 |
| 1 State-of-the-art | 3 |
| 1.1 An Overview of Redmine | 3 |
| 1.1.1 Who uses Redmine | 3 |
| 1.1.2 Features of Redmine | 4 |
| 1.1.3 Roles in Redmine | 5 |
| 1.1.4 Issue tracking feature of Redmine | 6 |
| 1.1.5 Summary of the Redmine overview subsection | 10 |
| 1.2 What is a help desk | 10 |
| 1.3 Competing helpdesk solutions for Redmine | 11 |
| 1.3.1 RedmineUP's Helpdesk Plugin | 11 |
| 1.3.2 Lightweight helpdesk plugin for redmine | 12 |
| 1.3.3 HelpDesk for Easy Redmine | 13 |
| 1.4 Papers on Helpdesk with a similar use case | 14 |
| 1.4.1 Colorado State University case study | 14 |
| 1.4.2 Oregon State University case study | 15 |
| 1.5 Competing helpdesk solutions outside Redmine | 17 |
| 1.5.1 Zoho Desk | 17 |
| 1.5.2 Jira Service Desk | 18 |
| 1.5.3 Spiceworks help desk | 20 |
| 1.6 A brief outline of the state-of-the-art chapter | 20 |
| 2 User requirements gathering | 23 |
| 2.1 Background of the topic | 23 |
| 2.2 Theory on requirements | 23 |
| 2.3 Functional requirements | 24 |
| 2.4 Non-functional requirements | 25 |
| 2.5 A discussion with a colleague | 25 |
| 3 Technological stack | 27 |
| 3.1 Choosing the right technology for the application | 27 |
| 3.2 Client-side JavaScript frameworks introduction | 28 |
| 3.2.1 A brief history of client-side JS frameworks | 28 |
| 3.2.2 Why do the frameworks exist? | 28 |

| | | |
|----------|--|-----------|
| 3.3 | Client-side JavaScript framework choice | 30 |
| 3.3.1 | React | 30 |
| 3.3.2 | Vue | 30 |
| 3.3.3 | Svelte | 30 |
| 3.3.4 | Comparison of the frameworks | 31 |
| 3.3.5 | The choice of the framework | 31 |
| 3.4 | Technologies bundle choice | 32 |
| 3.4.1 | Composition API vs Options API | 32 |
| 3.4.2 | Single File Component | 34 |
| 3.4.3 | The script setup | 34 |
| 3.4.4 | TypeScript or JavaScript | 34 |
| 3.4.5 | UI Component Library choice | 35 |
| 3.4.6 | Package manager choice | 35 |
| 3.5 | Conclusion of the technology choices | 36 |
| 4 | Design | 39 |
| 4.1 | Communication between helpdesk and Redmine | 39 |
| 4.1.1 | Authentication | 39 |
| 4.1.2 | Handling the API key | 39 |
| 4.1.3 | A remark on the OAuth 2.0 | 41 |
| 4.2 | Choosing the backend technology | 41 |
| 4.3 | UI design | 42 |
| 4.3.1 | Homepage and the layout of the application | 42 |
| 4.3.2 | Report an issue screen | 43 |
| 4.4 | The configurable fields | 45 |
| 4.5 | Handling the Helpdesk Configuration | 46 |
| 4.5.1 | Phase 1 - prototype deployment | 46 |
| 4.5.2 | Phase 2 - Redmine plugin | 46 |
| 4.5.3 | Phase 3 - Extending the Redmine plugin | 47 |
| 5 | Implementation | 49 |
| 5.1 | Start of the implementation | 49 |
| 5.1.1 | Initializing the frontend | 49 |
| 5.1.2 | Initializing the backend | 50 |
| 5.1.3 | Initializing the database | 50 |
| 5.1.4 | A remark on the IDE choice | 50 |
| 5.2 | Authentication | 51 |
| 5.2.1 | Registration | 51 |
| 5.2.2 | Login | 52 |
| 5.2.3 | Using the JWT in communication | 52 |
| 5.2.4 | Refreshing the JWT token | 53 |
| 5.3 | Formatting text in Redmine | 54 |
| 5.3.1 | Generating textile | 55 |
| 5.3.2 | Rendering textile | 56 |
| 5.4 | Requesting files from Redmine | 56 |
| 5.5 | Uploading files to Redmine | 57 |
| 5.6 | Handling the images in textile | 60 |
| 5.6.1 | Rendering the text containing images | 60 |
| 5.6.2 | Creating the text containing images | 60 |
| 5.7 | Routing in the frontend | 61 |

| | | |
|----------|--|-----------|
| 5.8 | Internationalization of the application | 62 |
| 5.9 | Utilizing ESLint, auto-import and GitHub Copilot for a more efficient coding | 62 |
| 5.9.1 | ESLint | 62 |
| 5.9.2 | Auto import | 63 |
| 5.9.3 | GitHub Copilot | 63 |
| 6 | Deployment of the prototype | 65 |
| 6.1 | Concept of containerization | 65 |
| 6.2 | Leveraged containerization technology | 65 |
| 6.3 | A remark on the database change | 66 |
| 6.4 | Cooperation with my colleague | 66 |
| 7 | User testing | 67 |
| 7.1 | The agenda of the testing | 67 |
| 7.2 | Participants profile | 68 |
| 7.3 | The testing process | 68 |
| 7.4 | Observations | 68 |
| 7.5 | Fixing the observed issues | 71 |
| 8 | Future of the project | 73 |
| 8.1 | Fixing the remaining known issues | 73 |
| 8.2 | Refactoring the Helpdesk Configuration | 73 |
| 8.3 | Shift in the email strategy | 74 |
| 8.4 | Implementing new functionalities | 74 |
| | Conclusion | 77 |
| | Bibliography | 79 |
| | A Acronyms | 89 |
| | B Contents of attachments | 91 |

List of Figures

| | | |
|-----|---|----|
| 1.1 | Status transition configuration example | 7 |
| 1.2 | Customers view when creating an issue | 8 |
| 1.3 | Customers view when editing an issue | 8 |
| 1.4 | Customers view of list of issues | 9 |
| 1.5 | Issues list customization menu | 9 |
| 1.6 | Jira Service Management form | 19 |
| 3.1 | Vue framework v-for snippet | 29 |
| 3.2 | Options API vs Composition API | 33 |
| 4.1 | Design of the architecture | 40 |
| 4.2 | Final layout and homepage | 43 |
| 5.1 | Frontend directory structure | 49 |
| 5.2 | JWT refresh logic | 54 |
| 5.3 | TinyMCE configuration | 55 |
| 5.4 | Processing file with multer middleware | 58 |
| 5.5 | Vue Router Navigation Guard | 61 |
| 5.6 | ESLint configuration for the FE | 64 |

List of Tables

- 3.1 Comparison of React, Vue and Svelte 31
- 3.2 Quantified comparison of React, Vue and Svelte 32

Introduction

The paper [1], titled “Issue Tracking”, was published in 2003 and mentions that the term *issue tracking* is overtaking the term *bug tracking*, which was more common in the past. This fact represents the idea that there is always more to do in the software projects - bug fixes of existing problems, enhancements of the current software with new functionalities, optimization of existing code, and so on. This idea is further supported by the diploma thesis [2] from 2009 titled “Issue tracking systems”. The author shares the idea that changes are a constant feature of software development. He also presents actions that are usually done in a software life cycle. Moreover, he presents the term **Configuration Management**. It stands for the process of controlling and documenting change to a developing program. The key part of Configuration management is in author’s words **Requirements Management**, which involves establishing and maintaining agreement between customer and developer on both technical and non-technical requirements. This agreement forms the basis for estimating, planning, performing and tracking project activities throughout the project and for maintaining and enhancing developed software. Nowadays, the Requirements Management activity is done by software tools, called Issue Tracking Systems, Trouble Tracking Systems, Bug Tracking Systems, etc; with the shared goal of collecting and managing the requirements, as well as tracking progress on the requirements.

To be able to manage the requirements correctly, the tools have to cover multiple aspects and also be able to hold information on multiple topics, such as:

- What should be fixed or created?
- How should it work the right way?
- Who reported the request, who confirmed, analysed, implemented and verified the solution?
- When was the request reported, when was the issue fixed and when was it verified?
- What changes in code were made?
- How long it took to handle the request?

These are only some of the functionalities of an issue tracking system, as per Janák [2]. Some of these functionalities are rather technical, what serves well the developer and project manager. Yet for the customer, it might present a problem. As we read in the definition of Requirements Management, the goal is **establishing and maintaining an agreement between customer and developer**. This might be rather hard in case the customer doesn't have technical background and lacks basic digital skills. According to a post [3] from Eurostat, 46% of EU citizens aged 16 to 74 didn't have even basic digital skills in 2021.

From the presented facts, it is plain to see that almost half of the active population of Europe would face problems navigating Issue Management Systems, which are being used as the main form of communication and between the developer and customer. This might not come as a problem to many, yet it has its impacts. The struggling customer wants to solve his issue, and since he can't navigate the system correctly, he can decide to do one of the following:

- Call the project manager or developer directly. This action takes time from both sides and can also negatively impact the life cycle of the issue, if certain process flow has been agreed on by both sides.
- Try to use the system to the best of his skills. The result is, in most cases, a poorly defined issue with multiple actions required from company side before the development can begin.

I think we can agree that all of these actions present negatives for both sides. In some cases, the customer might even feel that the customer service offered by the software company is lacking, and this is highly problematic. The article [4] from 2011 mentions that 70% of the customers leave a service provider not because of the price or product quality issues, but because they don't like the quality of the customer service. The companies should therefore stride towards a good-quality customer service, so that they are able to maintain their clients.

This is where my thesis comes into play. Redmine is an open-source project management web application, as stated on its official documentation page [5]. The page also talks about multiple functionalities of Redmine, presenting it as a technical and customizable tool. My goal is the design and implementation of a web app that will serve as a Redmine client for a technically less-skilled audiences. It should hide the technical part of the Redmine from them, yet still allow them to communicate trouble-free with the people from the software company. Such client application should provide a better customer service to the clients, leaving them more happy, as well as saving time for both, the developer and the customer sides. The name of such application is "help desk" and I will try to describe these terms better in the following chapter.

State-of-the-art

The chapter State-of-the-art will explore Redmine, as well as the idea of help desk and existing help desk solutions for Redmine.

1.1 An Overview of Redmine

I've already mentioned Redmine couple times. It is one of those Issue Tracking Systems, Bug Tracking Systems or Trouble Tracking Systems. For our purposes, I will call it Issue tracking tool, as that's what's written on the English Wikipedia site about Redmine [6], and it's also the first term that comes to my mind when I hear Redmine. In this section, I will present what the Redmine is capable of. My main source will be the official Redmine documentation [5], which describes the main Redmine features, but also contains several guides, such as User's guide and Developer's guide.

1.1.1 Who uses Redmine

To bring a bit of merit to the Redmine tool, I will list a couple of projects that use Redmine. Those projects are:

- **Ruby language** - the 17th most used programming language in the world, as of April 2024, according to PYPL index [7]. This case is quite interesting as the developers of the Ruby language use the Redmine system to track the development process of Ruby language. On the other hand, the Redmine software is built using Ruby on Rails framework, which is based on the Ruby programming language. This information was discussed on the official Redmine forums at [8].
- **TYPO3** - a web content management system, prominent in Europe, with around half a million installations declared on their Wikipedia entry at the time of writing this thesis [9].
- Universities in France, Ukraine, Switzerland, Germany, United Kingdom, USA.
- Several projects from different governments and ministries around the world, such as French, Chile, USA and Japan.

- Kingdom Come: Deliverance [10], a realistic medieval RPG developed by Warhorse studios [11], a Czech video game developer company.

There are more projects mentioned on the “Who uses Redmine?” page [12], an official Redmine source. To talk about my own experience, I’ve used Redmine during my studies at Czech Technical University at Prague, and also in my work in the software industry. The goal of this subsection was to present the fact that Redmine is a relatively widely used issue tracking tool.

1.1.2 Features of Redmine

Redmine offers a wide functionality and does not focus solely on the issue tracking area. The functionalities are divided into modules. Each module represents a singular unit of functionality. Examples of modules are **Issue tracking**, **Time tracking**, **News**, **Documents**, **Files**. The modules are configured per project, and they can be enabled and disabled whenever, with the related data being stored in case of disablement. Some of the Redmine functionalities include:

- **Managing multiple projects** - an instance of Redmine can handle multiple projects at the same time. This means a software company has a single instance of Redmine and is able to govern multiple projects inside this single instance of Redmine. Each project has its own context - issues, users, configuration. The software company operates in this single instance of Redmine, customers have per project access. There is also a support for sub-projects, creating a Child-Parent relations between projects.
- **Roles and permissions** - roles are defined per project. A member of a project can have one to multiple roles. A role is a collection of permissions. Permissions dictate what actions can be performed, such as managing projects, managing forums, documents, files, wiki pages and all the operations with issues.
- **Issue tracking** - issue is the core entity of Redmine. It is always bound to a project and can be owned by a user. Relations between issues, such as “related to”, “duplicates”, “blocks”, “precedes” are supported. Watchers are a second relations related functionality - list of users watching the issue. If an update to the issue happens, those users will be notified. Subtasks are a way of dividing an issue into a smaller tasks with more control over the issue in mind.
- **Gantt chart and calendar** - provide time-based view of a project. The Gantt chart is a bar chart and is one of the most popular and useful ways of showing activities displayed [13]. Redmine uses the more advanced version of Gantt charts which also display the inter-dependencies between issues present in the chart.
- **News, documents and Files management** - the News are a post about the project or sub-projects. The Documents can be used to store the documentation, such as User documentation and Technical documentation. Finally the Files are a way of storing the files that are related to the project, yet don’t have a special relation to an issue.

- **Per project wiki** - the wiki is, in my experience, used to store developer-focused knowledge about the project. There is a possibility for sub-pages, locking certain pages, seeing history of the pages. Watching pages is also possible, meaning a user gets notified whenever a page is updated, same as watching issues.
- **Versions** - allows the user to plan and track changes in the project. The issues are assigned to versions. A roadmap functionality allows for a version-based view, where the user can see percentage of issues done, as well as wiki pages, description, title and couple other attributes assigned to the version.

This is by far not the definitive list of Redmine functionalities. I've just presented a couple of core Redmine modules and functionalities, which were listed in the official Redmine documentation [5]. There is also a possibility to create a module or a plugin of it's own, a developer guide for such case exists [14]. A plugin is a piece of software that enhances the existing functionality of Redmine, or adds a completely new one.

1.1.3 Roles in Redmine

Redmine has two system roles - Non member and Anonymous. According to a blog post from RedmineUP [15], a service focused on Redmine plugins, Redmine comes with a set of predefined roles, such as Manager, Developer and Reporter. Thanks to the versatility of Redmine, we can edit and delete these predefined roles, as well as add completely new ones. Each instance of Redmine could have a different set of roles with different permissions attached to them. For the simplicity, I will present several roles. From my experience and belief, these roles should be applicable on most of the software projects:

- **Manager** - the responsible one from the software company. He should have full access to all the functionalities of a project. This role should also be able to configure the setting of the project.
- **Developer** - should be able to see the issues, alter it's states. The developer should also be able to access the wiki page of the project, the Gantt chart and other project's knowledge sources. He doesn't need the ability to alter the configuration of the project.
- **Customer** - the only roles for the other side of the spectrum. The customer needs to see the issues, their status and progress. He also needs the ability to report a new issue. Depending on the relation set between customer and software company, the customer might also want to see the time spent on each of the issue. He does not need the access to knowledge sources nor the ability to configure the Redmine project entity.

For all the roles I've mentioned above, I will use a configuration that has been used for a couple of years in my current place of work, a small software development company. In this configuration, the customer role doesn't have the "see time spent" permission granted by default. This permission is rather added additionally, by granting another role titled "+ time spent". This approach allows for a higher customization.

In the context of this thesis, I will work with the Redmine configuration from current work. It is described in the paragraphs above. The customer will be also granted the “+ time spent” role, unless explicitly stated otherwise.

1.1.4 Issue tracking feature of Redmine

Although many parts of Redmine are interesting, my main focus will be on the Issue tracking, the part that is being used by the customer the most. It is also the part of Redmine my work will focus on. In the following paragraphs, I will describe the core parts of Issue tracking.

Fields of Redmine issue

According to RedmineUp [16], a Redmine Issue has the following fields by default:

- **Tracker** - category of issue, clarified later.
- **Subject** - name of the issue.
- **Description** - resume of the issue.
- **Status** - current state of the issue, once again, this attribute will be clarified later.
- **Priority** - what is the priority of the issue. The predefined values are Low, Normal, High, Urgent, Immediate, the manager can add more levels.
- **Assignee** - an issue can be assigned to either a single user or to a group of users.
- **Private** - a flag reducing the visibility of the issue to a smaller list of users.
- **Parent task** - select another issue as a parent issue of this one.
- **Watchers** - I've already mentioned them: it is a list of users who follow the issue and receive e-mail notifications about the issue updates.
- **Start / Due date** - dates when the work on issue should start and when the issue should be finished.

These are just the predefined fields of an issue, there is a possibility to add custom fields. Both, the predefined and the custom fields, can be used as a filter and as a searchable when using the Redmine search function. More information on the custom attributes is available in the “Custom fields” part of the Administrator guide [17].

Manager's view of Issue tracking

The manager of the project is able to alter the configuration of the project. This is in most cases the entry point of every project - the manager sets up the configuration, of the project. The configuration consists of:

Status transitions
Fields permissions

Select a role and a tracker to edit the workflow:

Role: Developer Tracker: Bug Edit Only display statuses that are used by this tracker

| Current status | New statuses allowed | | | | | |
|----------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| | New | Assigned | Resolved | Feedback | Closed | Rejected |
| New issue | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| New | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Assigned | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Resolved | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Feedback | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Closed | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| Rejected | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> |

▶ Additional transitions allowed when the user is the author
 ▶ Additional transitions allowed when the user is the assignee

Figure 1.1: Status transition configuration example [18]

- **Types of trackers** - they allow for splitting issues into different categories, such as **Bug**, **Feature**, **Test**, **Epic**. The trackers are customizable, the manager can define what issue fields it contains, what is the default status and what is the workflow. The manager can also add new types of trackers.
- **Issue status** - what status is the issue in. Each of the statuses has a name and a flag indicating whether the status means the issue is closed. These statuses are highly modifiable, as per nature of Redmine.
- **% done** - reflects the level of completion of the issue. It can be semi-automatized with the use of issue statuses.
- **Workflow** - defines the status transitions allowed for the combinations of role and tracker type. It is basically a 2D map where the manager defines what transitions are allowed for each of the states. An example of such map can be found in the figure 1.1.
- **Field permissions** - set special flags, such as **read-only** and **required** for each field of the issue. These permissions are defined per state and per field.

Customer's view of the Issue tracking

When the customer wants to create a new issue, the screen from the figure 1.2 is displayed. In my opinion, there are several fields the customer does not explicitly need to set, such as Tracker, Assignee, Checklist, Parent Task, Start and Due dates as well as %Done. The “new issue” screen is very similar to the “edit issue” screen from the figure 1.3, sharing the same fields. This is the first area I'd like to improve the customer's user experience in.

