



Bachelor Degree Project

Improving the accessibility of API documentation for non-technical users



Authors: Gabriele Marinosci - Ratmir
Shchadrynski

Supervisor: Aris Alissandrakis

Examiner: Johan Hagelbäck

Semester: VT 2023

Subject: Computer Science

Abstract

Software documentation is one of the most important aspects of the software development process, as it allows the transfer of knowledge between individuals, regardless of their background and their technical knowledge. In particular, Application Programming Interfaces provide the basic structure for the communication and integration between different software systems. Therefore, providing solid API documentation is fundamental for the software development process as it represents one of the main learning tools for developers trying to learn new technologies.

While researchers have studied the fundamentals of good API documentation design for over twenty years, most studies only focused on the point of view of developers. However, with the rising amount of companies offering Software-as-a-Service products, it has also become important to produce documentation that is accessible to people with limited technical knowledge, i.e. customer representatives.

In order to fill this research gap, a case study was conducted in collaboration with a software company. By using a design science approach, this project focused on creating an API playground environment where users could interact with various APIs offered by the company. The results were evaluated through a questionnaire for user feedback and with an interview conducted with developers from the company. While the outcome of the evaluation was positive, the limited scope of the project prevented some important aspects of API documentation such as navigation from being thoroughly examined, therefore more extensive research is needed in the future.

Keywords: REST API, API documentation, User Experience

Preface

First of all, we would like to thank our supervisor Aris, who never failed to answer our questions with great detail and offered valuable guidance for the success of this project. Secondly, we would also like to thank TietoEvry for the opportunity to work with them, everyone at the company has been very helpful and always ready to assist us with any issue. Finally, we would like to thank our families who always supported and motivated us throughout the whole duration of our studies.

Contents

1	Introduction	1
1.1	Background	2
1.2	Related work	2
1.3	Problem formulation	3
1.4	Motivation	3
1.5	Results	4
1.6	Scope/Limitation	4
1.7	Target group	4
1.8	Outline	4
2	Method	6
2.1	Research Project	6
2.2	Small-scale literature study	6
2.3	Design Science Case Study	6
2.3.1	Evaluation	7
2.4	Reliability and Validity	7
2.4.1	Reliability	7
2.4.2	Validity	8
2.5	Ethical consideration	8
3	Theoretical Background	9
3.1	Early research	9
3.2	Approaches to programming	9
3.3	The issues with traditional documentation	10
3.4	Crowd-sourced documentation	10
3.5	Improving the usability API documentation	11
3.5.1	Content	11
3.5.2	Structure	12
4	Research project – Implementation	13
4.1	Back-end	13
4.2	Front-end	13
4.3	The First Iteration	13
4.4	The Second Iteration	16
4.5	The Third iteration	19

5	Results	23
5.1	The First Iteration	23
5.2	The Second Iteration	27
5.3	The Third Iteration	31
5.4	Final interview with developers	36
6	Analysis	40
6.1	Comparing the outcome of first and second iteration	40
6.2	Comparing the outcome of all the iterations	41
6.3	Analysis of interviews with developers	43
7	Discussion	45
8	Conclusion	47
8.1	Future work	47
	References	48
A	Appendix A	A

1 Introduction

Nowadays, Application Programming Interfaces (APIs) play a huge role in providing communication and integration between different software systems [1]. They provide developers with a set of rules and tools to interact with a particular software or platform. However, despite their potential, APIs often present a significant challenge for non-technical users who lack programming expertise or extensive technical knowledge.

Non-technical users face several challenges when interacting with APIs, including understanding API concepts, interpreting technical documentation, and implementing APIs in their applications. The first step in becoming acquainted with APIs involves exploring their documentation. Effective API documentation is crucial for bridging this knowledge gap and empowering non-technical users to leverage the full potential of APIs. It serves as a guide, offering clear instructions, intuitive examples, and comprehensive explanations to facilitate the understanding and utilization of APIs.