The second area, that would in my opinion deserve an improvement, is the list of issues. An example is visible in the figure 1.4. The list is in my opinion not that bad, yet it might contain too many columns. The columns in the

1. STATE-OF-THE-ART

New issue

Tracker * Requirement ▾

Subject *

Description **Visual editor** Preview

Paragraph ▾ **B** *I* U ~~S~~ <> A - A - *I*x

Status * New ▾

Priority * Normal ▾

Assignee * ▾ Assign to me ✕

Parent task ✕

Start date 15 / 04 / 2024 ✕

Due date dd / mm / yyyy ✕

% Done 0 % ▾

Checklist

Files **Browse...** No files selected. (Maximum size: 198 MB)

Create **Create and add another**

Figure 1.2: Customers view when creating an issue

Edit

Change properties

Project * Helpdesk test ▾

Tracker * Requirement ▾

Subject * Test 01

Description **Edit**

Status [s] * New ▾

Priority * Normal ▾

Assignee * ▾ Assign to me ✕

Parent task ✕

Start date 13 / 03 / 2024 ✕

Due date dd / mm / yyyy ✕

% Done 0 % ▾

Checklist

Notes **Visual editor** Preview

Paragraph ▾ **B** *I* U ~~S~~ <> A - A - *I*x

P

Files **Browse...** No files selected. (Maximum size: 198 MB) [Edit attached files](#)

Submit [↵] **Cancel**

Figure 1.3: Customers view when editing an issue

list are customizable per project and per user. This customization is rather technical, as visible in the figure 1.5 and is not persisted per save. Therefore, a lot more optimal approach would in my opinion be having a table with less columns.

1.1. An Overview of Redmine

Issues New Issue ...

Filters Add filter

Status open

Options

Apply Clear

| <input type="checkbox"/> | # | Tracker | Status | Priority | Subject | Assignee | I plan to finish by | Due date | % Done | Category | Target version | |
|--------------------------|-------|-------------|----------------|----------|-----------------------|------------------------|---------------------|----------|--------|----------|----------------|-----|
| <input type="checkbox"/> | 23071 | Requirement | New | Normal | Velky problem! | | | | | | | ... |
| <input type="checkbox"/> | 22964 | Requirement | New | Normal | April v2 | | | | | | | ... |
| <input type="checkbox"/> | 22963 | Requirement | New | Normal | April v1 | Jagu Development | | | | | | ... |
| <input type="checkbox"/> | 22797 | Requirement | New | Normal | Toto ide. z FE | | | | | | | ... |
| <input type="checkbox"/> | 22796 | Requirement | In development | Normal | Testovacia issue c.1. | | | | | | | ... |
| <input type="checkbox"/> | 22795 | Requirement | New | Normal | Testovacia issue c.1. | | | | | | | ... |
| <input type="checkbox"/> | 22794 | Requirement | New | Normal | Test 01 | Tester (Jakub Lukačín) | | | | | | ... |

(1-7/7)

Figure 1.4: Customers view of list of issues

Issues

Filters open

Status

Options

Columns

Available Columns

- Project
- Parent task
- Parent task subject
- Author
- Updated
- Start date
- Estimated time
- Total estimated time
- Spent time
- Overall spent time

Selected Columns

- Tracker
- Status
- Priority
- Subject
- Assignee
- I plan to finish by
- Due date
- % Done
- Category
- Target version

Group results by

Show Description Last notes

Totals Estimated time Spent time

Figure 1.5: Issues list customization menu

Another area to improve in the issue list field are in my opinion the filters. Applying a filter in the Redmine UI works in two steps:

1. From a drop-down list, select the filter you'd like to add. This list is rather long, with more than 40 possible filter options, just in the default issue configuration, without the use of any custom fields.
2. After selecting the desired filter, choose the operator¹ and the value².

¹dependant on type, **is**, **is not**, **smaller or equal**, **higher or equal**, ...

²again, depending on type, could be a date, a type, a value, ...

The filtering is therefore rather complex. Don't get me wrong, it works great if you know what you're looking for, as you are able to use any logical combination of filters imaginable. This technical approach is extremely well-suited for the project manager, as he can take full advantage of it. Yet, the customer does not really need such control over the issues and this autonomy could bring more problems than benefits. That's why I'd like to simplify these filters in my implementation.

1.1.5 Summary of the Redmine overview subsection

In this section, I've introduced the Redmine software. At the beginning, I've tried to emphasize the fact that Redmine is not just a small unknown project, but rather a widely and internationally used issue tracking tool. Afterwards, I've represented it's main features and architecture and roles. Finally, I delved deeper into the Issue tracking feature of Redmine. I've presented the main entity of this feature - the **issue** entity, as well as the areas where this entity is used, displayed and filtered. During this presentation, I've pinpointed several places that would benefit from a more simpler approach, an approach I'd like to take in my solution.

1.2 What is a help desk

Besides **Redmine**, the word **Helpdesk** is second, rather non-generic word in the title of my thesis **Redmine Helpdesk Web Application**. I'll therefore devote this section to the topic of help desk, trying to shed a bit of light on it.

First of all, there is the grammar side of things. According to Cambridge dictionary [19], the word **helpdesk** exists and means "a service provided by a company to help customers when they have problems with products they have bought, for example, computers, or to give them information". There is also an entry for the phrase **help desk** in this dictionary [20], meaning "a service that provides information and help to people, especially those using a computer network". Thanks to these two dictionary entries, as well as similarities in their meanings, I'll assume both of these entries can be used interchangeably. I'll therefore accept sources talking about both helpdesk as well as a help desk. To keep things standardized though, I'll try to stick to a word helpdesk in this thesis.

Looking at the helpdesk entry in the English Wikipedia [21], the talk is about a department or a person providing assistance and information, usually for electronic and computer problems. This matches the Cambridge entries with a small difference that the dictionary is talking about a service, rather than a department or a person. What is interesting in the Wikipedia entry is the following paragraph: "A main function of the Help desk is to separate issues from defects. Many issues can be solved at the Help Desk level such as password resets and simple misunderstandings. Some issues will be the result of actual product defect which should be forwarded to a development team for resolution." After reading this bit of information, the talk about a person or a department makes a lot more sense. An old-school helpdesk should serve as the first level of service, where someone with limited privileges can perform basic support. This basic support however doesn't cover any changes in the

provided software. The requests on the software changes should be forwarded to the developers.

I'd like to discuss the application of this division on my current case. Redmine supports the trackers customization could be used to fulfill this division. Despite this fact, I think I won't implement this division in my solution. The explanation is rather simple - introducing this division would require the customer to correctly assign the right type of request - either a defect or an issue. This might not be all the time possible from with the information accessible to the customer. A better approach to this case is in my opinion a default value for the tracker field, with the project manager setting the right tracker after processing the issue inside Redmine.

1.3 Competing helpdesk solutions for Redmine

Taking into consideration the vast usage of Redmine, it doesn't come as a surprise that there already exists a bunch of Redmine helpdesk solutions. In this section, I'll analyse few of them, with the goal of exploring a couple of interesting features suitable for my own solution. For each of the reviewed plugins, I will present its features and my perspective on it.

1.3.1 RedmineUP's Helpdesk Plugin

The first solution I looked into is named RedmineUP's Helpdesk Plugin. This plugin adds a new module to your instance of Redmine. It was developed by the RedmineUP company, a software company focusing on plugins for Redmine. The mentioned Helpdesk plugin is one of their flagships, along the CMS plugin. The plugin itself costs 399 US dollars and in this price you also get twelve months of updates and support. I've sourced the information about the plugin from RedmineUP's official site available at [22].

Features

- **Customer information** - display the information about the customer at the "View Issue" screen. The customer information is further customizable, with the usage of tags. Also supports "previous customer's issues" relation.
- **Reply to the customer directly from the Issue's page** - the ability to respond to the assigned customer directly from the mentioned "View Issue" screen.
- **Turn issues into tickets** - extend the functionality of the Issues list, adding a "filter by customer" functionality.
- **Respond faster with auto-responders** - the ability to create an auto-responder for a customer's first message.
- **Helpdesk Widget with API** - employees and customers can send inquiries or set tickets from any page. The fields inside this form can be modified and prepopulated.

- **Smart workflow automation** - apply automatic e-mail processing to sort incoming messages based on given rules and criteria. It is able to modify the issue fields, as well as move the ticket to another project.

Conclusion

The main advantage of this plugin is in my opinion the auto-responder. From my experience as a customer, it feels good to have a confirmation email. The widget that allows a customer to create a new issue on the fly is also an interesting idea, although a form is in my opinion a better suit for such case. Overall, the plugin adds lots of functionality to Redmine, but is in my opinion more focused towards enhancing the experience of the project manager. I will nevertheless consider some of it's features in my design.

1.3.2 Lightweight helpdesk plugin for redmine

The second existing solution I looked into is available at the [23] repository. It is a lightweight helpdesk plugin. From my understanding, it is tightly coupled with the "create issue from an email" functionality of Redmine. Since it is so tightly coupled, it makes sense to cover this functionality first, before discussing the plugin itself.

Create issue from an email functionality

This functionality is rather well defined in the administrator's guide [24]. The main idea is the fact that an issue can be created, as well as commented on directly from an email. A customer sends an email to a predefined e-mail address. The Redmine instance accesses this e-mail inbox and generates either an Issue or a response to an issue. This functionality is available on Redmine without any extra installation, a user, however, needs to set up a bunch of configuration before this functionality works. A full guide on this functionality available under the already mentioned guide [24].

Features

The mentioned lightweight helpdesk plugin adds a single feature to this functionality - a first reply email, which is sent every time a new issue is processed by this functionality. A user can define the body and footer of this email, as well as use some of the issue fields in the response. The plugin is free to use, has almost 200 stars on GitHub [23] and supports many versions of Redmine, such as 3.0.x, 4.0.x, 4.2.x and 5.0.x.

Conclusion

Overall, the plugin seems to solve the one functionality it focuses on very well. The problem is that it is dependant on the "Create issue from an email" approach. I understand that the first response e-mail is an important concept for a helpdesk solution, but it does not impact any other parts of my problem. This plugin is therefore not really applicable to my problem.

1.3.3 HelpDesk for Easy Redmine

HelpDesk for Easy Redmine is another competing solution. According to its English Wikipedia entry [25], the initial release of the Easy Redmine software was in 2007. It functions as an extension to Redmine. In the initial release, the Easy Redmine had modules for project financing. Currently, it offers several extensions of Redmine, such as **Resource Management**, **Agile**³, **Finance Management**, **Business2Business CRM** and also **HelpDesk**. According to their website about the HelpDesk plugin [27], more than 300 000 Redmine users are supported via this HelpDesk plugin.

Features

The main features of the Easy Redmine helpdesk plugin are the following ones:

- **SLA monitoring and reports** - the project manager can see information such as average first response, time to solve, SLA fulfilment. This data is especially important for managers.
- **Tickets from e-mail** - this is the same functionality as the one I've described in the previous subsection. My guess is that Easy Redmine built a more accessible configuration over the setup of this functionality.
- **Simplified UI** - a PCMag review from 2021 [28] says the Easy Redmine makes Redmine "slicker looking". I have to agree on that, the UI images available on the Easy Redmine websites are way nice than the default Redmine UI. A cleaner UI is something I'd like to have in my solution.
- **Sorting of tickets according to customer/products and keywords** - the ability to sort issues by customers was already mentioned in the RedmineUP's plugin. Easy Redmine goes a step further, allowing the user to filter even by products and keywords.
- **Customizable performance statistics** - since this plugin focuses heavily on the SLAs, dashboards depicting performance in this area are available.

This solution, however, does not come free. There are two types of pricing methods available:

- **A cloud-based solution** - meaning the Easy Redmine will host the solution on their servers for you. This is the pricier one, with the cost of "Platform" tier, which includes the HelpDesk feature, being 23.9 euros per user per month.
- **On-premises** - an approach where you host the solution on your own. For this method, the pricing starts at 5.9 euros per user per month.

³software development model [26]

Conclusion

The Easy Redmine HelpDesk solution seems the most robust one out of the three reviewed. This robustness, however, comes at a relatively high service cost. Looking at its functionality, it is in my opinion more suited towards the project managers, with the SLAs and time tracking being the main focuses of the solution. An interesting thing I noticed is the fact that the solution includes a template of a message that is sent to a user after a new issue is registered in Redmine. This is a functionality that has been repeated in all three reviewed plugins.

1.4 Papers on Helpdesk with a similar use case

I've made a literature review with keyword searches such as **helpdesk solution**, **helpdesk software**, **helpdesk**. I've found two papers talking about cases similar to mine. I'll talk about them in the following subsections. These papers were using term **ticket**, and I will therefore clarify it before accessing it. According to PCMag's encyclopedia entry [29], a ticket is "a document that is filled out to request technical support. The help desk ticket, which contains the customer's name and, if applicable, an account number, describes the problem to be solved. The numbered ticket becomes part of the support workflow until the case is resolved." It has a very close relation to issue.

1.4.1 Colorado State University case study

The first paper [30] was published by people from the Engineering College at Colorado State University. Their circumstances were as follows - providing computing support for a growing population of customers on a shrinking budget. The helpdesk group consisted of eight full-time workers and around 25 part-time workers, supporting 2500 students and close to 500 faculty staff. Looking at their case, I've found some interesting ideas:

- **Centralize all desktop support work.** - Don't assign a department to a single worker, rather pool all the incoming issues and allow the workers to access this pool of tickets. On one hand, you might lose expertise, on the other hand, you gain better work distribution and therefore hopefully more happy clients.
- **Create one central point of contact for the entire IT group.** - As I understood it, the main goal was to centralize the knowledge base and communication between the IT group, by this expediting the problem solving.
- **If you are skilled enough to modify the software, go for open-source.** - Since the group has a technical background, it wasn't hard for them to customize the software to their liking. Therefore, an open-source is a go to.
- **Two types of notes on an issue.** - Have a "public" note visible to both the customer and the IT support group. Have a "private" note, that is only visible to the IT support group.

- **Very simple usability for the customer.** - A very simple web-based and email interface that provides the same information was mentioned. A creation of a ticket can create a ticket by either emailing a central email address, or accessing the web-based tool. A custom form on the website also allows for a ticket creation, without a need to enter the tool interface.

Conclusion

Although the circumstances of the IT group were not the same as mine, I'm confident I can apply some of the concepts of their paper in my own design. Their "very simple usability for customer" concept already matches my view. The "two types of notes" is an interesting idea, something I haven't exactly considered yet.

1.4.2 Oregon State University case study

Another study [31] I looked into also took place at a USA based university. Their initial circumstances were a bit different to the Colorado ones - approximately half of the campus in Oregon was supported by one central IT organization, while the other half of the campus was supported directly by the individual colleges and departments. They had tried and failed to acquire several commercial helpdesk software in the years past, and therefore decided for a development of a scalable helpdesk solution that could be used by any unit on campus.

In this paper, I've found several interesting concepts and I'll address them one by one in the following paragraphs.

The chosen approach

The constraints during the early phase development were low budget on the financial side, as well as a call for both a web interface and an email interface on the functional side. This is a a reoccurring pattern from the previous study.

The authors have decided to divide the helpdesk into several modules:

- **Call Tracking module**, focused towards the issue handling and data persistence.
- **Knowledgebase**, a module focused on providing information to the client without any interaction with the technical staff.
- **Scheduling** module, allowing the customer to schedule a meeting with the technical staff.
- **Inventory** module, displaying information about the equipment of the department.
- **Service Level Agreements** module. This module was added additionally, not being developed before the paper publication.

I'll cover the Call Tracking module as well as the Knowledgebase one in more detail, as I think they are the most important ones for my project.

Call Tracking module

The decision of the group was to try Bugzilla as a core for the Call Tracking module. Bugzilla is an open source bug tracking solution developed by the Mozilla Organization. Significant modification had to be made by the developers. These modifications were mostly required to ensure the compliance with federal laws, as well as to handle the personal information correctly.

The paper's section titled "What It Looks Like" on the "Call Tracking module" starts with a line "The customer interface has been kept as simple as possible". The authors of the paper later go on to explain that the customer can log in to a request form, where their contact data is automatically populated. They are free to fill up the affiliation, subject and description of their issue. Once the user submits the form, a new issue is registered and the customer is informed via email about the fact that a new support request has been registered in their name. The technicians have much more control over the tickets - they can filter them, edit their fields, ask for clarification, comment the ticket.

There is once again a talk about the private comments, only visible to the technicians. Another interesting functionality mentioned in the paper is linking multiple tickets together, under the "related to" or "duplicates" relation. This helps with the navigation, also allowing the more junior staff to handle a ticket by simply following an approach used in the past. This could be quite easily replicated in Redmine helpdesk with the relations between issues functionality of Redmine.

Knowledgebase

The knowledgebase module is another module I'd like to briefly cover. An initial implementation of this module allowed for comment copying from the tickets to the knowledgebase document for editing. A year and a half after the initial implementation, a major enhancement of the knowledgebase module started. The UI was improved, but mostly the search capabilities of the module were improved.

A standard use case for this module is described as follows: "Customer will have the ability to enter a question in a sentence format. When they submit the question a list of articles, ranked by relevance, will be displayed to them to choose. They will also have the ability to browse the knowledgebase by category and subject." No architecture, nor technology, was mentioned in the chapter about this module. That does not really matter, as the idea alone is something I'm more than happy with. I didn't really consider a non-interactive part of helpdesk until now. It is another tool I can consider in the design part of my work.

Conclusion

The paper talked about developing a robust, universally applicable helpdesk solution divided into multiple modules. The idea of simple UI for customer, as well as two-way issue registration⁴ were mentioned. An interesting part of

⁴email and form

the paper was the knowledgebase module, which is in my opinion very well applicable for certain use cases.

1.5 Competing helpdesk solutions outside Redmine

Although I'm partially limited by the usage of Redmine, I think it's a good idea to also review some non Redmine-based helpdesk solutions. I might be able to uncover some features of helpdesk that would serve my use case. I've tried looking up a paper on helpdesk solutions comparison, yet I didn't have any luck with that. I've afterwards tried Google. There were couple posts that sparked my interest, but most of them were self-promotions. In the end, the post that I judged the most trust-worthy was the one titled "Best Help Desk Software" [32] by Forbes Advisor, as Forbes is in my eyes a fairly reputable organization. They've also declared that they earn a commission from partner links on Forbes Advisor, but these commissions don't affect the editors' opinions or evaluations. This post was from 2024, therefore very current. It presented 10 different solutions. I'll briefly cover the most interesting ones in their own subsections.

1.5.1 Zoho Desk

The Zoho Desk was marked as the best overall, getting full 5 out of 5 rating. The reviewers praised mainly the fact that it is feature-packed, yet easy to use. Another major advantage was the fact that bonus features such as live chat and mobile app were available out of the box and were not hidden behind second paywall. The pricing is 14 to 50 dollars per agent per month⁵.

Features

I've accessed the documentation of the Zoho Desk, available at [33]. The description was rather vast, with standards like superb ticket management, monitoring agent productivity, automation of repetitive activities and security. Innovative features were:

- **Contextual AI** - share relevant solution from your knowledge base directly with customer. This relates to the Oregon's university knowledge-base, with an improvement in the used technology, as the Oregon's paper was published approximately twenty years ago. Some other improvements delivered by the use of AI were auto-tagging tickets based on key aspects, sentiment analysis, reply assistant with the use of knowledge base and an AI based notifications that are fires in case an unusual activity in ticket stream occurs.
- **Self-Service** was another big feature category I feel worth mentioning. The main idea is once again the Knowledge Base which serves as a repository of solutions for commonly asked questions. Other than the knowledge base, Guide Conversations can be configured as a platform for self-service experience. The self-services are later embeddable into websites and mobile apps outside of the main Zoho tool. An option to divide

⁵an agent is a person handling the incoming requests

these functionalities per-brand is also available, allowing for a distinct self-service portal for each brand, in case the company owns more of them.

- **Extensibility** - although extensibility is not a novelty, a single functionality caught my eye. Zoho offers an SDK that allows for a custom mobile apps development. This is in my opinion an innovative approach. I don't think I'll be able to replicate something like that in my project, but it occurred as a rather progressive idea to me and I wanted to let the readers know about such option.

Conclusion

The Zoho Desk is a very complex out of the box helpdesk solution, which seems to be top of the class in the area of helpdesks. The solution also leverages artificial intelligence, which is a big trend these years in many fields, such as medicine [34], education [35], product development [36] and agriculture [37]. Many parts of the helpdesk are customizable to a high degree, with an option of custom mobile app development available. Reports and dashboards seem to be well covered as well. Over hundred thousand businesses worldwide use the Zoho Desk, and the pricing doesn't seem to be over the roof.