Despite extensive research highlighting the significance of documentation, including the provision of examples, consumers frequently express dissatisfaction with its effectiveness. Notably, existing literature primarily focuses on examining challenges related to API documentation solely from the perspective of consumers [2, 3]. In addition to that, the company that collaborates on this study, raised an issue related to potential customers misunderstanding the functionalities of their services. They report that customer representatives often lack technical expertise and high-level descriptions of their services are not always enough to provide a clear image of the functionalities. For this purpose, they suggested the implementation of an interactive API documentation platform, where the functionalities of their APIs can be demonstrated with a higher level of abstraction in order to be accessible to users with lower levels of technical expertise, with the aim of reducing misunderstandings in later stages of production.

Therefore, this thesis focuses on the critical issue of improving the accessibility of API documentation specifically designed for non-technical users. By addressing this challenge, we aim to enhance the usability and adoption of APIs among users from various backgrounds, such as business analysts, project managers, and end-users. The objective is to develop user-friendly API documentation that provides concise and descriptive information, allowing non-technical users to quickly understand APIs. We will explore techniques for approaching documentation and consider factors that affect the level of perception and understanding of documentation by users. Through this research, we seek to bridge the knowledge gap between non-technical users and APIs.

In summary, the objective of this thesis is to enhance the accessibility of API documentation specifically for individuals who lack technical expertise or programming knowl-

edge. By developing user-friendly documentation, we intend to empower individuals from diverse backgrounds to effectively utilize APIs.

1.1 Background

Effective API documentation requires careful consideration of issues such as the content quality and the way of its presentation [4]. To be more precise, the main content-related problem is its incompleteness [4]. It is essential to maintain up-to-date documentation that accurately reflects the current state of the API, including any changes in functionality, new features, or modifications. Overall, API documentation is an essential aspect of software development, and its quality can have a significant impact on the usability, reliability, and security of software systems that use APIs [5, 6]

Considering the non-technical target group, the challenge of inadequate API documentation has significant implications, including reduced adoption, increased errors, and delays in the development process. To overcome these obstacles, there is a need to improve the accessibility and usability of API documentation, making it more user-friendly and fulfill to the needs of non-technical users.

1.2 Related work

There has been a significant amount of research and development in the field of API documentation, with many different approaches and techniques proposed to improve the quality and effectiveness of API documentation. While the issue of the complexity of API documentation seems like a recent problem, already at the end of the 90s and in the early 2000s numerous researchers tried to focus on identifying the biggest obstacles that developers faced while trying to learn a new API, and in doing so, designing high-quality documentation was proven to be the most deciding factor for the learning outcome [7].

Further studies later highlighted the importance of understanding the different approaches that developers show towards learning new technologies, and therefore focused on observing how developers use API documentation, in order to gather the requirements for comprehensive API documentation [5, 8].

Other researchers have focused their efforts on analyzing the current state of API reference documentation and their patterns of knowledge, defined by Maleej et al. as "*the different types of knowledge it contains and how this knowledge is distributed among documents*" [9, p. 2].

1.3 Problem formulation

Plenty of research has already been made in order to tackle the issues that a traditional style of API documentation presents, and how it can be improved to be accessible to different types of developers. However, with the rising number of companies offering Software as a Service (SaaS) products, nowadays there is a newfound need to make the documentation of software products, more particularly REST APIs, accessible not only to developers but also to potential customers, which often presents varying levels of technical expertise. In order to address this gap in research, the following research questions will be used to guide this project:

- **RQ1** - What factors contribute to the quality and effectiveness of API documentation?
- **RQ2** - How can the design of API documentation be optimized to meet the needs of users with varying levels of technical expertise, including both developers and non-technical users?

1.4 Motivation

This problem is important from different perspectives. Foremost, in the context of the software industry, the accessibility of API documentation has practical impacts on developers, businesses, and customers. Inadequate documentation can lead to delays, errors, and increased support costs, negatively impacting product development cycles. By focusing on user-friendly documentation, organizations can enhance their API adoption rates, improve customer satisfaction, and gain a competitive edge in the market. Moreover, non-technical users, such as project managers or business analysts, play critical roles in decision-making processes and require accessible documentation to effectively communicate their requirements and integrate APIs into their workflows.

As introduced in **section 1**, the collaborating company has reported problems with customers misinterpreting the functionalities of their services, which results in problems in the later stages of production and additional costs. Our aim is to use this case study to find basic guidelines that can be potentially generalized and extended to other companies facing similar issues.

By conducting research and developing user-friendly documentation, we can advance the understanding of how to design effective communication channels between software systems and non-technical users. This research contributes to the broader scientific knowledge base and establishes best practices for creating accessible and inclusive software interfaces.

1.5 Results

This project will start with a comprehensive study of the current state of research for API documentation, that will be used to collect some guidelines for good documentation design. These findings will provide a solid theoretical foundation for the success of the project. The second part will focus on the implementation of a possible solution to the identified research gap. Each artifact produced by the iterative development process will be evaluated with a usability survey in order to gain valuable feedback.

1.6 Scope/Limitation

The work was conducted in collaboration with a company that provided access to existing framework infrastructure and servers within the scope of utilizing .NET and Vue.js technologies. However, as a collaborative effort, unexpected technical challenges were encountered, which impeded the workflow. For instance, server access was limited to specific working hours (9 to 18) on weekdays only. Moreover, we faced technical obstacles as the laptops provided by the company came with preinstalled VPN and certificates. This caused initial challenges in accessing the original API since the token certificate was inactive as soon as the laptops were delivered.

Given the limited time that was allocated for this project, it was not possible to add a higher number of APIs to be tested in the playground. For this reason, important aspects of API documentation such as the ease of navigation could not be investigated thoroughly. Moreover, since this is a case study conducted in collaboration with a company that sells Software-as-a-Service (SaaS) products, our project focused only on REST APIs and did not explore other types of APIs.

1.7 Target group

The target group for this thesis on improving the accessibility of API documentation for non-technical users includes individuals who may not have extensive programming expertise or technical knowledge. This group comprises non-technical users who are interested in leveraging APIs for various purposes, such as business analysts, project managers, end-users, or individuals from diverse backgrounds who need to interact with APIs but lack specialized technical skills.

1.8 Outline

- **Introduction:** This section provides an overview of the thesis, including the background, related work, problem formulation, motivation, and initial results. It also

outlines the scope and limitations of the study and identifies the target group for the research.

- **Method:** In this section, the research methodology and approach are described. It covers the research project itself, including its design and implementation, as well as the reliability and validity considerations.
- **Theoretical Background:** This section explores the theoretical foundations relevant to the research topic. It includes early research in the field, different approaches to programming, issues with traditional documentation, the concept of crowd-sourced documentation, and strategies for improving the usability of API documentation.
- **Research Project Implementation:** This section details the implementation of the research project, focusing on the back-end and front-end aspects. It discusses the tools, technologies, and frameworks used and provides an overview of the first and second iterations of the project.
- **Results:** This section presents the findings and results of the research project. It includes the outcomes of both the first and second iterations, highlighting any improvements or challenges encountered during the implementation.
- **Analysis:** Here, the obtained results are analyzed and interpreted. It may involve comparing the findings with the initial objectives, identifying patterns or trends, and drawing meaningful conclusions from the data.
- **Discussion:** This section provides a deeper analysis and discussion of the results, relating them to the existing literature and research in the field. It may address any limitations, implications, or potential future directions based on the findings.
- **Conclusions and Future Work:** The final section summarizes the key findings and conclusions of the thesis. It also highlights areas for further research and suggests potential future work or improvements in the field of API documentation accessibility.