1.5.2 Jira Service Desk

Jira is a tool allowing for bug tracking, issue tracking and agile project management [38]. It is therefore one of the issue tracking tools, same as Redmine. The Jira Service Management⁶ is another, different product, offered by Atlassian, the owner of Jira. One of the features of the Jira SM is Request Management [39], and this feature offers a Service desk, corresponding to a helpdesk. Thanks to this fact, the Jira SM was also included in the mentioned review of helpdesks by Forbes, getting the "Best for enterprise service management" tag. The pricing of the Jira SM is very similar to Zoho Desk, capping at around 50 dollars per user per month. The difference is that Jira SM also offers a free tier, for up to three agents. All of this information was sourced at the official Jira SM pricing site [40].

Features

Both the Request Management and the Service desk are able to leverage machine learning and AI to support the clients, as well as the staff. Standards like self-service, reports, metrics and SLAs are all available in the Jira SM. Two interesting features of the Jira SM are:

- **Slack and Microsoft Teams support**, allowing for a two-way sync between the communication software and the Jira SM. This reduces context switching and information gaps for employees and agents.
- **Dynamic forms** - Jira offers a node-code/low-code form builder, which provides dynamic forms that only surface the relevant fields to the employee. Over 300 pre-built form templates are available, and they allow for a fast collection and validation of the information about a request.

⁶Jira SM for short

The image shows a web form titled "Report incident" from Jira Service Management. At the top left, it says "Helpdesk FIT / ICT" and has a wrench and screwdriver icon. The form fields are: "Email confirmation to" (text input), "Summary" (text input), "Category" (dropdown menu), "Description" (rich text editor with a toolbar showing options like bold, italic, text color, list, link, image, table, code, quote, and link), "Priority" (dropdown menu with "Medium" selected), "Impact" (dropdown menu), and "Attachment" (a dashed box with the text "Drag and drop files, paste screenshots, or browse" and a "Browse" button). At the bottom left are "Send" and "Cancel" buttons. At the bottom center, it says "Powered by Jira Service Management".

Figure 1.6: Jira Service Management form

I've actually seen one of the Jira SM forms before, as the ICT department at my faculty leverages the Jira SM. The image of this form is available in the figure1.6. I will analyse this form more in the design chapter, as I plan to take inspiration from it. For now, I just wanted the reader to get an idea of how does such a form look like.

The link between Jira and Jira SM

As I've already mentioned, both the Jira and Jira SM are a product of their own. Each of them can function on their own. This implicates the fact that the Jira SM has an own database of tickets. Of course, there is a possibility that a company will use both products. In this case, an automation can be used to create a link between a Jira issue and a Jira SM ticket. According to an official post from Atlassian on the topic "How Jira Service Management and Jira work together" [41], when running both the Jira and Jira SM, this automation allows for comments sharing across these linked entities.

Conclusion

The Jira SM is an interesting solution, as it very closely resembles my case. The SM synchronizes and works in in cooperation with Jira, an issue tracking tool, where I'd like to implement a solution that works in cooperation with Redmine,

another issue tracking tool. Although I haven't explicitly found any innovative features of the Jira SM I'd like to consider in my design, I got information on data architecture of the Jira SM - Jira, which is in my opinion a very valuable insight.

1.5.3 Spiceworks help desk

The Spiceworks help desk is the third participant of the Forbes review that caught my eye. The main reason as the fact that it was marked "Best free option". According to the Forbes review, it includes all the basic features you need to get started, such as ticket management, remote support sessions, ticket rules, monitoring, reports, customization, self-service and ticket collaboration. It has no ticket nor admin limit and will remain free. This is due to the fact that it has partnerships with complementary software. Many of these vendors offer discounts and their service right inside Spiceworks tool, so that you can further expand your toolkit.

Features

As I wanted to know what does a completely free solution of a helpdesk offer to its customer, I've accessed the documentation of the software, available at [42]. Some of the interesting features are:

- **Extremely Easy to use** - the Spiceworks runs the software for you, therefore there is no server procurement, nor any setup or maintenance required. On one hand, this is extremely helpful for start-ups and small companies. On the other hand, you as a customer are quite prone to a vendor lock-in, a situation where you as a customer become dependant on a vendor for products, unable to use another vendor without substantial switching costs [43]. As already mentioned, this seems to be their business model.
- **Android and iOS native apps** - mostly a standard.
- **Automated responses and ticket tagging** - leverage the automation, just set up the configuration.
- **Own customized knowledgebase** - allows for sharing and creation of multiple sets of information. Also supports custom articles creations.

Conclusion

Overall, the Spiceworks help desk might be a good software, but it didn't capture my interest in any way. Comparing it with Jira SM and Zoho Desk, it seems rather lacking. This is further supported by the fact that they are marketing the automated responses as one of the features - this was common for the Redmine plugins, but not for an out of the box solution.

1.6 A brief outline of the state-of-the-art chapter

At the end of this chapter, I'd like to present a concise summary of presented ideas. In the beginning of the chapter, I've discussed Redmine with all of its

major features. I went a bit deeper into issue tracking, an area my project focuses on. After the Redmine introduction, I've also briefly introduced a concept of helpdesk⁷. These sections were followed by the presentation of already existing helpdesk solutions for Redmine. I've went over three possibilities, each of them interesting on their own, yet none really handling the problem I'd like to tackle. After analysing the competing Redmine solutions, I've looked at two papers talking about a help desk development in university area. Although scope of both of these projects was much bigger than mine, I've discover some interesting ideas, such as Knowledgebase. As the final piece of content in this chapter, I've presented three out of the box helpdesk solutions. The most interesting one for me was the Jira Service Management one, as it allows for a use case similar to mine - synchronising with an issue tracking tool. Moving forwards, i plan to use knowledge acquired during the state-of-the-art analysis to design a fitting solution for my use case.

⁷or a "help desk"

User requirements gathering

2.1 Background of the topic

I've had few discussion with my supervisor, Ing. Malec, about the idea of this project. He is the main project manager in the company I currently work for. This company uses Redmine as an issue tracking tool, and demands their customers to discuss and report their issues via the Redmine. Despite this fact, both Ing. Malec and Ing. Hunka, both faces of the company, have to handle the communication with customers via phone on almost daily basis. This is not always undesirable, as sometimes the eye-to-eye⁸ communication has it's positives. But when the customer just wants to report a possible bug or discuss a certain feature, it would be much more optimal to have the request in a written form.

The company tried to push their customers towards the Redmine usage, but often with mixed results. Due to technical aspect of the Redmine, the customers are often not able to navigate it well, as I've discussed in the introduction. I've also paid witness to several requests where the customer answered, yet forgot to reassign the issue. This might look as a small problem, but when a developer has twenty issues to cover, he can very quickly forget about the one customer forgot to assign. The result is rather negative for both sides - customer commented the issue, therefore he thinks the company should react. The developer doesn't have the issue assigned and possibly wasn't even notified about the comment - therefore he thinks the customer should react.

Since the issue lists and filters on Redmine are rather technical, Ing. Malec had an idea of designing a simple, customer-oriented interface for Redmine. The flow of the development - all the status, assignment changes as well as the comments should stay inside Redmine. The customer should be accessing the simple UI, and the activities performed there should affect the states of issues in Redmine.

2.2 Theory on requirements

In the next sections, I'll divide the requirements into two categories - the functional ones and the non-functional ones. At first, I've come across this cate-

⁸“ear-to-ear” in this case

gorisation during my bachelor studies at the university. The English Wikipedia entry [44] defines the functional requirement as a “function of a system or a component, where a function is described as a summary or specification or statement of behavior inputs and outputs.” They should define what the system is supposed to accomplish. They are supported by non-functional requirements, also known as “quality requirements”. The non-functional requirements impose constraints on the design or implementation, such as performance requirements, security, or reliability. According to the Wikipedia entry [44], the functional ones should be expressed as “system must do X”, while the non-functional ones should be expressed as “system shall be X”. An interesting point of view is also the fact that:

- **The functional requirements drive the application architecture of the system.** – Implement what you have to do.
- **The non-functional requirements drive the technical architecture of the system.** – Pick the right tools for your job.

2.3 Functional requirements

The main user of the helpdesk is the customer. The customer needs to:

- Report an issue.
- See the details of a reported issue.
- See the list of the reported issues.
- Be able to filter on the list of reported issues.
- See the issues assigned to me.
- See the issues waiting for my action.
- See the issues, comments and changes made in both Redmine and help desk under his own account.

The company is another user of the helpdesk. They might not be using it directly, yet it very much affects them, as it creates tickets in their project management tool. The actions done by customer in the helpdesk application should be reflected in Redmine. On the other hand, not all the changes done in Redmine by the developer should be visible to the customer. Seeing an information about a change of a a custom issue field value is in most cases not required by the customer. Also the private comments, focused on the communication between the developers, shouldn't be shown in the issue detail. Therefore, the requirements from the company's point of view are as follows:

- See the reported issue in the project management tool as an issue.
- Have the helpdesk automatically inform client about the status updates of the issue.
- The helpdesk should inform the user about the registration of an issue.

- As a developer, have the ability to mark a comment on issue as a “team” comment. This type of comment isn’t displayed to the customers.

The functional requirement might seem a bit empty to some, but in my opinion they rather well represent the main idea of the Unix philosophy [45] - “Do one thing well.” The one thing in this case is “Serve as a simple customer-focused helpdesk integrated with Redmine.” Taking into consideration the state-of-the-art as well as the needs of the company mentioned above, it is my strong belief that building well a rather simple UI allowing the customer to report an issue and view the issues reported by the customer is a very fitting plan.

2.4 Non-functional requirements

I’ve already briefly touched on the concept of non-functional requirements in the beginning of this chapter. The article [46] from 2007 grouped exactly thirteen different definitions of non-functional requirements across multiple reliable sources. A type of consensus we can find among these thirteen definitions is the definition of what the non-functional requirements are not - they are not a not concerned with a functionality of the system. They rather describe the non-behavioural aspects of the system, such as performance of the system, quality of service and system’s design. This definition tallies with the Wikipedia entry definition presented. During my discussion with the supervisor, I’ve gathered several non-functional requirements:

- **The system supports multiple language mutations.** This particular requirement has high priority due to the fact that the contracting company has customers from multiple countries, such as Czech Republic, Slovakia and Hungary. The help desk would not fulfill its purpose well if it could not serve the customers in a language they understand.
- **Store the data in the Redmine database.** The instance of Redmine functions with its own database. As is customizable the instance of Redmine, so is this database. An example of the database customization is available in the Plugin Tutorial part [47] of the Redmine developer guide. The user can therefore store data such as custom fields or new module data in the default Redmine database. This approach allows the helpdesk to be just a one-layer application, without a requirement for its own database layer. It would ease up the deployment of the application, as just a single application would be required to run. A disadvantage of the mentioned approach is the fact that the potential logic and calculations would have to happen in this single layer, and that is not always optimal.

2.5 A discussion with a colleague

After writing down all the user requirements in the sections above, I felt that the list might not be final, as the count is not that high. I’ve therefore opted to seek assistance from my colleague Libor.

Libor is a salesman and can relate to customer’s point of view, as he was in their shoes once, being a customer of a software company. We had a call

2. USER REQUIREMENTS GATHERING

together and I presented him the intended idea of the application. I've also presented him the user requirements I had gathered. He liked the idea, and presented me with couple interesting ideas of his own, such as:

- A full text search in the description and subject of the issues would be helpful for the customers. This functionality should probably be located on the issue list screen.
- The customers usually report two types of issues. A bug, for which they want an immediate fix, and a new functionality request, which might take couple weeks to be fully implemented. The user should be able to work with both of these categories in help desk.
- The customer is mainly interested in two pieces of information: whether the task is being worked on and when it will be ready. The customer should therefore be able to see both of these details in the help desk.

The consultation was from my perspective quite helpful as I was able to define three more functional requirements for the help desk. Furthermore, I had the opportunity to validate my concepts with an individual who previously belonged to my application's target group - customers, which gives me confidence in the accuracy of the gathered requirements.

Technological stack

3.1 Choosing the right technology for the application

The one thing I'm sure about is the fact that I'll be developing a web application that will serve as a UI. According to this post [48] about Web standards by Mozilla, there are three technologies present in the web development:

- **HTML** for structure and semantics.
- **CSS** for styling and layout.
- **JavaScript** for controlling dynamic behaviour.

Another important concept is a **DOM** - a shortcut for Document Object Model. Sourcing the official Mozilla documentation on DOM [49], a DOM is an interface that treats a HTML document as a tree structure, where each node is an object representing a part of the document. The DOM has methods, which allow for a programmatic access to the tree, where a change in structure, style or document of the tree can be performed. Another Mozilla post titled "How the web works" [50] describes how are these concepts interconnected. Imagine you want to access a website from a browser on your computer. There are two sides present in this action:

- **Client** - you, the side accessing the website.
- **Server** - a computer that stores web-pages, sites or apps.

When you as a client want to access a website, a copy of a website is downloaded from the server onto your machine and displayed in your browser. This website can consist of multiple types of files - HTML, CSS, JavaScript, but also images, videos, documents, ... These files are interconnected and their goal is displaying a functional and "pretty" web-page. Looking at the Wikipedia's entry about web development [51], we can observe that much has happened since the beginnings of the web development in 1990s. Currently, "JavaScript frameworks are an essential part of modern front-end web development, providing developers with tried and tested tools for building scalable, interactive web applications". This citation was taken from post titled "Understanding client-side JavaScript frameworks". Before we continue further, I think an explanation of the term "frontend" or "front-end" is due. In web development,

this term refers to “the development of the graphical user interface of a website through the use of HTML, CSS, and JavaScript so users can view and interact with the website”. This definition sums up very well the area I will navigate in the following chapters. I will go a bit more into detail about the JavaScript front-end framework in the following subsection.

3.2 Client-side JavaScript frameworks introduction

The JS frontend frameworks are a big trend. The information I’ll present in this section is taken from Mozilla guide [52] on client-side JS frameworks and its sub-chapters.

3.2.1 A brief history of client-side JS frameworks

JS debuted in 1996. It added a much needed interactivity and excitement to a web that was, up until then, composed of static documents. This sparked a big interest, as the web became a place to do things, not just to read things. The developers working with JS started developing the reusable packages called libraries. These libraries bundled functionality that solved the problems they faced in JS development. This shared ecosystem of libraries helped to shape the growth of the web.

Now, the JS is an essential part of the web, and is being used on 98% of all websites. The web is an essential part of the modern everyday life. Web users write blog posts, manage their finances, stream music, watch movies and communicate with others, all of this done via the web, with the help of JS. All these modern interactive websites are referred to as web applications.

The phrase “advent of modern JavaScript frameworks” sounds rather decent and I’ve stumbled across it multiple times in the process of writing this thesis. It refers to a time frame when multiple JS frontend frameworks were released. I haven’t found an exact time frame for it, but taking into consideration the release dates of the frameworks, years 2010 to 2020 are a good bet. What is a framework, might you ask. A framework is a library that offers opinions about how software gets built. These opinions allow for predictability and homogeneity in an application. The predictability allows the software to scale to enormous size and still be maintainable. The combination of predictability and maintainability are essential for the health and longevity of software.

3.2.2 Why do the frameworks exist?

The main issue the frameworks are trying to solve is the following one: The programmer has to access the DOM directly, in order to render the data. The code achieving the rendering of elements is rather repetitive and long. Simple action of populating and rendering a list of elements took almost thirty lines of code in an example of the sourced guide [52]. The JS frameworks were created to make this kind of work a lot easier - by this, providing a better developer experience. They don’t bring any brand-new powers to JS, they just give you an easier access to JS’s power so you can build.

The JS frameworks also offer a way to write UI’s more decoratively. That basically means you write the code that describes how your UI should look, and the framework makes it happen in the DOM behind the scene. This is done by


```
HTML   
  
<ul>  
  <li v-for="task in tasks" v-bind:key="task.id">  
    <span>{{task.name}}</span>  
    <button type="button">Delete</button>  
  </li>  
</ul>
```

Figure 3.1: Vue framework v-for snippet [52]

providing certain framework-specific snippets by the framework. An example of such snippet is visible in the figure 3.1. The keywords `v-for` and `v-bind:key`, as well as the curly braces are framework specific code and reduce almost thirty lines of code down to six lines. Thanks to this framework, developer doesn't have to write the code, he just uses the framework's keywords to describe how should each item look like. I present couple other advantages of frameworks in the following paragraphs.

Tooling

Popular frameworks have large and active communities. These communities develop tools that improve the developer experience. These tools make it easy to add things like testing⁹ and linting¹⁰ to the project.

Compartmentalization

Most major frameworks encourage the developers to abstract different parts of their UI into components. The components are maintainable, reusable chunks of code that can communicate with one another. A code for such component is stored in a single specific file or a couple of specific files, so that the developer exactly knows where to go and make a change.

Routing

Allowing user to navigate from one page to another is a very essential feature of the web. That is due to the fact that web is a network of interlinked documents. When user follows a link on a website, the browser communicates with a server and fetches new content to display. As it does so, the URL¹¹ in the address bar changes. You can save this new URL and when you access it later or share with others, you can easily find the same page. Browser also remembers the user's navigation and allows him to navigate back and forth. This approach is called **server-side routing**. An alternative approach to this is called **client-side routing**. It describes the case when the routing is handled by the client application instead. This is mostly done in modern web applications that

⁹ensure your application behaves as it should

¹⁰ensure your code is error-free and stylistically consistent

¹¹address on the Web, is a reference to a resource that specifies its location on a computer network [53]

typically do not fetch and render new HTML files, they just load a single HTML shell and continually update the DOM inside it. This approach is referred to as single page app, SPA for short. It is, however, important to keep the routing functionality even in the SPA, as that is what a user is used to.

3.3 Client-side JavaScript framework choice

The definition of JavaScript framework fits my use case very well. I'll therefore implement my application with the help of one of them. Now, the question is which one to choose. There are several to choose from. I'll consider three options - React, Vue and Svelte.

3.3.1 React

The React is a free and open-source JS library maintained by Meta [54]. Its initial release happened in May 2013. React is declarative and supports components. It doesn't come with a built-in routing, but third-party libraries can be used to handle the routing. A rather specific functionality are "React hooks". These hooks allow the access to React state and life-cycle features from function components. They exist to make React codebase more intuitive. Looking at a blog post about React usage [55], we can note that it was used in the development process of Airbnb, Netflix, Uber Eats, Twitter, Paypal, Reddit and also Facebook. All of these are major names in the online area, therefore it is safe to assume that React is rather well functioning technology and should be considered when looking for a JS framework.

3.3.2 Vue

According to the English Wikipedia entry on Vue [56], it was initially released in February 2024, making it approximately one year younger than React. Citing the author of Vue, the idea behind the Vue framework was to "extract the part I liked about Angular and build something really lightweight" [56]. The word Angular refers to AngularJS, an already discontinued JS web framework [57]. Vue is based on components. It also offers reactivity, meaning each component keeps track of its reactive dependencies so the system knows precisely when to re-render and which components to re-render. Multiple tools, such as Pinia for state management and Vue router for routing exists as official libraries to enhance the development experience in Vue. According to the post [58] from 2021, parts of Facebook are powered by Vue. Also Netflix uses Vue for several internal projects. Some other important project developed with the help of Vue are GitLab, Google, Apple and Trivago.

3.3.3 Svelte

Sourcing the Wikipedia entry on Svelte [59], we can see that its release date is November 2016, making it the youngest of the three reviewed frameworks. There is a difference between Svelte and the other two mentioned frameworks, as Svelte compiles the HTML templates to specialized code that manipulates the DOM directly. By this, it may reduce the size of transferred files and give a better client performance. Application code is also processed by the compiler.

This avoids the overhead associated with runtime intermediate representations, which is typical for React and Vue. The bulk of the work in the case of Svelte is therefore done at the compilation time. In the case of React and Vue, the bulk of the work is done at runtime - in the browser. Svelte also uses the concept of components, more on the subject can be read in the official documentation available at [60]. The modern best-practices of the web development that are not UI focused are handled by SvelteKit. This includes routing, but also several optimizations and different rendering types. More about this subject can be read in the official SvelteKit documentation at [61]. Looking at a post titled “Top 10 Big Companies Using Svelte” [62], the big companies that use the Svelte in their projects are Apple, NBA, New York Times, IKEA, Spotify and Cloudflare.