2 Method

The aim of this project is to provide a possible solution to the aforementioned problem and offer guidelines for developers to improve the usability and accessibility of their API documentation. After careful analysis of previous research on the usability of API documentation, given the iterative nature of the development process chosen for this project, the design science approach will be applied in order to produce artifacts that can be evaluated and improved after each iteration [10].

2.1 Research Project

This project is the product of a collaboration with TietoEvy Care. They offer SaaS products for healthcare providers and they have highlighted that their customers sometimes possess limited technical knowledge about computer science and software, therefore struggling to understand what the product offers when reading the company's REST APIs documentation. For this reason, they have asked us to design a new API playground environment, where users with limited technical knowledge can make use of a more user-friendly environment that provides many live examples, in order to get a better understanding of what the company has to offer. The project will start with a small literature study, which will provide an overview of the current state of research and highlight some guidelines for the design of good API documentation. After that, the iterative development process will start and in order to build a comprehensive plan, the design science guidelines offered by Hevner et al. in [10] are used as a reference.

2.2 Small-scale literature study

A small-scale comprehensive literature study represents the first step of this project. Plenty of research analyzed the shortcomings of traditional API documentation and focused on eliciting the requirements for solid API documentation design. The knowledge gained following this literature review will provide a solid foundation for the development process of this project.

2.3 Design Science Case Study

In order to attempt to find a possible solution to the issue raised by TietoEvy, a case study will be conducted following a design science approach. The iterative process will start with the elicitation of the requirements and the clarification of the objectives of the project. Once the objective is well defined, it will be time for the implementation of

said objectives. The results of the implementation will be tested and refined iteratively. Finally, the resulting artifact will undergo an evaluation. This process will be repeated until a desirable outcome is reached.

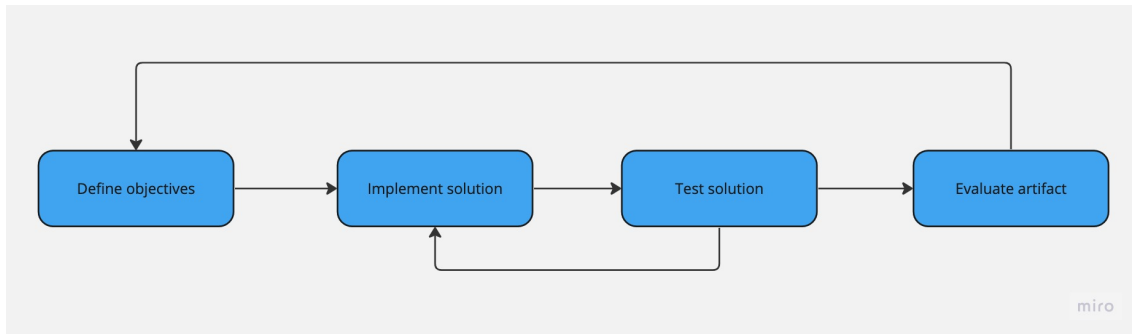


Figure 2.1: Iterative process

2.3.1 Evaluation

The evaluation of the artifacts produced by each iteration will be evaluated with a questionnaire about the usability of the provided solution. The target group for the questionnaire will be composed of 11 employees of the support team of TietoEvy, they are considered non-technical users and it is reflected in the questionnaire results. Each participant will try out the functionalities implemented in the platform and subsequently will answer the questions to provide feedback for their user experience.

Finally, an interview with two developers from the company will also be conducted, in order to gather their opinion on whether the offered solution could present a good tool for solving the issue or if the approach taken overly simplifies the information and does not represent the APIs appropriately.

2.4 Reliability and Validity

As a result of the scope and time constraints for this project, there are some issues that could potentially threaten the reliability and validity of the results.

2.4.1 Reliability

As previously discussed, the aim of this project is to improve accessibility to API documentation for non-technical users and more particularly, in the case brought to us by TietoEvy, for potential customers exploring the different products offered by the company. For this reason, it was initially selected TietoEvy customers as the target for the evaluation process. However, this was not ultimately possible and the evaluation group

will include employees of the support team of the company. While both groups are considered non-technical users, customers of the company are directly affected by the potential findings of this research and therefore would have offered better motivation to answer the questionnaire. This issue could thus affect the reliability of the results.

2.4.2 Validity

The main threat to the validity of the outcomes of this project by the questionnaire, as the findings are dependent on the quality and comprehensiveness of the questions asked. Moreover, the participant's attitude towards the evaluation process could also negatively impact the validity.

2.5 Ethical consideration

The usage of a questionnaire as an evaluation method warrants a few ethical considerations to be made. No personal information or sensitive data will be collected within the questionnaire. The only requirement for accessing the question form is to access using a Google account but under the suggestion of the company it was decided not to collect the email address, as it is not needed for the purpose of this evaluation. Thus, in order to avoid the production of duplicated responses, the questionnaire was set up to only be answered once per iteration.

Moreover, selecting employees within the company as participants for the questionnaire, might in some cases introduce some bias, therefore the participants were purposefully selected from the support team, which has limited contact with the development team.

3 Theoretical Background

The importance of APIs in software development is rather undeniable. They represent the fundamental building blocks of a software product. They provide standard implementations that are well-tested and by doing so, they allow better communication between developers, while also reducing the complexity of software artifacts by favoring interoperability and integration, resulting in improved reusability and efficiency [11].

For this reason, several researchers throughout the last 20 years have focused their efforts on identifying the knowledge that is needed for developers to learn new APIs and how this information should be presented in order to design good API documentation [12].

3.1 Early research

Researchers were already discussing the importance of API documentation towards the end of the 90s when the first documentation generation tools like Javadoc comments for Java were released. While discussing the relevance and importance of such a tool, researchers highlighted the need for a unified document for the description of software interfaces [13]. Moreover, multiple studies underlined the necessity for a middle ground between prose-style documentation and more formal types of documentation and therefore started experimenting with the usage of executable examples based on test cases, that could offer a better learning process for developers [14].

With the rapidly increasing use of component-based software development, the use of APIs incremented even more in early 2000, and as a consequence the importance of good API documentation grew with it [14], because as Bloch mentioned in [15, p.1] "No matter how good an API, it won't get used without good documentation. Document every exported API element: every class, method, field, and parameter."

3.2 Approaches to programming

When discussing the factors contributing to good API documentation design, it is also important to take into account the different attitudes that different developers show toward the programming process. In a study from 2007 [8], Clarke identified three main types of developers' work styles: the *systematic developer*, who gain a thorough comprehension of a technology prior to its utilization, the *pragmatic developer*, who methodically approaches code writing while gaining a sufficient comprehension of the technology to facilitate its practical application, and finally, the *opportunistic developer*, who adopts an exploratory approach when writing code, aiming to uncover possibilities and potential

solutions to effectively address business challenges [8].

These difference in programming work style directly impacts not only the way each developer makes use of the documentation but also what kind of information they require from it. Systematic developers deeply analyze the documentation before trying to implement a new feature, while opportunistic developers are purposely more error-prone and use API documentation in a more selective manner [5]. When it comes to content, systematic developers prefer high levels of detail in the API specification, while other types of developers often value the presence of reusable code examples more. Thus, good API documentation should be structured in a way that can support all the different work styles [5].

3.3 The issues with traditional documentation

Traditional API documentation presents high levels of detail about the API specifications, usually offered through a hierarchical structure, which is, more often than not, hard to navigate. Efficiently organized information plays a crucial role for developers to easily find the relevant material in an API, and developers often report difficulties in locating relevant information inside API documentation, which often results in developers abandoning documentation and turning to Q&A website for seeking information [16].