3.3.4 Comparison of the frameworks

The data for the comparison were sourced from the post “Choosing the Right Frontend Framework” at [63], as well as the GitHub repositories of the respective projects at [64], [65] and [66].

The table 3.1 lists some important attributes of the three framework options. The learning curve is quite self explanatory. The bundle size refers to the total amount of code and assets that your web application sends to the browser [67]. The stars are a functionality of GitHub, where a user can star a project he finds interesting [68]. I’ve also took into consideration the experience aspect - my own experience with the framework and the experience of my company with the framework, as the former means I can develop faster and the latter means I can be supported and helped better. The sourced post described the three frameworks like this:

- **React** - a robust choice for large-scale applications.
- **Vue** - user-friendly and versatile.
- **Svelte** - excels in performance and simplicity.

3.3.5 The choice of the framework

The values in the table 3.2 are my take on the quantification of the values from the table 3.1. The winner of the “contest” is the Vue, seconded by Svelte. These results are acceptable to me. For a longer time, I’ve considered React a rather heavy and enterprises focused framework. This was further supported by the source blog post. As I’m a newcomer to the field of web frontend development,

Table 3.1: Comparison of React, Vue and Svelte

| | React | Vue | Svelte |
|----------------------------|-------------|-----------|--------------|
| Learning curve | steep | easier | easy |
| Bundle size | the largest | medium | the smallest |
| Stars on GitHub (in 1000s) | 221 | 44.5 | 76.3 |
| Own experience | minimal | low | none |
| Company experience | low | excellent | minimal |

having a steep learning curve React out is good for me. The Vue and Svelte are both marked as “relatively easy to learn”. Looking at the point in the table 3.2, the experience rows won it for Vue. This corresponds to the fact that I had some small experience with Vue, and also many people in my company are quite skilled with Vue. Svelte, on the other hand, got zero points as there is no own and very minimal company experience with the technology.

3.4 Technologies bundle choice

Since I’ve decided to use Vue, I’ll be able to use whatever this framework and it’s third party libraries have to offer. I’ve discussed the choice of Vue with my supervisor Ing. Malec. He was happy with the choice, as he has rather vast experience with the technology. We’ve agreed on the fact that I’ll be using version 3 of Vue. The release notes of Vue [69] state that version 3 was initially released in September 2020. It was also the latest major release at the time of writing this thesis. It is always a good idea to stick to the latest major version of the technology, as per Semantic Versioning 2.0.0 [70], a new major version means an incompatible API change. Therefore, when you decide to choose the latest major version, you should be working with the newest version of the technology. Choosing the newest major version might not be the best idea in cases when the version was released just recently, as there might not be enough documentation on the subject. Second big disadvantage of the newest major version choice can be the missing compatibility with external libraries. This isn’t my case, as the major version 3 is out for almost four years.

3.4.1 Composition API vs Options API

A composition API is a new approach in Vue. According to the official composition API FAQ available at [71], it is built in feature of Vue3 and Vue2.7. For the previous versions, it can be enabled with the use of plugin. There are few advantages of the Composition API:

- **Better logic reuse** - Composition API enables clean, efficient logic reuse in the form of composable functions¹². The composables solve all the drawbacks of mixins, the primary logic reuse mechanism of the older Options API.

¹²composable is a function that encapsulates and reuses stateful logic. Stateful means something that changes over time, e.g. the position of the mouse on the screen. More on the composables can be read in the official documentation at [72].

Table 3.2: Quantified comparison of React, Vue and Svelte

| | React | Vue | Svelte |
|----------------------------|-------|-----|--------|
| Learning curve | 0 | 4 | 5 |
| Bundle size | 4 | 5 | 5 |
| Stars on GitHub (in 1000s) | 5 | 3 | 4 |
| Own experience | 0 | 1 | 0 |
| Company experience | 1 | 5 | 0 |
| Summary | 10 | 18 | 14 |

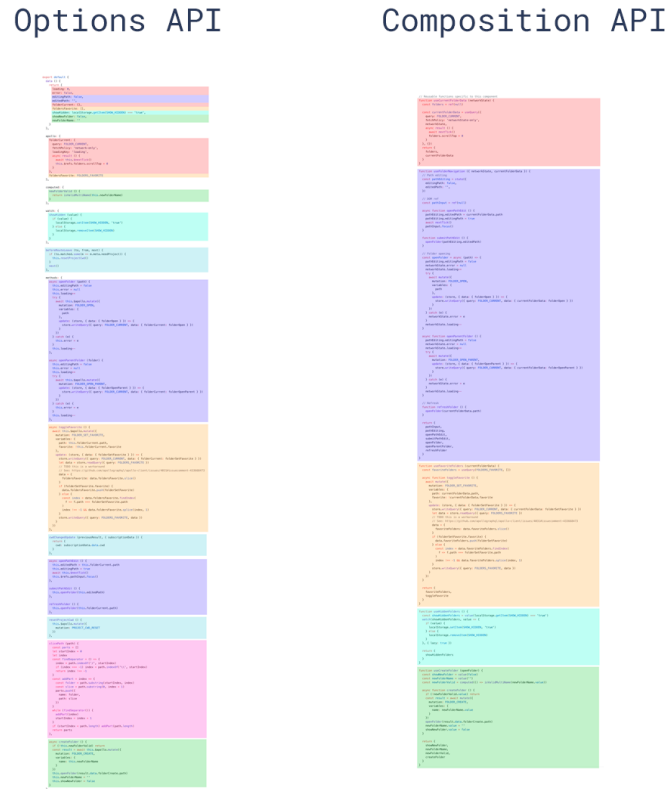


Figure 3.2: Options API vs Composition API [71]

- **More Flexible Code Organization** - This is mainly beneficial in the cases when a single component has to handle multiple logical concerns. The options API prescribed the place for each part of the code, and the logically matching parts of code couldn't be grouped together. The composition API, on the other hand, allows for more flexible code organization, allowing the same logical concerns to be grouped. An example of this advantage is visible in the figure 3.2, where the same color means same logical concern.
- **Better Type Interference** - Allowing for adaptation of TypeScript, helping to write more robust code, providing better development experience with IDE support.

Although my supervisor told me that the Composition API is a bit harder to grasp than the Options one, I've decided to go with Composition API, as I'm not scared of challenges.

3.4.2 Single File Component

While reading about the composition API, I stumbled across another feature of Vue - Single File Component¹³. A full specification on this feature is available at [73]. The feature allows the representation of a Vue component by a single file. This file has “vue” file extension and an HTML-like syntax describing the Vue component. There are three blocks in such file:

- **template** block, representing the HTML code of the component.
- **script** block, representing the JS code, usually defining the interactions in the file.
- **style** block, encapsulating the styles for the current component.

An alternative to the “Single File Component” approach is dividing the content into multiple files, such as one HTML file for the template block, one JS file for the script block and one CSS file for the style block. I will, however, stick with the Single File Component, which in my opinion offers better maintainability and cleaner code.

3.4.3 The script setup

The script setup is a feature available in case the project used both the Composition API and the SFC. This feature brings a bunch of advantages over the normal script syntax:

- More succinct code with less boilerplate.
- Better runtime performance.
- Better IDE type-interference performance.

As there were no explicit disadvantages of this approach mentioned in the official documentation of the feature available at [74], I’ll stick to it in order to achieve a cleaner codebase.

3.4.4 TypeScript or JavaScript

“TypeScript is an extension of JavaScript intended to enable easier development of large-scale JavaScript applications. TypeScript enhances JavaScript by offering a module system, classes, interfaces, and a rich gradual type system.” This definition of TypeScript was fully taken from the paper [75] called “Understanding TypeScript”. Said in simple words, using the TypeScript allows you to add types to JavaScript code. These types can detect many common errors via static analysis at build time, as is said in the post about TypeScript in Vue available at [76]. This reduces the chance of runtime errors in productions and also allows for a more confident large-scale changes in the code. The final advantage of TypeScript is an improved developer experience via type-based auto-completion in IDEs. In my history of using JS, I haven’t paid much attention to TypeScript, but since it brings many advantages and promotes a cleaner code, I’ll use in my application.

¹³SFC for short

3.4.5 UI Component Library choice

My supervisor also advised me to use a use an UI component library. As I've mentioned, Vue is based on components. These components are self-contained modules with markup, styles, and logic bundled with them. The UI component libraries are collections of these reusable building blocks. They provide pre-built base components like buttons, forms, inputs, cards and so on, which can be easily integrated into project. Using such library has two main advantages:

- **Ensuring a consistent UI experience.** – That's due to the fact that popular libraries are developed and styled by people who know what they do.
- **Saving time and effort for the user.** – Using already pre-built components saves the effort needed to build and maintain the mentioned components.

All of this information was sourced from a blog post talking about the Vue component libraries available at [77]. The information provided matches with the talk I had with my supervisor, as he explained to me that using a good component framework enables a good user experience for your app, no matter the device used. I've therefore use the already mentioned blog post [77] to get to know possible options. In the end, I've decided to settle for Vuetify, a material design based library. Material design is a design language developed by Google to unify the user experience across different platforms and devices while emphasizing clean, modern, and intuitive design. The decision was further supported by my supervisor, as he knew the library and had some experience developing with it. I've decided to settle go for version 3 of Vuetify. That was, at the time of writing this issue, the latest major release [78]. This was also the only major release that supported Vue version 3.

3.4.6 Package manager choice

As I've decided to use Vuetify, I'll need to download the package of this library. This is done using Node.JS packages. According to a definition from W3Schools [79], a package in Node.js contains all the files you need for a module. Modules are JS libraries you can include in your project. These libraries contain functionality, such as Vuetify UI components, etc... There are several package managers that provide the functionality of downloading a library for a JS project, and I will present the most used ones in the following subsections.

npm

A default package manager for Node.js. Besides the default package management functionality, it also offer a custom scripts options, which allows the developer to define and use custom scripts. These scripts is used to automate development tasks, such as building, testing and deployment. One of the main disadvantages is versioning complexity, as managing the package versions and dependency conflicts can be challenging, particularly in projects with complex dependency trees.

Yarn

A package manager developed by Facebook. It tries to address some of the limitations of npm, such as performance issues and versioning complexity. It comes with a deterministic dependency resolution, which helps to prevent the dependency conflicts and ensures consistency across different development environments. This deterministic dependency resolution brings management overhead, as you need to take care of another configuration file. Another disadvantage of yarn are compatibility issues, as although it is compatible with npm, there still might occur occasional compatibility issues or behavior differences between the two.

pnpm

The term pnpm is short for “Performant npm”. It emphasizes efficiency, disk space optimization, and installation speed. The main perk of the pnpm is a shared dependency model, where common dependencies across multiple projects are stored in a single location on disk. It also introduces the deterministic dependency resolution known from yarn, preventing the dependency locking. The disadvantages of pnpm is lockfile handling - due to the deterministic dependency resolution, you have to manage the file holding such configuration. Although pnpm has gained some adoption within the Node.js community, its community support might still not be as high as for npm and yarn. In few cases, compatibility issues with npm and yarn are possible as well.

The decision

All the information on the package managers was taken from the blog post [80] titled “Choosing the Right Node.js Package Manager in 2024”. The choice of the package manager does not directly affect the performance of the application, rather the developers experience. I’ve decided to use pnpm, as it seems the most modern out of the three presented options. It also looks like an overall better version of npm and currently, I’m not that scared of the lower community support, nor compatibility issues, as I’m embarking on a greenfield project.

3.5 Conclusion of the technology choices

In the previous sections, I’ve gone over and decided for multiple technologies and concepts. This section groups all of these choices in one place, summarizing my decisions.

- **Approach:** a web application
- **Framework:** Vue V3
- **UI Component library:** Vuetify V3
- **API:** Composition API
- **SFC:** yes
- **script setup:** yes

3.5. Conclusion of the technology choices

- **TypeScript:** yes
- **Package manager:** pnpm

Design

In the Design chapter, I go over the design decisions I made before starting the implementation. These decisions concern both the architecture of the application, as well as the main functionality of the frontend.

4.1 Communication between helpdesk and Redmine

I've mentioned that the helpdesk will be an independent application. This application has to communicate its requests to the Redmine instance. This communication can be achieved with the help of REST API. The official documentation for this API is available at [81]. I'll source this documentation for the rest of this chapter, but also during the implementation process. The API provides the access to most of the entities available at Redmine.

4.1.1 Authentication

As I've declared in the functional requirements "the customer should see the issues, comments and changes made in both Redmine and helpdesk under his own account". Therefore, I will need to access this API in the name of the user currently using the helpdesk. The API support this with the Authentication feature. The Authentication can be done in two different ways:

- Using the regular login/password via HTTP Basic authentication.
- Using the API key.

The API key is much safer option, as it avoids putting the password in a script. As I want to keep the data of my users as secure as possible. I will therefore be using the API key alternative.

4.1.2 Handling the API key

While reading through a post titled "Best practices for working with API keys in the frontend", available at [82], the first thing that resonated with me is the fact that there are no secrets in frontend. After being compiled¹⁴, frontend is

¹⁴compilation is "the action of taking source code and translating it into an executable program" [83]

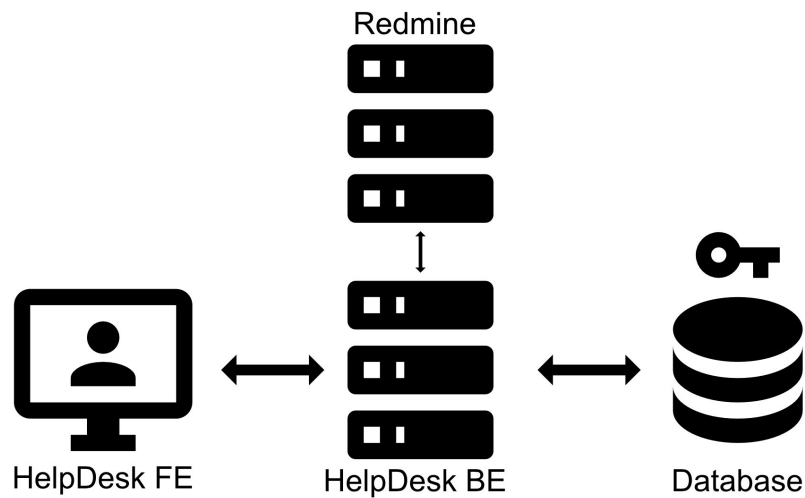


Figure 4.1: Design of the architecture

just a bunch of HTML, JS and CSS files. Therefore, if there's a key that JS needs, it's usually just hardcoded as a variable in JS code. An attacker can simply inspect all the variables in memory and track down the key. Sometimes, it is not even that difficult. The keys are used to call APIs, so it might be enough to open the Network tab in the developer tools of your browser and inspect the API key there.

As I don't want to put the API key at risk, I can't hold it in the frontend. This idea was further supported in the discussion with my colleague Max, who suggested that I should leverage a simple backend that will hold the API key in the database. According to a definition from AWS at [84], the term backend encompasses the data and infrastructure that make your application work. It stores and processes application data for the user.

A diagram of the intended architecture is available in the figure 4.1. The shortcut BE stands for backend, FE for frontend. My friend Viktor helped me with the graphical side of the diagram.

As visible from the diagram, the database holds the API key. User accesses the HelpDesk FE. The FE communicates with the HelpDesk BE, which is the central part of the architecture, as it communicates in three ways:

- Process the requests from frontend, serve the data for frontend.
- Store and retrieve the user data, including the API key, from the database.
- Request the data from Redmine, with the use of API key, retrieved from the database.

My frontend will not communicate directly with the Redmine, but rather with the backend. Two distinct cases of requests can happen between frontend and backend:

1. **FE request Redmine data** - in this case, the BE should authenticate user, retrieve his API key token from the database, and request the data

from Redmine with the user's API key. In this case, the BE functionality is basically rerouting the request between FE and BE, while attaching the key required for Redmine authentication.

2. **User-based requests and authentication** - the FE either request the data about the current user, which are stored directly in the database, or performs a login / registration request. No communication with the Redmine server is required in this case.

This new architecture clashes with the idea that the solution should be 1-layered, resulting in a more complex deployment. The multi-layered architecture is a necessity though, as I want to handle the API key securely. The API key will be required from the user during the registration, more on that in the following chapter, where I'll present the implementation process of the authentication.

4.1.3 A remark on the OAuth 2.0

A possible upgrade to the API key approach would be the use of OAuth 2.0. It is a modern standard for securing access to APIs. A simplified guide on this standard is available at [85]. It works on the basis of granting applications access to their account. Unfortunately, at the time of writing this thesis, there is no official technical support for Redmine being an OAuth2.0 provider. An official feature request titled "OAuth2 support for Redmine API Apps (OAuth2 Provider)" exists on the Redmine features request forum [86]. This request is over seven years old at the time of writing this review, yet was marked as a "Candidate for next major release" by a Redmine project release manager about two years ago.

4.2 Choosing the backend technology

I want to keep the backend as light as possible. I also want to keep the technology stack as low as possible. I've therefore decided to choose Express framework as the technology for my backend. It is "a minimal and flexible Node.js web application framework that provides a robust set of features for web or mobile applications". According to Express documentation at [87], Express provides a myriad of HTTP utility methods, which makes creating a robust API quick and easy. It also matches the technology I'd be using when communicating with Redmine via the REST API.

For the database technology, I've decided to go with SQLite, a small, fast, self-contained, high-reliability SQL¹⁵ database engine. According to its home page [89], it is also the most used database engine in the world. It is built into all phones and most computers and comes bundled inside countless other applications that people use every day. This database matches my use case very well, as I want something lightweight and at this point, I plan on storing only the user data, and these shouldn't be accessed that often.

¹⁵SQL stands for Structured Query Language. It is "a programming language for storing and processing information in a relational database. A relational database stores information in tabular form, with rows and columns representing different data attributes and the various relationships between the data values. You can use SQL statements to store, update, remove, search, and retrieve information from the database." [88]

4.3 UI design

This section presents the thoughts behind designing the important UI parts of the helpdesk.

4.3.1 Homepage and the layout of the application

At the beginning of the design process, I was unsure what to put on the homepage. I knew it was an important decision as homepage is the first page users encounter after logging into the application. I had a clear vision of the functionality I wanted the whole application to have: the ability to report an issue, view the details of a reported issue, and view the list of all reported issues. Yet, since I have very little background in the User Interface design, I didn't exactly know how to combine all these functionalities. I thought that designing the homepage screen would solve my problem, therefore I've decided to start the design process with this screen.

Despite reading several posts about screen design, I wasn't sure how to start. After some contemplation, an idea struck me. What about accessing another product of my company, a product which has two strong bounds to my application:

- Same component framework - Vuetify.
- Same focus group - the customer's of the company I work for.

The user documentation for this product is publicly available at [90]. It also includes several images showcasing the UI of the application. Having analysed its UI, I've found out that the UI contains of three main components - an app bar, a navigation drawer and the main screen. This division is quite smart, as each of the components handles its own area:

- **The app bar** - contains a menu that lets the user set up personal preferences and overall configuration of the application.
- **The navigation drawer** - stacks all the main functionalities of the application, serving as navigation - clicking an option in this drawer redirects you to the screen of the chosen functionality.
- **The main screen** - depicts the currently chosen functionality.

Having seen this division, I've decided to choose the same layout for my application. The layout of my application will consist of:

- **An app bar** at the top of the application - this app bar will allow the user to change the language mutation of the application and log out. If more personalization features of the app get implemented in the future, they'll be stored here as well.
- **A navigation drawer** on the left side of the application - this drawer will contain a home page link, a big "Report a bug" button and links to issues tables and reports.
- **A main screen** - this doesn't come as a surprise. The screen will serve as the place where the main content is served to the user.

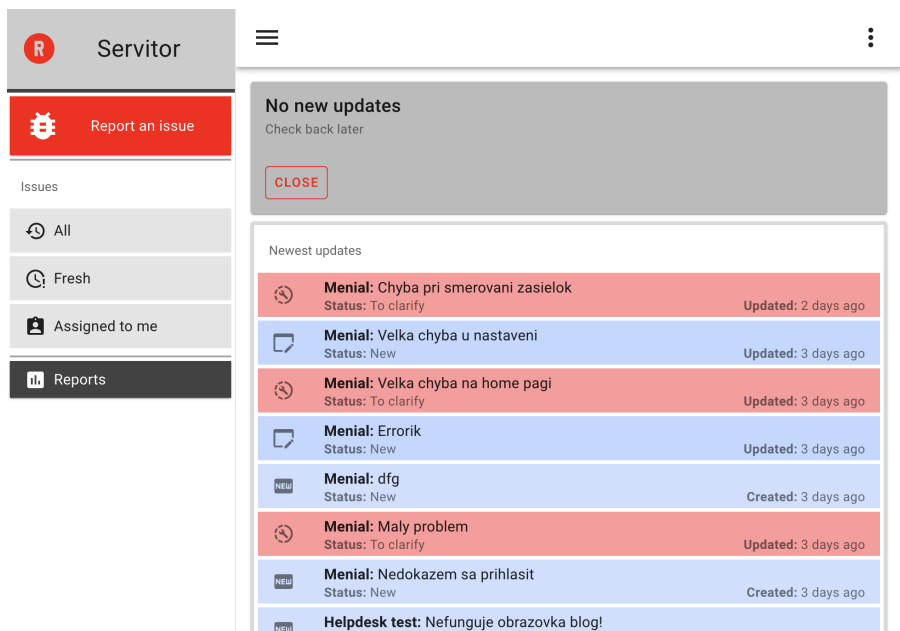


Figure 4.2: Final layout and homepage

Trying to design a home screen for my application, I've non-intentionally managed to define its layout. This, however, also helped in the the homepage design question. As I've placed all the main functionality to the navigation bar, there is not a single functionality I explicitly need to include on the homepage. This leaves me with some maneuvering space. In the end, I've decided to include just two simple lists on the homepage screen:

- **Updates since last login list**, displaying the issues that got changed since the last user's login. In case there are no such issues to be displayed, a information about this fact will be shown to the user instead. This information component will be closeable by a button, allowing the user to remove it from the homepage for the current session. This will provide more space for the second component of the homepage.
- **List of up to 10 issues reported by the current user**, sorted by the last updated one. The of this list might duplicate a bit the data in the first list, but that in my opinion is not that much of a problem. Where the first list should serve more as a notification about the updates that took place, this list should serve more as an **issues you've reported recently** showcase.

The described layout and homepage, created with the help of Vuetify, is visible in the figure 4.2.

4.3.2 Report an issue screen

When creating an issue as a customer in the Redmine UI, the user can set multiple fields, as I've already touched on in the introduction to Redmine

section. In the following list, I will address these fields and also present my decision on whether they should be visible on the 'report an issue' screen of my helpdesk system.

- **tracker** - this field is debatable. On one hand, you can say that the customer can decide whether the problem is a bug or a new feature. On the other hand, they usually have almost no technological knowledge about the software in question. Due to this fact, I'm not keeping the field in the form.
- **subject** - the subject of the issue is a core field and I'm definitely keeping it in the form.
- **description** - the description is also an important field and I'm keeping it in the form.
- **status** - in my opinion, it makes almost no sense letting the user select the status of the reported issue. It should be defaulted to a "New" value, or its equivalent for the Redmine instance.
- **priority** - While discussing this field, my supervisor made the remark that from his experience, the customer often sets the priority to "Urgent", just because they want the issue to be solved as soon as possible. This classification does not always match with the real priority of the task and the project manager can often have a better insight. That is why I've decided not to keep the priority field in the form.
- **assignee** - this field is in my opinion a prime example of the field that shouldn't be set up by the user, but rather by the project manager.
- **parent task** - there is no need letting the customer decide what is the parent task of the newly created issue.
- **start date** - doesn't make sense to fill by hand; Redmine defaults this value to the day when the issue was created, which is good enough for me.
- **due date** - in case I'd be handling "New feature" requests with the helpdesk, this field could be interesting. Since I'm mostly focused towards bugs now, I'm not keeping it in the form.
- **%Done** - this field is in my opinion purely IT team focused. It should also start at 0% for a new issue, therefore I don't see any advantage of keeping it in the form.
- **checklist** - this functionality allows adding TO-DOs to an issue. The TO-DOs are checkable as DONE. These TO-DOs are in my opinion much more focused towards the developers, therefore I'm not including them in the form.
- **files** - files are a crucial field within the form. They allow the customer to include a screenshot of their problem.

I've talked about all the default fields of a Redmine issue in the configuration I was working with. I've decided to keep the subject, description and files field in the form. The final field that I need to add is the project, under which the issue should be categorized.

I'd also like to remark on the idea of **configurable** fields. The fields like tracker, status and assignee in my opinion belong to this **configurable** category. This **configurable** flag should group the fields that depend on the project manager's decision. The use case would be as follows - for project A, the developer John should be the user defaultly assigned to a new issue. For project B, it should be Patrick. I'll discuss the idea of **configurables** and how to implement them in the following section of this chapter.

4.4 The configurable fields

While discussing the "Report a new issue" screen, I've presented the idea of configurable fields. As I see it, the configurable fields should be a subset of a bigger set. Let's call this bigger set **Helpdesk Configuration**, or **HC** for short. The **HC** configuration should "personalize" the helpdesk application, so that it works correctly with more than one instances of Redmine. This requirement sparks from the configurable essence of Redmine. Suppose I'd implement the helpdesk in a way that each time an issue created on the "Report a new Issue" screen, it automatically receives the "New" status. What if the instance of Redmine doesn't have the "New" status? How should the helpdesk function? That's when the **HC** comes in. Instead of defaulting to the "New" status, the helpdesk should look at the values in the **HC**, and assign the pre-configured status to each created issue. This also allows for a further customization of the helpdesk. The question now is, how should such functionality be implemented. I can see few possible implementation approaches to the **HC**:

- **The .env file** - this is the most "programmer-focused" approach. The values would be set directly in the environment files of the project and would be read during the deployment of the application. No explicit database besides the .env files would be required. The values would, however, be hard to modify and it also isn't a very elegant approach, therefore I won't be opting for it.
- **Helpdesk based configuration** - modifying the configuration would be available directly in the helpdesk application. This approach would be probably the best from the user perspective, but would bring non-desired complexity to the system, as I'd have to take care of admin role and privileges for the helpdesk application. In my current plan, all the users of the helpdesk will have the same "role", and therefore they'd be able to access the same pages. This approach would leverage the database of the helpdesk itself.
- **Redmine module** - the third option is implementing a Redmine plugin that will add a new module to the Redmine instance. When you would add this module to your Redmine and enable it for the project, you would be able to configure the HC directly in the Redmine UI. The disadvantage of this approach is the call for completely new technology, the Ruby

language. The database used with this approach would be the default database used by Redmine.

In my opinion, the best of these three approaches is the Redmine module one. Despite the fact that it asks for a plugin implementation, it is in my opinion the cleanest and best option out of the three presented. It also sticks to the idea presented earlier, that I want to hold as much data as possible in the default Redmine database.

4.5 Handling the Helpdesk Configuration

The Redmine module plan is in my opinion a good idea, yet I've agreed with my supervisor that the complexity of it exceeds the intended scope of my thesis. Therefore, I only present the plan on how to work with HC during the different phases of the project.

4.5.1 Phase 1 - prototype deployment

I will deploy the prototype of my solution before the thesis hand in date. In this phase, the HC will be hardcoded in the implementation. This will allow the solution to function with one instance of Redmine. This limitation is not a problem for the prototype phase.

4.5.2 Phase 2 - Redmine plugin

In the second phase, I'll transform the solution to the Redmine plugin approach. This phase will include:

- **Implementing the Redmine plugin.** – The first step of the phase will be the implementation of the Redmine plugin. The plugin implementation will consist of two parts - the Redmine part, written in Ruby, and the database migrations, which will modify the database.
- **Deploying the new plugin to Redmine instance.** – The second stage of the Phase 2 should be installing the plugin. This will provide the required changes in the database of the Redmine instance and also add the module to the Redmine instance.
- **Enabling the plugin for a project.** – Once the plugin is successfully installed, the project manager will be able to enable it for the desired projects.
- **Setting the HC values for a project.** – In the projects, where the module will be enabled, the manager will have to fill up the required fields of the Helpdesk Configuration.
- **Refactoring the application to work with the module.** – The final step of Phase 2 will be the refactor of frontend and backend to a module-based approach. This means removing the hardcoded values from the backend and frontend code and implementing the HC requests in the existing code.

Overall, the most time consuming operations of the Phase 2 will be implementing the plugin and refactoring the application. The other parts seem rather straightforward.

4.5.3 Phase 3 - Extending the Redmine plugin

There are options for several improvements to the Redmine plugin once the plugin is in use. The most interesting one in my opinion is the confirmation email functionality. As I've mentioned in the state-of-the-art review, all three existing helpdesk plugins for Redmine allow sending the confirmation email to the address reporting the issue. This could be also implemented by my plugin solution - either as the part of the plugin handling the HC, or as a completely new plugin.

Another interesting idea for the plugin extension are custom emails for the Servitor users. In the default Redmine configuration, the user gets notified about the change in the issue by an email that has a standard body, with links leading to Redmine. The plugin might be able to change this swapping the default Redmine email templates for a custom ones with links pointing to the helpdesk.

Implementation

In this chapter, I present several interesting areas and concepts I've encountered during the implementation process. The chapter does not cover the whole implementation process. Instead, it tries to describe the most important decisions made during the implementation, so that the reader can imagine how I tackled the obstacles.

5.1 Start of the implementation

At the beginning of the implementation, I already knew the technologies I'd like to use. I had to initialize the frontend project, initialize the backend project and initialize the database.

5.1.1 Initializing the frontend

The initialization of the frontend was quite straightforward. I've accessed the official "Get started with Vuetify 3" guide available at [91] and I've initialized the project with the `pnpm create vuetify` command. This command creates an interactive dialogue. I've answered the dialogue's questions in the fashion that matches my technology choices presented at the end of **Technological stack** chapter. The command created the Vuetify project based on the chosen configuration.

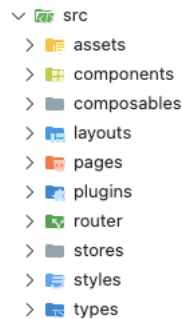


Figure 5.1: Frontend directory structure

My current frontend directory structure is visible in the figure 5.1. To achieve this structure, I had to do three changes to the pre-generated structure:

- Creating the `composables` directory. This directory will group the shareable logic functionality of the application.
- Renaming the `store` directory to `stores` directory. The new name in my opinion fits better.
- Creating the `types` directory, which will group the types usable by TypeScript language.

Having a well standardized project structure is in my opinion vital for almost any software project. I'll therefore stick to this predefined structure for the rest of my implementation.

5.1.2 Initializing the backend

Initialization of the backend was rather simple. I've read over the Express JS documentation available at [87] and tinkered a bit with the configuration. As I've decided to use TypeScript for the FE, I've also decided to use it for the backend. This will keep the technological stacks similar and hopefully the whole project easier to maintain.

5.1.3 Initializing the database

I've initialized the `SQLITE` database structure with a simple SQL script that got called once the backend connected to the database. The database itself got created by another function call of the JS code. As its not really important, I won't be going into more detail on this topic.

5.1.4 A remark on the IDE choice

The shortcut IDE stands for Integrated Development Environment. According to a definition sourced from Amazon at [92], the IDE is “a software application that helps programmers develop software code efficiently. It increases developer's productivity by combining capabilities such as software editing, building, testing and packaging in an easy-to-use application.” The IDE makes the programmers life easier. There are several IDEs, a programmer can even not use one and be productive.

I've started implementing the application in a JS-focused IDE called Webstorm. That was due to the fact that I've implemented several JS tasks in this IDE during my studies. When developing the frontend using this IDE, I've soon come to a realisation that it doesn't properly support Typescript and Vue, the technologies I've intended to use. I've therefore decided to try another IDE called Visual Studio Code. This IDE is not focused on a single language, rather supports multiple. From my experience, it supported the Typescript and Vue development much better than the Webstorm. Due to this fact, I would advise using Visual Studio Code for similar use cases.

5.2 Authentication

In the previous chapter, I've declared that I want to communicate with Redmine as an existing Redmine user. I've also decided to use API key as the Authentication approach. This API key will be stored in the database, and utilized by the backend for communication with Redmine.

Yet, I need another layer of authentication - between the FE and BE. I'll further call it helpdesk authentication. This authentication will leverage the JWT concept, where JWT stands for JSON Web Tokens. The JWT is an open standard that defines a compact and self-contained way of securely transmitting information between parties as a JSON¹⁶ object. The parties in question are the server - backend and the client - frontend. According to Auth0 documentation [94], the JWT contains all the required information about an entity to avoid querying the database more than once.

5.2.1 Registration

During registration, the user will have to provide three distinct data values - email, password and an API key. The API key is accessible under the user's profile in Redmine. The user will be asked to provide it just once - at the time of registration. These three credentials will be passed from FE to BE. The BE will perform two validations:

1. Check, whether the email is not already in use.
2. Check, whether the API key is valid. This validation is done by querying the Redmine API.

If any of the validations fail, the registration process is marked as unsuccessful and finishes. If the validation checks pass, a new user entry has to be added to the database. The entry contains:

- **Email**
- **Password** - the password provided by the user, in a hashed format. The hashing will be done with the use of an existing cryptography library.
- **API key**
- **Redmine ID of the user** - this data will be queried in the validation request and saved for a future use.
- **Last login date** - initialized to year 1970, this value will be updated each time the user logs in. Its value can be leveraged to show the most important issues on the frontend.

Once this entry is created, the registration process is successful. The FE is informed about the success with a 201 HTTP code, which indicates "Created" status. The HTTP codes are used to indicate whether a specific request has been successfully completed, as per their Mozilla Developer Network documentation available at [95]. For my use case, they will serve as an information handler between the backend and frontend.

¹⁶JSON is "a light-weight data-interchange format" [93]

5.2.2 Login

Login is another important step in the authentication process. The user enters email and password to a form on the FE, the FE sends the data to the BE. The BE uses already mentioned cryptography library to compare, whether the password matches the hashed one. If the check succeeds, the login is successful. As a response to a successful check, the BE sends back a 201 HTTP response code along with the session data. The session data consist of:

- **Email**
- **Redmine user ID** - required by FE logic, e.g. to decide whether the task is assigned to the current user.
- **Redmine instance URL** - required by FE to enable redirecting to a specific Redmine page.
- **Last login date** - for FE logic.
- **Expiration** - indicating how long the token is valid.
- **JWT**

All this data should be saved by the FE. In my FE implementation, I'm using Pinia stores. According to its official documentation available at [96], a store allows the developer to share a state across components and pages. That's exactly what I need in my application, as I want the data acquired by the login to be shareable across the whole application. When using Pinia, the developer can define multiple stores in his application, where each store handles its own data. A successful login initializes the data in the **user store** for my frontend.

5.2.3 Using the JWT in communication

After logging in, the user is allowed to access the “protected” part of the frontend. Where the “non-protected” part is simply the Login and Registration page, the “protected” part of the application is every other page of the application. A further discussion about the protected and non protected parts of the frontend will be in the section about Routing.

All the operations in the protected part of the application, either requesting or modifying the data, require the JWT to be accepted by the BE. When performing these protected operations, the FE accesses the user store, retrieves the JWT token, bundles it with the request and sends this request to the BE. The BE validates the provided JWT and if the token matches, handles the operation in the name of the user. This process might seem a bit complicated, yet thanks to already mentioned store, it is quite simple from the frontend part.

In the backend part, the Express framework offers a middleware functionality, mentioned in their official documentation at [97], to make the validation and similar processes easier. The middleware functionality has the access to both the incoming request and the potential response, even before these two are processed by the intended receiver. For my use case, I've leveraged the middleware functionality in the following fashion:

- Developing the **authenticate token middleware** function. This function accesses the authorization header¹⁷, the place where the FE stores the JWT. After retrieving the JWT, the function performs the verification. In case the verification fails, the middleware function can directly send an invalid response code and end the processing of the request. In case the verification is successful, the middleware function can bundle additional data and pass the request and response for further processing to the intended receiver.
- Declaring which BE functionalities should be protected by the mentioned **authenticate token middleware** function. In Express, this is done by simply referencing the middleware function in the functionality declaration.

The process mentioned above has a good impact on the overall quality of the code - the developer declares the authorization process at a single place and references it in the other parts of the application. Any further changes to the authorization process can be done at a single place, while having impact on the whole application. My application uses the npm library **jsonwebtoken** for all the JWT logic.

5.2.4 Refreshing the JWT token

The BE provides the expiration time of the token along the JWT. An expired token doesn't provide anything, therefore the FE has to control the expiration and request a new token once the old one nears the end of its lifetime. While handling this task, I've sourced a blog post by Jason Watmore, available at [99]. In this post, titled "Vue 3 + Pinia - JWT Authentication with Refresh Tokens Example & Tutorial", Jason presents a way of handling and refreshing the JWTs in a project using Vue3 and Pinia. As that is my technological stack, I've taken inspiration from this blog post and implemented a simplified approach of his design for my frontend.

The idea is based on timeouts, implemented inside the user store and visible in the figure 5.2. When user logs in, a countdown is started. This countdown runs out approximately thirty seconds before the expiration time of the token. This action fires a request to backend, asking for a new JWT. Once BE responds with a new JWT, the session data is updated and the countdown timer is reinitialized. A function clearing the session data, along with stopping the refresh token timer, is also provided by the user store. This function is called once the user logs out of the helpdesk.

A better approach to refreshing the JWT would be using a refresh token in the process. This would introduce another layer of security, but also another layer of complexity to the application. I've decided to stay away from the refresh tokens for now, as I'm implementing only a prototype at this point. In the future, it would be smart to consider refactoring the refresh token process.

¹⁷a header is "a part of a request, it lets the client and the server pass additional information" [98]

```
5 export const useUserStore = defineStore("user", {
6   state: () => ({
7     sessionData: ref<UserSession | null>(null),
8     refreshTokenTimeout: ref<number | null>(null)
9   }),
10  actions: {
11    login({loginData} : {loginData : LoginData}) {
12      return useBackendApi().login({loginData}).then((data) => {
13        this.sessionData = data;
14        this.startRefreshTokenTimer();
15      });
16    },
17    clearSessionData() {
18      this.sessionData = null;
19      this.stopRefreshTokenTimer();
20    },
21    refreshToken() {
22      useBackendApi().refreshToken().then((newSessionData) => {
23        if (!this.sessionData) {
24          throw new Error("Trying to access non existent session data!");
25        }
26        this.sessionData.accessToken = newSessionData.accessToken;
27        this.sessionData.expiresInSeconds = newSessionData.expiresInSeconds;
28        this.startRefreshTokenTimer();
29      })
30      .catch((error) => {
31        console.error(`Error occurred while refreshing token ${error}`);
32      });
33    },
34    startRefreshTokenTimer() {
35      if (this.sessionDataWithCheck.expiresInSeconds < 30) {
36        console.error("Expiration smaller than 30 seconds!");
37        return;
38      }
39      this.refreshTokenTimeout = window.setTimeout(
40        this.refreshToken,
41        (this.sessionDataWithCheck.expiresInSeconds - 30) * 1000
42      );
43    },
44    stopRefreshTokenTimer() {
45      window.clearTimeout(this.refreshTokenTimeoutWithCheck);
46    }
47  }
48 }
```

Figure 5.2: JWT refresh logic

5.3 Formatting text in Redmine

The official Redmine User guide [100] declares that Redmine supports two syntaxes of formatting based on configuration - either **Textile** or **Markdown**. I've found out that the first Redmine instance I'd be synchronizing with uses Textile, therefore I've decided for the textile support in my helpdesk application. I've got to handle two cases - generating the textile and rendering the textile. I'll discuss each of these in a separate subsection.