In addition to the issues with navigation, the main criticism of traditional documentation is the lack of quality in its content, which is often reported to be incomplete, ambiguous, and obsolete. Moreover, while many API specification documents offer usage examples, they are often lacking a proper explanation [4], and more importantly, they present a lack of intent behind the implementation, causing many developers to struggle with understanding the purpose of certain functionalities and the motivations that drove certain design decisions [7].

APIs face frequent changes and updates during their lifetime, therefore keeping up-to-date specifications is a challenging task that requires a lot of resources, which causes numerous documentation sources to become stagnant and obsolete [17]. For this reason, it is also worth discussing whether software architects should be forced to strictly update and maintain their documentation, considering that the complexity of many APIs frequently results in the maintenance cost outweighing the benefits [18].

3.4 Crowd-sourced documentation

As previously discussed, the frequent issues with the accessibility and information quality of official documentation, causes many developers to prefer crowd-sourced information on forums and Q&A website such as Stack Overflow, with some studies reporting that

approximately 10% of the interview developers never even checking the official documentation, on the grounds that the question-oriented approach offered by Q&A websites fits better with their works style and offers a more efficient process for seeking information [16].

Developers value the opinions of other experienced developers as valuable sources of real-world expertise, believing that such insights can guide them in the right direction. They recognize that certain aspects of API usage can only be learned through the shared experiences of other developers and therefore, when faced with the task of selecting an API or addressing specific development requirements, developers utilize the knowledge and insights expressed in the opinions and evaluations of competing APIs to inform their decision-making process. By leveraging this collective wisdom, developers aim to make informed choices and effectively meet their development needs [19].

Despite the more dynamic and interactive nature of user-based documentation, which provides many benefits for developers such as ease of access and better knowledge transfer, the responses posted by developers on Q&A websites exhibit variations in quality, lacking consistent editing or proofreading and as a result, ensuring the accuracy of the information sourced from crowds is challenging [16]. Because of this, numerous researchers have discussed measures for extracting useful scenarios for crowd documentation in order to integrate it in API documentation, as an effort to improve usability and quality of standard specifications documents, while also reducing the costs for maintaining up-to-date information [20].

3.5 Improving the usability API documentation

After careful analysis of the flaws highlighted in the traditional approaches to the design of API documentation, the current state of research offers many guidelines that can be followed to produce good documentation. An effective API documentation design should clearly describe the entry points to the API and establish connections between specific elements of the API and particular tasks or usage scenarios. Successful API learning hinges upon addressing these key issues. Furthermore, API documentation must satisfy the requirements of developers who follow an opportunistic and exploratory approach to their work [6].

3.5.1 Content

As often expressed by developers [4], one of the most important, if not the most important aspect of good API documentation is the quality of its content. To assist developers

in mapping concepts to code, it is essential to highlight text-to-code relations in API documentation, as this approach minimizes the effort required for reading and enhances the identification of pertinent information. Moreover, enabling fast and productive utilization of the API necessitates the inclusion of code examples and seamless integration of try-out functionalities. These features allow developers to test API elements promptly and effortlessly, promoting a more efficient and streamlined development process [6]. However, it is important to note that when an API offers multiple approaches to solving a task, programmers gain flexibility but also face challenges in comprehending the API design. The presence of choice can be advantageous only if the options truly complement each other, rather than adding unnecessary complexities to the overall design [21].

According to the majority of studies, code snippets are considered an essential component of API documentation. Although code snippets typically represent small segments of API functionality, their significance is complemented by step-by-step tutorials, which incorporate several examples of the API's functionalities, often accompanied by screenshots, to guide developers through the process of developing a non-trivial application using the API. This combination of code snippets and step-by-step tutorials proves valuable in facilitating a comprehensive understanding and practical implementation of the API's capabilities [22].