5.3.1 Generating textile

```
1 <template>
2   <main id="sample">
3     <Editor
4       v-model="modelValue"
5       tinymce-script-src="/tinymce/tinymce.min.js"
6       :init="init"
7       output-format="html"
8     />
9   </main>
10 </template>
11
12 <script setup lang="ts">
13   import Editor from "@tinymce/tinymce-vue";
```

Figure 5.3: TinyMCE configuration

It is common to use a text editor to enhance the experience of writing the text. The Redmine instance in question offers the user an improved text editor, which is a bit more user friendly than the default one. I've talked about this with my supervisor and we came to a conclusion that offering such editor in the helpdesk could improve the user experience. I've performed analysis and found out that the improved editor in question is called **TinyMCE** editor. According to the official page of this project [101], it is a WYSIWYG editor, and more than hundred million products use it. The TinyMCE boasts over twelve integrations with other technologies, such as already discussed Vue, Svelte, and React.

I've decided to implement a reusable component that will offer TinyMCE editor for my frontend. The implementation of this component will leverage the already mentioned Vue integration, known as **tinymce-vue** npm package. According to the technical reference of the mentioned integration, available at [102], the two possible outputs are either html or text. Yet, the Redmine is expecting textile. To solve this issue, the BE will perform the html to textile transformation. I've employed the **to-textile** npm package on the backend to perform this transformation.

A remark on integrating the TinyMce Vue editor

Using the tinymce-vue integration out of the box solution didn't work for me, as it required a license key to run the editor. I didn't want to get constrained by any licenses, therefore I've opted for an alternative approach - self hosting the editor. I've sourced few existing blog posts and websites on how to self host the editor in your own project with no success. In the end, a comment by the user larrykkk, under an issue [103] titled "How can I self host this" on the

GitHub repository of the official Vue integration helped me out. I had to do the following:

- Install the official `tinymce-vue` package with the help of `pnpm`.
- Copy the content of this package into the `public` directory of my frontend application.
- In the Vue code, link the source script of the `tinymce` from the `public` directory. The configuration of the editor is visible in the figure 5.3.

A user can further extend the functionality and interface of the editor with the help of plugins. For my project, I've performed the following:

- **Wrap the editor in a custom Vue component.** This groups the whole configuration of the editor in a single place and allows the user to simply include the editor anywhere in the project, working out of the box.
- **Install the initial plugins for the editor.** I've installed three plugins, namely `lists`, `links` and `codesample`. These three plugins match the functionality of the editor used on Redmine and also the intended functionality of the helpdesk.
- **Design the toolbar.** The toolbar of the editor is heavily customizable. I've designed it in a way that matches the Redmine editor but also offers all the functionalities the helpdesk user would find useful.

5.3.2 Rendering textile

To render the textile to HTML presentable by the FE, I've employed another textile focused npm package called `textile-js`, available at [104]. This fully-featured textile parser seemed to work rather well for my use case, despite not being updated for more than three years at the time of writing this thesis. The transformation happens once again on the BE. The idea behind moving the transformations to the backend is the fact that the BE should handle most of the processing logic of the application.

Not being able to find an npm package that would single handedly perform the transformation both ways was sad for me, and I'll have to consider revisiting the transformation of texts in the future.

5.4 Requesting files from Redmine

The Redmine's issues, as well as the journals of the issues, can contain attachments - the files user uploads along with them. These attachments can often be images, describing the issue visually, and as I want my application to be **just a simpler UI** for the Redmine, it should be able to show these files to the user. Yet, due to the fact that the Redmine API key is not handled by the FE, accessing these files has to be routed through the BE. I'll discuss in steps how I managed to get this attachment routing to work:

1. **The FE requests an attachment from the BE.** – The attachment is defined by the its identifier and filename. The request’s `Content-Type` header is set to `application/octet-stream`.
2. **The BE request the attachment from Redmine.** – The response type option of the request configuration is set to the `arraybuffer` value. Sourcing the Mozilla page on `arraybuffer` at [105], the `arraybuffer` is used to represent a generic raw binary data buffer. This option therefore tells `axios`¹⁸ to await a data buffer rather than a standardized object. Setting this option was crucial, as without it, the routing didn’t work.
3. **The BE sends the data received from Redmine to FE.** – When sending this response, two headers have to be set up:
 - `Content-Type` header, getting it’s value from the Redmine response header `Content-Type`. This basically means that the BE looks what type of data Redmine sent and informs the FE that the data routed is of such type. An interesting problem I’ve encountered while working with this header was that despite the fact that the IDE tells the programmer to access the header `Content-Type` on the provided response, only lower cased version spelled `content-type` works.
 - `Content-Transfer-Encoding` set to `binary`. This indicated to the frontend that the incoming data is not a standardized text, but a data buffer.

When both these headers are set up, all that’s left to do is using the response with the `send` function, providing the Redmine response data as the argument.

4. **The FE receives the data from BE.** – In case of a success, the received entity is of interface `Response` from the Fetch API. Looking at the official Mozilla documentation at [106], the `Response` interface offers a `blob` method, which takes the response stream - the mentioned data buffer and reads it to completion. This enables the frontend to use fetch images and files as an instance of `Blob`, documented at [107]. An instance of `Blob` represents a “file-like” object of immutable, raw data, which can be read as text or binary data. In my further FE implementation, I leverage these `Blob` objects to render the images and allow the downloading of the attachments.

5.5 Uploading files to Redmine

The user also needs to be able to upload an attachment along with a newly reported issue, or a comment of an issue. As you can guess, the whole upload process has to be routed through the backend, due to the API key. The guide on uploading the files via the Redmine Rest API is available in the “Attaching files” section of the Redmine API documentation, available at [81]. The process for uploading an issue with an attachment is done in two steps:

¹⁸`axios` is a HTTP client, as per [105]

5. IMPLEMENTATION

```
201 const upload = multer({ dest: "uploads/" });
202
203 app.post("/uploads",
204   jwt.authenticateTokenMiddleware,
205   upload.single("file"),
206   (req: AuthenticatedRequest, res) => {
207     const params = queryParams.parseParsdQsToArray(req.query);
208     const filenameParam = params.find((p) => p.title === "filename");
209     if (!filenameParam) {
210       res.status(422).json({
211         message: "Missing filename query parameter!"
212       });
213       return;
214     }
215     if (!req.file) {
216       res.status(422).json({
217         message: "Missing file!"
218       });
219       return;
220     }
221     redmine.uploadFile(req.apiKey, req.file, filenameParam.value)
222       .then((data) => {
223         res.status(201).json(data);
224         fs.unlink(req.file.path, (err) => {
225           if (err) {
226             console.error(`Error while deleting file: ${err}`);
227           } else {
228             console.log(`File ${req.file.path} removed successfully.`);
229           }
230         });
231       })
232       .catch((error) => {
233         console.error(error.message);
234         sendUnexpectedError(res);
235       });
236   });
```

Figure 5.4: Processing file with multer middleware

- **Upload the file with a request to uploads endpoint.**¹⁹ – The request body should be the content of the file and the `Content-Type` header should be set to `application/octet-stream` value. This request should have a query parameter²⁰ with key `filename` and value the filename of the file. In case of a successful upload, the Redmine responds with a token for the uploaded file.
- **Create the issue using the upload token** – Send the request to create and issue, while indicating in the body of the request that the file should be treated as the attachment of the issue. The attachment reference is

¹⁹endpoint is “the place where the APIs send the request and where the resources live” [108]

²⁰“Query parameters are a defined set of parameters (key-value pair) attached to the end of a URL used to provide additional information to a web server when making requests.” [109] An URL is the address of a unique resource on the internet. It is one of the key mechanisms used by browsers to retrieve published resources, such as HTML pages, CSS documents, images, and so on, as per [110]

created with the use of token from the first step. The user has to also declare the filename and content type attributes of the attached file.

The workflow of uploading the files in my helpdesk works as follows:

1. **The FE sends the upload file request to BE.** – No special headers are required for this case. On FE, I'm using the `FormData` interface to send the file. As written in the `FormData` documentation by Mozilla at [111], this interface provides a way to construct a set of key/value pairs representing form fields and their values, which can be sent via several JS methods. A programmer can pass a `Blob` as the field, essentially attaching the content of `Blob` to the request. In my case, I'm appending a `Blob` representing the uploaded file, along with its filename and key `file` to an instance of a `FormData`. This `FormData` is later passed as the body of the request to the BE.
2. **The BE processes the upload file request.** – This is done with the help of `multer` npm package, documented at [112]. It is a middleware, primarily used for uploading files. The usage is quite simple, visible in the figure 5.4. First, I had to install and import the `multer` package. Afterwards, I've initialized it with a destination directory, as visible on the line 201. The third parameter of the `app.post` call on line 203 represents passing the call of the `single` function on line 205 as the middleware function for the POST uploads endpoint. I'm passing the `file` attribute to this call. The value matches the key `file` passed to `FormData` by FE. The `single` call, along with the displayed `multer` configuration, stores on the file system in the `uploads` directory. This allows me to access the `file` property of the `req` instance on lines 215 and 221, respectively. The displayed code also showcases that I'm accessing and validating the query parameter `filename` on lines 207-214. The filename, along with the file, is passed to `uploadFile` method on line 221.
3. **The BE sends the file to Redmine.** – This is done by the function `uploadFile`, mentioned in the above step. The request to Redmine has the `Content-Type` header set to `application/octet-stream` value and the body as an `fs.ReadStream` instance, which is instantiated by calling the `createReadStream` on the file argument. The `fs` is a Node.js module documented at [113]. It enables the programmer to interact with the file system. The file is basically read from the file system and passed to the request. Once this part is finished, the BE removes the sent file from its file system, freeing up the space.
4. **The BE reroutes the token to the FE.** – After a successful upload, the Redmine responds with a token for the file. BE routes this token to the FE.
5. **The FE receives the token.** – FE stores the received token in memory. The token can be later passed to the body of a create request, linking the created entity and file. The create request also requires the filename and content type of the file. These attributes can be accessed on the `Blob` instance, mentioned in the first step of this flow.

I've delivered quite a comprehensive overview of the file upload process. It may have been tiring to read through, yet I chose to include it as it represents one of the more complex aspects of my implementation. It could also benefit other developers, who are facing similar challenges with the file uploads.

5.6 Handling the images in textile

The textile allows including the images as a part of the text, while holding the source URL of these images. The problem is that this URL might not be accessible all the time. In the following subsections, I'll discuss this problem from two points of view.

5.6.1 Rendering the text containing images

Suppose the project manager responds to a customer on Redmine, and decides to include an image in the text of his response. As a result, an attachment is created on Redmine with a source URL `X`, and the image tag is inserted into the created textile code, with the `src` attribute of this tag pointing to `X`. The issue arises when this text is rendered for the customer on Redmine. While the backend correctly converts the image tag to HTML, the `src` URL remains `X`, which is inaccessible on the frontend because it refers to a resource located on Redmine and requires authentication to be accessed.

As mentioned, I've already implemented a way of routing the attachments from Redmine to the FE via BE. The problem is that images routed in such way are only accessible under a new dynamic URL, which gets initialized at the time of the `Blob` initialization. Therefore, I need to perform the `src` attribute remapping for the texts rendered on the FE.

I've implemented this remapping with the help of a mapping array. The logic is quite simple - for each attachment, when initializing its `Blob`, create a pair of old URL - `Blob` URL. Go over all the texts that should get rendered, and replace the old URL by `Blob` URL in them. Although this approach looks rather simple, it works very well for the specified use case.

5.6.2 Creating the text containing images

Now, let's consider the reverse situation. The customer uses the helpdesk's TinyMCE editor to report a problem and pastes an image into the text. The image is rendered in the editor on helpdesk, and once the text is sent to Redmine, the image is sent to Redmine in its base64 representation. Once the project manager accesses such comment on Redmine, he doesn't see the rendered image, but only its base64 text representation. To solve this issue, I've come up with a two-phase plan:

- In the first phase, disable the pasting of images into the editor on help desk. This will force the users to use the upload file functionality, which works fine.
- In the second phase, reprogram the logic of the TinyMCE editor paste option, so that it integrates with the upload image to Redmine. The plan is that pasting an image to the editor would also upload it to Redmine

and hold the token. Once the user submits the text, the frontend would remap the content of the text, so that it matches well the uploaded file and is displayed well on both platforms. This part won't be implemented in the prototype, as it is quite complex.

5.7 Routing in the frontend

```

21  router.beforeEach(
22    (
23      to: RouteLocationNormalized,
24      from: RouteLocationNormalized,
25      next: NavigationGuardNext
26    ) => {
27      const publicPages = ["/", "/create-account"];
28      const authRequired = !publicPages.includes(to.path);
29      const userStore = useUserStore();
30
31      if (authRequired && !userStore.isLoggedIn) {
32        next({
33          path: "/",
34          query: { returnUrl: to.fullPath }
35        });
36      } else {
37        next();
38      }
39    });

```

Figure 5.5: Vue Router Navigation Guard

According to Vue's official documentation of Routing at [114], using the officially supported Vue Router library is recommended for most SPAs. As I've got almost no experience with the frontend development I've decided to go the recommended way. The usage is quite straightforward, as described in the docs at [115]. The user defines the base configuration for the router - defining what URL path should be mapped to what component. In the Vue component, the user can access the `useRoute` and `useRouter` functionalities to navigate the user to a different path.

As I got my whole application generated by the call of `pnpm create vuetify` command, I've also got some predefined configuration. This configuration includes the file system based routing. The file system based routing eliminates the need for the developer to explicitly set up the basic routing configuration. The developer only needs to create Vue files within a main `pages` directory. Each Vue file within this directory corresponds to a specific route. The file's content serves as the component accessible at the route, while the filename itself defines the route.

Another functionality of the Vue Router are the Navigation Guards. The developer can define a function that gets executed whenever a navigation is triggered. This function has the ability to cancel or reroute the navigation. My usage of this functionality is available in the figure 5.5. I'm declaring an array of public pages. In case the user tries to access a non public page without being logged in, he gets redirected to login. Otherwise, he is free to proceed. I see these navigation guards as a quite simple, yet strong functionality.

5.8 Internationalization of the application

I've declared that the application should support multiple language mutations in the non-functional requirements for my application. To achieve this, I've used the Vue I18n plugin. It is an internationalization plugin focused on Vue. I've used it as an npm package. The full documentation of this plugin is available at [116]. According to this official documentation, it is easy to use. I can only agree with this statement, as using it during the implementation was almost effortless.

The plugin defines two important functions - one for template part of the Vue code, one for the JS part, used usually in the script part of the Vue component. Calling these functions with a single parameter - the phrase to translate - is all the programmer has to do to internationalize the phrase in the Vue component. There is, though, second part of the job to be done - provide the translations of these texts. The translations are held in separate files. My frontend project had one file per one language at the time of writing this section. To be honest, I haven't manually modified the content of these files even once. Instead, I've leveraged a plugin for the Visual Studio Code IDE called **i18n Ally**. This plugin offers several features, such as highlighting the phrases with missing translations and an on-hover functionality that allows the programmer to provide the translation directly in the Vue component, without the need to open the translations file. I would recommend this plugin to any programmers using Visual Studio Code and handling the internationalization, as from my experience it helps a lot.

The FE also displays few phrases that are taken directly from the Redmine. An example of this phrase is a name of an issue status. This should not be problematic, as the BE requests the data in the name of the currently logged user, and the Redmine serves the data in the language configured by that user.

5.9 Utilizing ESLint, auto-import and GitHub Copilot for a more efficient coding

My colleague Dan recommended me to use ESLint and auto-import to improve the overall quality of the project. As both of these features were helpful during the implementation process, I'll mention them here, along with the GitHub Copilot.

5.9.1 ESLint

ESLint statistically analyzes the code to quickly find the problems. As stated in its documentation at [117], it is available in most text editors. The user can

5.9. Utilizing ESLint, auto-import and GitHub Copilot for a more efficient coding

define a predetermined set of rules to adhere to in their project, and ESLint will notify them if they deviate from these rules. My set of rules for the frontend project is visible in the figure 5.6, and I've also got something similar for the backend. The core of this configuration was passed to me directly by Dan and I've decided to use them, while only tweaking couple of setting to be able to fully use the ESLint.

A very good feature of ESLint is auto fix - the ESLint can automatically correct certain errors in your files to conform to the predefined rules. This can be done with the call of `eslint -fix`. An even cooler functionality is the "lint on save" one. When you configure well your IDE or the text editor, every time you save your file, the file gets fixed by the ESLint. This functionality alone saved me couple of hours during the development process and I highly recommend it to everyone in the JS development area. The initial set up of the ESLint might be a bit tricky, but the pay off is immense.

5.9.2 Auto import

Another interesting feature I got recommended was the **unplugin auto import** package. This npm package, available at [118], allows for an auto import of multiple components. This leads to a much cleaner code, as many of the boilerplate import statements can be eliminated with the use of this functionality. For this package to work with the ESLint, a proper configuration is required. The configuration in question is described on the official pages of the package at [118]. I would once again recommend all the developers to have a look at this package, as its advantages are in my opinion well worth it.

5.9.3 GitHub Copilot

During the development of my application, I've also used the GitHub Copilot - an AI tool. The copilot can be used directly in the IDE, with the help of the GitHub Copilot plugin. After downloading this plugin, a login to a GitHub PRO account is required. I've got this account thanks to my student status.

The two main features of the Copilot are the code completion and chat. I haven't leveraged the code completion much, but I've tested the chat feature quite extensively. A very good perk of the GitHub Copilot is the fact that it can directly access your files, and so you can discuss the contents of the file with him. From my experience, the Copilot worked quite well when I discussed the configuration files with him. Especially when setting up the ESLint, I've encountered several problems. Although he didn't directly solve these problems, he came up with several possibilities that would take me half an hour to source out. I'd therefore highly recommend it as a discussion partner for the development and configuration process.

```
7 module.exports = {
8   root: true,
9   env: {
10    node: true,
11  },
12  parser: "@typescript-eslint/parser",
13  extends: [
14    "plugin:vue/vue3-recommended",
15    "eslint:recommended",
16    "./.eslintrc-auto-import.json",
17    "plugin:@typescript-eslint/recommended",
18  ],
19  rules: {
20    "vue/script-indent": [
21      "error",
22      4,
23      {
24        baseIndent: 0,
25      },
26    ],
27    "comma-style": [
28      "warn",
29      "last",
30    ],
31    semi: [
32      "error",
33      "always",
34    ],
35    "no-trailing-spaces": "error",
36    "vue/multi-word-component-names": [
37      "off",
38    ],
39    quotes: [
40      "warn",
41      "double",
42    ],
43    "@typescript-eslint/explicit-function-return-type": "off",
44  },
45  overrides: [
46    {
47      files: ["**/*.vue"],
48      parser: "vue-eslint-parser",
49      parserOptions: {
50        parser: "@typescript-eslint/parser",
51      },
52    },
53  ],
54 };
```

Figure 5.6: ESLint configuration for the FE

Deployment of the prototype

I've worked on the deployment of the prototype with my colleague Max. The prototype was hosted at the server of the company I worked for at the time of writing this thesis. In this chapter, I'll talk about the containerization - a concept I've decided to leverage for the deployment.

6.1 Concept of containerization

As per AWS, the containerization “is a software deployment process that bundles an application’s code with all the files and libraries it needs to run on any infrastructure. Traditionally, to run any application on your computer, you had to install the version that matched your machine’s operating system. For example, you needed to install the Windows version of a software package on a Windows machine. However, with containerization, you can create a single software package, or container, that runs on all types of devices and operating systems.” [119]

The definition of the containerization in the paragraph above describes mainly the portability benefit of the containerization - the ability to deploy applications in multiple environments without rewriting the program code. This portability can be leveraged not only for the production releases, but also for local development. I've worked on several projects that leveraged the containerization. For me as a programmer, this meant I didn't have to download specific versions of the programming languages and libraries just to be able to start the project locally. All I had to do was to download a single software - the containerization software. This software handled all the language and library versions for me.

6.2 Leveraged containerization technology

The containerization software I had experience with was the Docker. Due to this fact, I've decide to leverage Docker for this project as well. With the help of online resources, GitHub Copilot and trial-and-errors, I've managed to prepare a Dockerfile for both the FE and BE. For the BE, I've also prepared a docker-compose - a file that groups several containers and allows them to communicate with one another.

I've used the docker-compose file to group the backend with the database container. Near the end of the implementation process, I've decided to exchange the SQLite database technology for PostgreSQL. The docker-compose file I've created allows the user to initialize both the database and the backend with few command pasted to his terminal. It is another case where the portability of the containerization is perfectly leveraged.

6.3 A remark on the database change

According to the PostgreSQL official website at [120], the PostgreSQL is a powerful open source object-relational database system with over thirty five years of active development. I've used it for several projects in the past and I also have experience with its containerization. It was therefore a really good fit for my use case. The change from SQLite to PostgreSQL was not that complicated in the backend part, as both the SQLite and PostgreSQL use the SQL dialect and have similar sets of data types. As for the libraries used, I've gone with npm package **pg** as the database client and npm package **node-pg-migrate** for the migration management. Using a migration management library standardizes the database schema changes and improves the development process.

6.4 Cooperation with my colleague

My colleague Max refactored the Dockerfile for FE prepared by me, as he was able to optimize it better for the deployment. He has also managed to prepare auto deploy scripts, which allow for automatized deployment of the backend and frontend. The first functional version of the prototype was online on May 3, 2024.

User testing

I've performed a user testing of the prototype. In this chapter, I'll go over the testing agenda, talk about the participant recruitment and present the insights gathered.

7.1 The agenda of the testing

Creating a well thought agenda was the first step of the user testing process. The final agenda consisted of four parts, namely:

1. **A general introduction** – Thank the person for participating. Talk a few sentences about the project and the user testing that is taking place.
2. **A screener** – Quick questionnaire focused on determining whether the user fits the application user base. A user didn't fit the study in case his he had no prior experience with project management tools and also had no prior experience with a helpdesk. In case of such participant, the testing would end here.
3. **A task introduction** – Talk about the testing taking place. I've also told them that I'm in no way evaluating their performance, but I'm rather trying to gather information on the application in question. I also made a request to them to think-aloud if possible, as that would greatly benefit me.
4. **The testing** – Consisting of four scenarios. A scenario represents a realistic situation, where a person performs a list of tasks using the product being tested. I've decided to have a "Registration" scenario, a "Report a bug" scenario, a "Is the task solved" scenario and a "See the developer's reaction" scenario. These should encompass the most common actions performed in the system. A more detailed description of these tasks is available in the attachment **utAgenda.pdf**.
5. **A post-test questionnaire** – Conducted as the last part of the user testing. It was aimed to gather even more feedback on the prototype from the participants.

Some of the information presented in the list above was sourced from a presentation titled “User Interface Testing” [121], which is a part of “User Interface Design” course. I’ve taken this course during my master studies.

7.2 Participants profile

As for the recruitment process, I’ve recruited some of my friends and also a customer of the company I’ve been working for at the time of writing this thesis. There were three participants in total. All of the participants had a technical background, with two being IT students and the third one working in a warehouse. All three of the participants had some previous experience with the use of help desks, but only one had ever used Redmine before. My exclusion criteria was “User has never used a help desk before.” and since all of the participants had some experience, they were able to participate.

7.3 The testing process

The testing took place in-person twice and once over the internet, via Google Meet software, with the help of screen sharing functionality. The in-person testing was more valuable from my point of view, as I was able to observe more aspects than in the online case. I’ve used the mentioned User testing agenda to moderate the testing process. In the following section, I will discuss the observations made during the testing.

7.4 Observations

I’ve noted down multiple observations. Some were explicitly voiced out by the participants of the study, some were observed by me and further confirmed with the participant and some were gathered during the post-test questionnaire. The full list is as follows:

1. **Using the go back functionality of the browser to a return to a page containing a data table should keep the previously applied filters applied.** The filters should stay applied, as this would lead to a better user experience.
2. **After uploading an image describing the issue, the user might not know what to write to the description field.** An indication of the fact that description is not required in this case could smooth out the user experience.
3. **The status filter values and status column values in the issue table don’t match.** The filter contains only values New, Open and Closed, but the table’s column can contain all the possible Redmine statuses such as To clarify, In development, ... A unification should be done, either by providing all the possible statuses as a filter, or by simplifying the status values presented in the data table.
4. **The enter button click should trigger the login button on the login screen.** This issue was observed multiple times.

5. **The comment functionality buttons are not clear in their functionality.** The comment functionality contained three buttons in case the issue was assigned to the client. The not clear buttons were **Add a comment** and **Comment and reassign**.
6. **The idea whether the list of issues on homepage and the table of issues from the navigation drawer represent the same entities is not clear.**
7. **The upload file modal is designed poorly.**
8. **Missing the auto-upload functionality.** The idea that the file has to be explicitly uploaded is not communicated well enough. An auto upload or a notification for case when the file is inserted but not uploaded could solve this issue.
9. **The subject field on the Report a bug screen could be styled to look more important.**
10. **The link on the Issue detail screen leading to Redmine is poorly designed. The redirect is not intuitive and the clickable area is way too big.** A solution could be switching to a button titled “Open in Redmine”.
11. **The attributes on the issue detail page are not well positioned.** The status is the most important attribute for the customer, so it should be presented in such way. The customer doesn’t really care about the created on attribute.
12. **The modal indicating no updates happened doesn’t match the rest of the design, due to it’s background color.**
13. **The list on the homepage could introduce a drop down button that would present a more detailed look on the issue.**
14. **The information that an attachment belongs to a comment is not communicated on the issue detail screen.**
15. **The mark as resolved functionality on the issue detail page is not clear.** One of the participants didn’t exactly know what it was meant to represent.
16. **Creating an issue and a comment shouldn’t redirect to a list of issue, but rather to a detail issue of the current page.**
17. **The FRESH tasks idea is not communicated well.** One of the participants didn’t understand what do the fresh tasks means, what is the time span. A hover information stating that those are the tasks updated in the last two weeks could solve this issue.
18. **The “Is waiting” status is not clear enough.** One of the participants mentioned that from his point of view, the status “Is waiting” is not clear enough, as it doesn’t explicitly state who should perform the next action. The customer may be able to determine who should perform the next

action by looking at the assignee field of the issue. If the assignee is the customer, then he should provide more info, but in such case, the “Is waiting” status doesn’t seem that optimal and the software company side should choose something else, such as “To specify”. This is overall an interesting observation, as per my knowledge, the mentioned status is a custom Redmine status. It promotes two ideas:

- Showing all the custom Redmine statuses to the customer might not be optimal. This matches the 3rd item of this list, where the Redmine and helpdesk statuses didn’t match. Creating a set of statuses visible on helpdesk and mapping all the existing Redmine statuses there might be an option to solve this.
 - The information about the fact that an action is required from the customer is not communicated well enough in the prototype.
19. **The watchers field is missing on the issue detail screen.** The watchers of the issue refer to the users who have activated notifications to receive updates about any changes made to the issue. One participant told me that for him, it is essential to know whether his boss knows about the changes made to the issue, which is communicated by the watchers field in Redmine. In my design of helpdesk, I’ve left this feature out.
20. **Missing tracker field on report a bug screen.** As per my design, I’ve removed the tracker field from the report a bug form, as I’ve been of an impression that the project manager can better decide this field. In a discussion with one of the participants, he revealed that often, he doesn’t exactly know, whether he encountered a bug or he simply can’t navigate the software in question. He also doesn’t want to offend the software company by explicitly stating that it is a bug. I see two possible solutions to this problem:
- Define three categories, such as **Bug** - an error with software, **Request** - a request for a new functionality, **Support** - a request to help navigating the software. These categories would be linked with appropriate Redmine tracker types.
 - Rename the bugs and issues to tickets in the helpdesk. “Creating a ticket” instead of “Reporting a bug” might have a better psychological impact on the customer, as ticket sounds more neutral than a bug.

I think that the second option is better, as it would keep the project manager’s hands free and overall bring less complexity to the system.

21. **Missing the options to set the priority on the report a bug screen.** I’ve also removed the option to set a priority of the created issue from the report a bug form. The reasoning behind this removal was similar to the tracker field - the customer often wants it as soon as possible, so he goes for a high priority. After some discussion with a participant, I’ve come to a conclusion that the priority field should be kept, but maybe in a more simpler way. The user simply needs to be able to express whether the issue is urgent or not. A simple checkbox on the

frontend side could solve this. On the backend part, either mapping this to a Redmine priority option - **Normal** for non-urgent tasks and **Urgent** for urgent tasks, or adding a flag attribute called “Urgent in customer’s eyes” to the Redmine issue.

22. **Information about the estimated time of fix would be valuable for the customer.** After reporting an issue, it would be useful for the customer to see the information about an estimated time of fix. This would especially help in cases when the reported issue is critical and impacts the customer’s ability to perform the work. The solution would be introducing a new attribute to the issue, where such estimate would be held.
23. **A better communication of the issues for which an action from the user’s side is required would be helpful.** While accessing the list of issues, a participant told me that sometimes, he doesn’t really know whether an action is required on his part, or the issue is already closed. From my observation, this ties to two things:
 - The Redmine state “Solved” being unclear to the customer. Although it sounds like a closed state, it doesn’t have such attribute in the Redmine instance I’ve been using. From customer’s perspective, the solved might mean that the issue is already closed. From the perspective of the software company, it might mean that the customer is required to confirm the issue being solved by changing the state to the “Closed” value. This relates to the idea about a set of statuses specific for helpdesk presented above.
 - The actions required from user not being communicated well enough. Once again, this has already been mentioned above, yet I mention it here, as it is in my opinion quite important. A solution could be introducing a list of issues, something like “Awaiting your reaction” to the homepage.

7.5 Fixing the observed issues

I’ve chosen several issues mentioned in the list above and fixed them. More precisely, the changes I’ve made are:

- **Persisting the previously applied filters, sort and pagination in the issue table.** In my eyes, this was a significant problem, I’ve therefore prioritized it. The fix was done with the help of query parameters. The configuration of the data table is persisted as the query parameters in the URL. When user visits another page and afterward returns to the issue table, he returns to an URL containing query parameters. These parameters are processed by the Vue logic and applied correctly to the table. Due to the complexity of this component, I’ve also decided to use the debounce functionality. This has allowed me to have a better control over the requests sent to the backend. I’ve leveraged the **lodash debounce** npm package and I’ve sourced the post “How to Debounce Input in Vue3”, available at [122].

- **Allow the submission of login and register form with the Enter button click.** I've prioritized this problem as well, as it was observed during two user testing sessions. The fix was quite straightforward, I've simply used the `@submit` functionality of the Vuetify's form component and mapped it to the correct button.
- **Move the "See in Redmine" functionality from the page title to a button on the Issue detail screen.** One of the participants had problem with the clickable area around the title on the Issue detail screen. I've decided to fully remove the title. The "See in Redmine" functionality is however retained, but I've moved it to a button titled "See in Redmine", which is in my opinion a more user friendly approach. I've also moved the issue Id to the issue detail's card.
- **Fix redirection on new bug and new comment.** In the previous version, adding a new comment to an issue and also reporting a new bug both redirected to the issues list. In the new version:
 - When adding a comment to an issue the user already is on the issue detail screen. I've removed the redirection and added the BE call to reload the data source, when the user adds a new comment. This leaves the user on the detail of the current issue, but renders the newly added comment.
 - After reporting a bug, the user is redirected to the issue detail page of the newly reported bug.
- **Fix the issue attributes design on the Issue detail screen.** The attributes of issue were not prioritized well on the mentioned screen. I've decided to restructure them and I've also changed the `created` attribute to show time in a more user friendly way.

Overall, I've managed to fix four out of five functional problems observed during the user testing. The only observed functional problem left to be fixed is the upload file modal one. An auto-upload would probably fit best, but I'll have to analyse it and decide on the most optimal approach. Most of the observed non functional problems require some type of the design decision and that's why I haven't fixed them directly. I plan to address all observed problems in the weeks following the submission of the thesis.

Future of the project

There are several options for the future of the project. I've touched on some in the previous chapters, and I'll try to summarize them here.

8.1 Fixing the remaining known issues

During the user testing, I've observed multiple issues with the system. Some of them, I've already solved, some are yet to be solved. In addition to these observed issues, I've also encountered several UI and functional issues during the development process. Those issues were:

- **Redmine text not always rendering correctly in the help desk.**
This is a data related issue, meaning text gets often rendered correctly, but for some cases, it doesn't. This might be due to specific tags used by Redmine, or due to the incompetence of the libraries used for the transformation.
- **TinyMCE editor not working well with images inside the text.**
As I've mentioned, the TinyMCE editor and the way I'm required to handle the file uploads don't really match. I've tried to disable the option of pasting an image to the editor, but my solution didn't function in all the cases I've tested. I've therefore decided to undo the disablement. A better approach to this problem – one that will cooperate well with the file uploads and also allow the pasting of images to the editor – should be looked into and implemented.

These issues should be in my opinion the number one priority, as they are already known and were marked problematic by the user's during the testing process.

8.2 Refactoring the Helpdesk Configuration

At the end of the **Design** chapter, I've touched on the idea of Helpdesk Configuration, an approach where a set of configurable variables should be introduced. These variables should personalize the functionality of the software. The approach presented in the mentioned chapter describes the Redmine plugin plan, which should in my opinion be a second step taken in the future of this project.

Once this Helpdesk Configuration plugin is implemented and properly tested, the help desk will be able to function with multiple instances of Redmine, considering these instances will use this plugin with a proper configuration. At this point, a possibility to market the solution and offer it as a service to potential customers is open.

8.3 Shift in the email strategy

Changing the notification email sent to the customers is another area that can be looked into. In the current state, a customer is informed about the change by a standard Redmine email, which contains links to Redmine. The idea would be to exchange these default templates for a custom ones with a more user friendly content and links leading directly to help desk. An email indicating that a bug report was done successfully would also be useful. From the technical standpoint, the mailer used by Redmine should be leveraged. This approach points towards another plugin for Redmine. There is a possibility to group it with the Helpdesk Configuration plugin, but I'd say a new plugin is a cleaner approach.

8.4 Implementing new functionalities

I've managed to implement almost all of the requirements in the prototype, but there are few things missing:

- **The ability to mark comment as a team one.** - These comments should be visible only by the developers. Should be implemented with the help of the HC plugin.
- **Full text search in issues.** - The Redmine developer guide informs that the search is supported and can be performed with the use of query parameters. Vuetify also supports a search input on its data table component. These two functionalities could therefore be leveraged together to support the full text search.
- **Display the estimated completion time to customer.** - Currently, the help desk does not display the information on the estimated time of the issue completion. This attribute should be firstly added via the HC plugin to Redmine, and afterward displayed on proper places.
- **The ability to work with both bug and feature request in help desk.** - Although this is supported, the current version of the help desk's UI is more suited towards the bugs. A partial redesign might be suitable here, as this requirement also matches the observation number 20 from the user testings, where the participants mentioned that in some cases he isn't able to decide on the category of his request.

Besides the functionalities mentioned in the user requirements, there is also an option of introducing completely new ones. Looking back at the the analysis of existing help desk solutions, some interesting improvements could be:

1. **SLA monitoring** - helping the project manager handle the project better.

2. **Knowledgebase module** - providing support to the customers without the need of human interaction. Either in a way of most frequently asked questions list, or some type of a chat bot.
3. **Introducing machine learning** - providing the information from the already solved issues as well as open issues to the AI models would be helpful in more than one way. Firstly, this model could be leveraged to generate more precise answers for the chat bot. Secondly, such model could also be leveraged by the project managers and developers when handling the issues.
4. **Allowing to report a bug without being logged in** - some of the analysed solutions, such as Jira Helpdesk, offer this functionality. Some existing processes would have to be analysed and improved upon to allow for such functionality, but with the help of HC plugin, it would be possible.

Conclusion

The goal of this thesis was to design and implement a web application that will serve as a helpdesk portal for Redmine, a project management tool. The primary goal of this web application was a simplified user interface.

In the introduction, I've presented the idea that changes are a constant feature of a software development, and software called project management tools are a way of managing these changes.

The introduction was followed by the state-of-the-art chapter, where in the beginning I presented the Redmine software and the notion of a help desk. In this chapter, I've also performed an analysis of a few existing helpdesk solutions - three solutions specific to Redmine, two solutions described in a scholar literature and three out-of-the-box solutions.

The second chapter, titled "User requirements gathering", presented the background of the topic of my thesis, followed by the presentation of sixteen user requirements gathered from the discussions with my supervisor, analysis of existing solutions and a consultation with a colleague.

The third chapter presented the technological stack suitable for the web application. It starts with a brief introduction to web development and client-side JavaScript frameworks. Afterward, three popular JavaScript frameworks are presented and compared. In the end, the Vue framework comes out as the winner. The rest of the third chapter focuses on the Vue framework technology and its features. In the end, I conclude the chosen technological stack.

The "Technological stack" chapter is followed by the "Design" chapter. In this chapter, I introduced the way of communication between helpdesk and Redmine. I also presented the architecture of the application. The chapter further presented the choice of the technology for the backend. This section was followed by the UI design section, where I talked about the ideas behind the layout of the application, as well as two screens. I finished the "Design" chapter with the talk about the configurable fields and Helpdesk Configuration, a utility that would allow for the distribution of the helpdesk to multiple instances of Redmine.

In the fifth chapter, I talked about interesting areas and concepts I've encountered during the implementation. These areas included the initialization of the projects, authentication, handling the text encoding, working with files and images, routing in the frontend, internationalization, and leveraged developer productivity tools.

The deployment of the prototype was described in the sixth chapter. Containerization was leveraged in this process, and my colleague Max assisted me with the deployment. Together, we managed to get the prototype online, so that the user testing can be performed on it.

The deployment chapter was followed by the “User testing” chapter, where I talked about the agenda of testing and introduced the profile of the participants. I finished this chapter with the list of observations done during the user testing and the improvements made to fix the observed issues.

In the last chapter of the thesis, I talked about the future of the project, touching on the topics like fixing the remaining known issues, shift in the email strategy and adding new functionalities.

Overall, I’ve managed to fulfill all the goals of the thesis, by analysing, designing, implementing, deploying, and user-testing a prototype of a help desk for Redmine. This helpdesk supports internationalization and promotes a simple user interface suited and optimized for users without technical expertise.

Bibliography

- [1] Johnson, J.; Dubois, P. Issue tracking. *Computing in Science & Engineering*, volume 5, no. 6, 2003: pp. 71–77, doi:10.1109/MCISE.2003.1238707.
- [2] Janák, J. *Issue tracking systems*. Dissertation thesis, Masarykova univerzita, Fakulta informatiky, 2009.
- [3] Eurostat. How many citizens had basic digital skills in 2021? c1995-2024. Available from: <https://ec.europa.eu/eurostat/en/web/products-eurostat-news/-/ddn-20220330-1>
- [4] Wang, D.; Li, T.; et al. iHelp: An Intelligent Online Helpdesk System. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, volume 41, no. 1, 2011: pp. 173–182, doi:10.1109/TSMCB.2010.2049352.
- [5] Redmine Team. Redmine. c2006-2023. Available from: <https://www.redmine.org/>
- [6] Wikipedia contributors. Redmine — Wikipedia, The Free Encyclopedia. <https://en.wikipedia.org/w/index.php?title=Redmine&oldid=1215858257>, 2024, [Online; accessed 19-April-2024].
- [7] PYPL team. PYPL PopularitY of Programming Language. c2023. Available from: <https://pypl.github.io/PYPL.html>
- [8] Redmine boards contributors. Can you please explain me about redmine application ? c2006-2023. Available from: <https://www.redmine.org/boards/1/topics/27559>
- [9] Wikipedia contributors. TYPO3 — Wikipedia, The Free Encyclopedia. <https://en.wikipedia.org/w/index.php?title=TYPO3&oldid=1212034273>, 2024, [Online; accessed 14-April-2024].
- [10] Warhorse Studios s.r.o. Kingdom Come: Deliverance 2. c2024. Available from: <https://www.kingdomcomerpg.com/>
- [11] Warhorse Studios s.r.o. About us. c2019. Available from: <https://warhorsestudios.cz/about/>

BIBLIOGRAPHY

- [12] Redmine team. Who uses Redmine? c2006-2023. Available from: <https://www.redmine.org/projects/redmine/wiki/WeAreUsingRedmine>
- [13] Gantt.com team. What is a Gantt Chart? c2024. Available from: <https://www.gantt.com/>
- [14] Redmine team. Developer guide. c2006-2023. Available from: https://www.redmine.org/projects/redmine/wiki/Developer_Guide
- [15] RedmineUP team. Redmine Tutorial: How to Get Started and Make Profit. c2010-2024. Available from: <https://www.redmineup.com/pages/blog/how-to-install-redmine-and-launch-projects?ssp=1&darkschemeovr=1&setlang=en&cc=CZ&safesearch=moderate>
- [16] RedmineUP team. Create Issues. c2010-2024. Available from: <https://www.redmineup.com/pages/help/redmine/create-issues?ssp=1&darkschemeovr=1&setlang=en&cc=CZ&safesearch=moderate>
- [17] Redmine Team. Custom fields. c2006-2023. Available from: <https://www.redmine.org/projects/redmine/wiki/redminecustomfields>
- [18] Redmine Team. Issue tracking system. c2006-2023. Available from: <https://www.redmine.org/projects/redmine/wiki/redmineissuetrackingsetup>
- [19] Cambridge University Press. Helpdesk. c2024. Available from: <https://dictionary.cambridge.org/dictionary/english/helpdesk>
- [20] Cambridge University Press. Help desk. c2024. Available from: <https://dictionary.cambridge.org/dictionary/english/help-desk>
- [21] Wikipedia contributors. Help desk — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Help_desk&oldid=1204970376, 2024, [Online; accessed 15-April-2024].
- [22] RedmineUP team. Redmine Helpdesk plugin. c2010-2024. Available from: <https://www.redmineup.com/pages/plugins/helpdesk>
- [23] qutic development GmbH. Redmine Helpdesk. https://github.com/jfqd/redmine_helpdesk, c2012-2022, [Online; accessed 15-April-2024].
- [24] Redmine Team. Receiving emails. c2006-2023. Available from: <https://www.redmine.org/projects/redmine/wiki/RedmineReceivingEmails>
- [25] Wikipedia contributors. Easy Redmine — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Easy_Redmine&oldid=1193557030, 2024, [Online; accessed 15-April-2024].
- [26] Wikipedia contributors. Agile software development — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Agile_software_development&oldid=1216852586, 2024, [Online; accessed 15-April-2024].
- [27] Easy Software Ltd. Redmine Help Desk plugin. c2005-2024. Available from: <https://www.easyredmine.com/redmine-helpdesk>

-
- [28] Duffy, J. Redmine - Review 2021. c1996-2024. Available from: <https://uk.pcmag.com/project-management/120844/redmine>
- [29] PC Mag Editors. Help desk ticket. c1996-2024. Available from: <https://www.pcmag.com/encyclopedia/term/help-desk-ticket>
- [30] Herrick, D. R.; Metz, L.; et al. Effective zero-cost help desk software. In *Proceedings of the 40th annual ACM SIGUCCS conference on User services*, 2012, pp. 157–160.
- [31] Sinnett, C. J.; Barr, T. OSU helpdesk: a cost-effective helpdesk solution for everyone. In *Proceedings of the 32nd Annual ACM SIGUCCS Conference on User Services*, SIGUCCS '04, New York, NY, USA: Association for Computing Machinery, 2004, ISBN 1581138695, p. 209–216, doi: 10.1145/1027802.1027851. Available from: <https://doi.org/10.1145/1027802.1027851>
- [32] Haan, K. Best Help Desk Software (2024). c2024. Available from: <https://www.forbes.com/advisor/business/software/best-help-desk-software/>
- [33] Zoho Corporation. Features | Zoho Desk. c2024. Available from: <https://www.zoho.com/desk/features.html>
- [34] Lee, P.; Goldberg, C.; et al. *The AI revolution in medicine: GPT-4 and beyond*. Pearson, 2023.
- [35] Chan, C. K. Y.; Tsi, L. H. The AI Revolution in Education: Will AI Replace or Assist Teachers in Higher Education? *arXiv preprint arXiv:2305.01185*, 2023.
- [36] Cooper, R. G. The Artificial Intelligence Revolution in new-product development. *IEEE Engineering Management Review*, 2023.
- [37] Bhat, S. A.; Huang, N.-F. Big data and ai revolution in precision agriculture: Survey and challenges. *Ieee Access*, volume 9, 2021: pp. 110209–110222.
- [38] Wikipedia contributors. Jira (software) — Wikipedia, The Free Encyclopedia. [https://en.wikipedia.org/w/index.php?title=Jira_\(software\)&oldid=1215859843](https://en.wikipedia.org/w/index.php?title=Jira_(software)&oldid=1215859843), 2024, [Online; accessed 17-April-2024].
- [39] Atlassian. ITSM software features | Request Management. c2024. Available from: <https://www.atlassian.com/software/jira/service-management/features/itsm>
- [40] Atlassian. Jira Service Management pricing. c2024. Available from: <https://www.atlassian.com/software/jira/service-management/pricing>
- [41] Atlassian. How Jira Service Management and Jira work together. c2024. Available from: <https://www.atlassian.com/software/jira/service-management/product-guide/tips-and-tricks/jira-service-management-and-jira-software>

BIBLIOGRAPHY

- [42] Spiceworks Inc. What is Spiceworks Cloud Help Desk? c2006-2024. Available from: <https://www.spiceworks.com/free-cloud-help-desk-software>
- [43] Wikipedia contributors. Vendor lock-in — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Vendor_lock-in&oldid=1212388775, 2024, [Online; accessed 17-April-2024].
- [44] Wikipedia contributors. Functional requirement — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Functional_requirement&oldid=1212851555, 2024, [Online; accessed 17-April-2024].
- [45] Raymond, E. S. Basics of the Unix Philosophy. Available from: <http://www.catb.org/~esr/writings/taoup/html/ch01s06.html>
- [46] Glinz, M. On Non-Functional Requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, 2007, pp. 21–26, doi: 10.1109/RE.2007.45.
- [47] Redmine Team. Plugin Tutorial. c2006-2023. Available from: https://www.redmine.org/projects/redmine/wiki/Plugin_Tutorial
- [48] MDN Web Docs Contributors. Web standards. c1998-2024. Available from: <https://developer.mozilla.org/en-US/curriculum/core/web-standards/>
- [49] MDN Web Docs Contributors. Document Object Model (DOM). c1998-2024. Available from: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
- [50] MDN Web Docs Contributors. Web standards. c1998-2024. Available from: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/How_the_Web_works
- [51] Wikipedia contributors. Web development — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Web_development&oldid=1216918289, 2024, [Online; accessed 17-April-2024].
- [52] MDN Web Docs Contributors. Understanding client-side JavaScript frameworks. c1998-2024. Available from: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks
- [53] Wikipedia contributors. URL — Wikipedia, The Free Encyclopedia. <https://en.wikipedia.org/w/index.php?title=URL&oldid=1210046855>, 2024, [Online; accessed 17-April-2024].
- [54] Wikipedia contributors. React (JavaScript library) — Wikipedia, The Free Encyclopedia. [https://en.wikipedia.org/w/index.php?title=React_\(JavaScript_library\)&oldid=1219382752](https://en.wikipedia.org/w/index.php?title=React_(JavaScript_library)&oldid=1219382752), 2024, [Online; accessed 17-April-2024].

-
- [55] Patel, S. 13 Most Popular Websites Built With React in 2023-2024. c2024. Available from: <https://www.cmarix.com/blog/most-popular-websites-built-with-react/>
- [56] Wikipedia contributors. Vue.js — Wikipedia, The Free Encyclopedia. <https://en.wikipedia.org/w/index.php?title=Vue.js&oldid=1216191150>, 2024, [Online; accessed 18-April-2024].
- [57] Wikipedia contributors. AngularJS — Wikipedia, The Free Encyclopedia. <https://en.wikipedia.org/w/index.php?title=AngularJS&oldid=1217368237>, 2024, [Online; accessed 18-April-2024].
- [58] Kugell, A. 15 Global Websites Using Vue.js in 2024. c2024. Available from: <https://trio.dev/websites-using-vue/>
- [59] Wikipedia contributors. Svelte — Wikipedia, The Free Encyclopedia. <https://en.wikipedia.org/w/index.php?title=Svelte&oldid=1210470585>, 2024, [Online; accessed 18-April-2024].
- [60] Svelte contributors. Svelte components. c2023. Available from: <https://svelte.dev/docs/svelte-components>
- [61] Svelte contributors. Introduction SvelteKit. c2023. Available from: <https://kit.svelte.dev/docs/introduction>
- [62] Ahinon, J. Top 10 Big Companies Using Svelte. Okupter. Available from: <https://www.okupter.com/blog/companies-using-svelte>
- [63] kiraaziz. Choosing the Right Frontend Framework: React vs. Vue vs. Svelte. c2016-2024. Available from: <https://dev.to/kiraaziz/choosing-the-right-frontend-framework-react-vs-vue-vs-svelte-2n48>
- [64] facebook. react. <https://github.com/facebook/react>, c2024.
- [65] contributors, E. Y. . V. vuejs/core. <https://github.com/vuejs/core>, c2013-2024.
- [66] contributors, S. svelte. <https://github.com/sveltejs/svelte>, c2024.
- [67] LinkedIn team. How do you reduce bundle size and improve user experience? c2024. Available from: <https://www.linkedin.com/advice/3/how-do-you-reduce-bundle-size-improve-user>
- [68] GitHub team. Saving repositories with stars. c2024. Available from: <https://docs.github.com/en/get-started/exploring-projects-on-github/saving-repositories-with-stars>
- [69] Vue.js team. Release v3.0.0 One Piece. c2013-2024. Available from: <https://github.com/vuejs/core/releases/tag/v3.0.0>
- [70] semver contributors. Semantic Versioning 2.0.0. Available from: <https://semver.org/>
- [71] Vue.js team. Composition API FAQ. c2014-2024. Available from: <https://vuejs.org/guide/extras/composition-api-faq.html>

BIBLIOGRAPHY

- [72] Vue.js team. Composables. c2014-2024. Available from: <https://vuejs.org/guide/reusability/composables>
- [73] Vue.js team. SFC Syntax Specification. c2014-2024. Available from: <https://vuejs.org/api/sfc-spec>
- [74] Vue.js team. <script setup>. c2014-2024. Available from: <https://vuejs.org/api/sfc-script-setup.html>
- [75] Bierman, G.; Abadi, M.; et al. Understanding typescript. In *ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Sweden, July 28–August 1, 2014. Proceedings 28*, Springer, 2014, pp. 257–281.
- [76] Vue.js team. Using Vue with TypeScript. c2014-2024. Available from: <https://vuejs.org/guide/typescript/overview>
- [77] Bonisteel, S. 10 Vue Component Libraries You’ll Want to Know. c2013-2024. Available from: <https://kinsta.com/blog/vue-component-libraries/>
- [78] Vuetify team. The Vuetify roadmap. c2016-2024. Available from: <https://vuetifyjs.com/en/introduction/roadmap>
- [79] W3Schools team. Node.js NPM. c1999-2024. Available from: https://www.w3schools.com/nodejs/nodejs_npm.asp
- [80] NodeSource team. Choosing the Right Node.js Package Manager in 2024: A Comparative Guide. c2024. Available from: <https://nodesource.com/blog/nodejs-package-manager-comparative-guide-2024/>
- [81] Redmine Team. Redmine API. c2006-2023. Available from: https://www.redmine.org/projects/redmine/wiki/rest_api
- [82] CORINA. Best practices for working with API keys in the frontend. c2024. Available from: <https://jsramblings.com/best-practices-for-working-with-api-keys-in-the-frontend/>
- [83] J Zeil, S. Compiling Programs. c2023. Available from: <https://www.cs.odu.edu/~zeil/cs252/latest/Public/compilation/index.html>
- [84] Amazon Web Services team. What’s the Difference Between Frontend and Backend in Application Development? c2024. Available from: <https://aws.amazon.com/compare/the-difference-between-frontend-and-backend/>
- [85] Parecki, A. OAuth 2.0 Simplified. Available from: <https://www.oauth.com/>
- [86] Redmine issues contributors. Feature #24808: OAuth2 support for Redmine API Apps (OAuth2 Provider). c2006-2023. Available from: <https://www.redmine.org/issues/24808>

-
- [87] expressjs.com contributors. Express. c2017. Available from: <https://expressjs.com/>
 - [88] Amazon Web Services team. What is SQL (Structured Query Language)? c2024. Available from: <https://aws.amazon.com/what-is/sql/>
 - [89] SQLite team. SQLite Home Page. 2024. Available from: <https://www.sqlite.org/>
 - [90] JAGU team. Uživatelská dokumentace. c2024. Available from: <https://manual.sklady.jagu.cz/user-documentation/guide/>
 - [91] Vuetify team. Get started with Vuetify 3. c2016-2024. Available from: <https://vuetifyjs.com/en/getting-started/installation/>
 - [92] Amazon Web Services team. What is an IDE (Integrated Development Environment)? c2024. Available from: <https://aws.amazon.com/what-is/ide/>
 - [93] Crockford, Douglas. Introducing JSON. Available from: <https://www.json.org/json-en.html>
 - [94] Okta team. JSON Web Tokens. c2024. Available from: <https://auth0.com/docs/secure/tokens/json-web-tokens>
 - [95] MDN Web Docs Contributors. HTTP response status codes. c1998-2024. Available from: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>
 - [96] Pinia team. Introduction. c2019-2024. Available from: <https://pinia.vuejs.org/introduction.html>
 - [97] expressjs.com contributors. Writing middleware for use in Express apps. c2017. Available from: <https://expressjs.com/en/guide/writing-middleware.html>
 - [98] MDN Web Docs Contributors. HTTP headers. c1998-2024. Available from: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>
 - [99] Watmore, Jason. Vue 3 + Pinia - JWT Authentication with Refresh Tokens Example & Tutorial. c2024. Available from: <https://jasonwatmore.com/vue-3-pinia-jwt-authentication-with-refresh-tokens-example-tutorial>
 - [100] Redmine Team. Text formatting. c2006-2023. Available from: <https://www.redmine.org/projects/redmine/wiki/redminetextformatting>
 - [101] Tiny Technologies team. TinyMCE. c2024. Available from: <https://www.tiny.cloud/>
 - [102] Tiny Technologies team. TinyMCE Vue.js integration technical reference. c2024. Available from: <https://www.tiny.cloud/docs/tinymce/latest/vue-ref>

BIBLIOGRAPHY

- [103] GitHub issues contributors. How can I self host this? c2024. Available from: <https://github.com/tinymce/tinymce-vue/issues/20>
- [104] Porsteinsson, B. textile-js: A fully featured Textile parser in JavaScript. <https://github.com/borgar/textile-js>, 2024.
- [105] Axios. Axios Documentation. <https://axios-http.com/docs/intro>, accessed: 2024-04-29.
- [106] MDN Web Docs Contributors. Response: blob() method - Web APIs. <https://developer.mozilla.org/en-US/docs/Web/API/Response/blob>, accessed: 2024-04-29.
- [107] MDN Web Docs Contributors. Blob - Web APIs | MDN. <https://developer.mozilla.org/en-US/docs/Web/API/Blob>, c2024, accessed: 2024-04-29.
- [108] SmartBear Software. API Endpoints. <https://support.smartbear.com/zephyr-scale-cloud/docs/en/rest-api/rest-api--overview.html>, c2024, accessed: 2024-04-29.
- [109] Branch team. Query Parameters. c2024. Available from: <https://www.branch.io/glossary/query-parameters/>
- [110] MDN Web Docs Contributors. What is a URL? c1998-2024. Available from: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_URL
- [111] Mozilla Developer Network. FormData - Web APIs | MDN. <https://developer.mozilla.org/en-US/docs/Web/API/FormData>, c2024, accessed: 2024-04-29.
- [112] Multer Contributors. Multer. Available from: <https://www.npmjs.com/package/multer>
- [113] OpenJS Foundation. Node.js fs Module. <https://nodejs.org/api/fs.html>, accessed: April 29, 2024.
- [114] Vue.js team. Routing. c2014-2024. Available from: <https://vuejs.org/guide/scaling-up/routing>
- [115] Vue Router team. Vue Router | The official Router for Vue.js. <https://router.vuejs.org/>, accessed: 2024-04-29.
- [116] vue-i18n contributors. Vue I18n. c2020. Available from: <https://kazupon.github.io/vue-i18n/>
- [117] OpenJS foundation. ESLint. <https://eslint.org/>, accessed: April 29, 2024.
- [118] Fu, A. unplugin-auto-import: Auto import APIs on-demand for Vite, Webpack, and Rollup. <https://github.com/unplugin/unplugin-auto-import>, 2024.
- [119] Amazon Web Services team. What is containerization? c2024. Available from: <https://aws.amazon.com/what-is/containerization/>

- [120] PostgreSQL Global Development Group. PostgreSQL: The World's Most Advanced Open Source Relational Database. c1996-2024. Available from: <https://www.postgresql.org/>
- [121] Pavlicek, J. User Interface Testing. https://docs.google.com/presentation/d/1t-4kCvHJSqpzqff30JhoAzPvaJQQE1J990geai3K9e8/edit#slide=id.ga8c1693005_0_368, 2023, [Online; accessed 4-May-2024].
- [122] Garrett-Smith, A. How to Debounce Input in Vue 3. c2024. Available from: <https://codecourse.com/articles/debounce-input-in-vue-3>
- [123] Yamaoka, H.; Yamamoto, K.; et al. Case Study of Implementing an IT Service Desk Ticketing System at Small Computer Center. In *Proceedings of the 2019 ACM SIGUCCS Annual Conference*, SIGUCCS '19, New York, NY, USA: Association for Computing Machinery, 2019, ISBN 9781450357746, p. 140–144, doi:10.1145/3347709.3347820. Available from: <https://doi.org/10.1145/3347709.3347820>
- [124] pnpm contributors. Pnpm. c2015-2024. Available from: <https://pnpm.io/>
- [125] MDN Web Docs Contributors. ArrayBuffer - JavaScript. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer, accessed: 2024-04-29.

Acronyms

| | |
|-------------|--|
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| AWS | Amazon Web Services |
| BE | BackEnd |
| CRM | Customer Relationship Management |
| CSS | Cascading Style Sheets |
| DOM | Document Object Model |
| FAQ | Frequently Asked Question |
| FE | FrontEnd |
| HTML | HyperText Markup Language |
| HTTP | HyperText Transfer Protocol |
| ICT | Information and Communication Technology |
| IDE | Integrated Development Environment |
| IT | Information Technology |
| JS | JavaScript |
| JWT | JSON Web Token |
| NBA | National Basketball Association |
| URL | Uniform Resource Locator |
| REST | REpresentational State Transfer |
| SDK | Software Development Kit |
| SFC | Single File Component |
| SLA | Service Level Agreement |

A. ACRONYMS

UI User Interface

WYSIWYG What You See Is What You Get

Contents of attachments

```
readme.txt ..... the file with CD contents description
documents
├── utAgenda.pdf ..... the agenda used during user testing
src
├── app
│   ├── backend ..... the source code of the backend project
│   └── frontend ..... the source code of the frontend project
└── thesis ..... the source code of the thesis in the LATEX format
text
├── thesis.pdf ..... thesis in the PDF format
```