3.5.2 Structure

Upon first access, to enable quick orientation, API documentation should provide a brief overview of the API's main features and overall purpose. Furthermore, in the initial learning process, API documentation must include practical scenarios that demonstrate how to identify entry points into the API. This is crucial since identifying these entry points presents a significant challenge for developers [5]. In order to allow for better navigation and reduce confusion, API documentation should avoid excessive fragmentation and more importantly diminish the amount of structural information as it often hampers the relevant content needed by the reader [4].

4 Research project – Implementation

In order to answer the research question, it was made a decision to create a simple REST API playground platform, where the company's customers would be able to test core functionalities and services. The main reason for using this approach was the company's request and details that were provided during the interviews. As the first step of the research, a detailed analysis of the current market was carried out. The exact same platform was not found in the English-speaking Internet segment. However, Klarna provides the platform for their customers, where it is testable to build an actual template of the web store, which is relevant in terms of the general purposes and goals of our project, but different in terms of the target group, since Klarna platform is mainly focused on technical users.

4.1 Back-end

The programming language for the back-end was C# and namely its framework .NET. Hence, the default IDE was Microsoft Visual Studio, as it delivered powerful support for the needed features.

Since the company provided us with access to their APIs, the back-end solution was based on creating endpoints to these APIs, using supplied API keys and domains, in order to directly request them in the front-end part.

4.2 Front-end

As the project was a web application, we opted to build the front-end solution using JavaScript and the Vue.js framework. Specifically for our project, the issue of developing different versions of the user interface is the most important factor. The interaction of the user, as well as the assessment of its quality, is the key to solving the problem with the perception of information (in this case, documentation).

4.3 The First Iteration

For the first iteration, the basic layout included a header with a placeholder for the navigation bar and a short description of the provided API.

As the first attempt at the implementation, the user interface was inspired by the Swagger UI, by dividing each API request into three different drop-down menus that the user could interact with separately.

PersonnelV2

The personnel management system is designed to efficiently handle and maintain information about individuals within an organization. It provides three main functionalities: creating new personnel records, retrieving a list of all existing personnels, and updating specific personnel information.

Figure 4.2: Header and Description

PersonnelV2

The personnel management system is designed to efficiently handle and maintain information about individuals within an organization. It provides three main functionalities: creating new personnel records, retrieving a list of all existing personnels, and updating specific personnel information.

Add Personnel



Get Personnel



Update Personnel



Figure 4.3: Main layout

In the Add Personnel menu, the user can create new personnel with provided personal information in the input fields for the system and submit it using a submit button to make a POST request to the back-end.

Add Personnel



Get Personnel



Figure 4.4: Add personnel menu

In the Get Personnel menu, the user can get the list of all personnel that have been

registered in the system with the offset value the user can specify how many results to skip and limit how many results to present.

Get Personnel

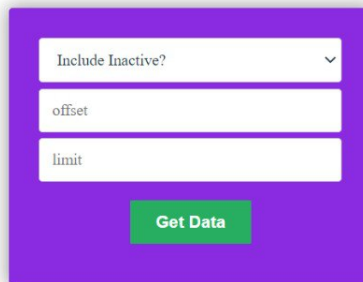


Figure 4.5: Get personnel menu

```
ID: 103
First Name: Mister
Last Name: Enigma
Person ID: 20020202TF02
Signature: @enigma
Title: Dev
HSA ID:
Phone Number:
Email Address:
Can Have Delegation: false
Disabled: false
```

Figure 4.6: Response form

In the Update Personnel menu, the user can update the specific personality with provided id and submit it using a submit button to make a PUT request to the backend. Since it is a put request, it overwrites the previous personality but does not update, then the user has to fulfill all input fields to not lose information.

Figure 4.7: Update Personnel menu

4.4 The Second Iteration

For the second iteration, was decided to first update the navigation bar to interact with different APIs, since in the future the project plans to develop playgrounds for other provided endpoints.

Figure 4.8: Response form

In the second iteration, a key decision was made to develop a functional playground. This involved creating a single form that would store and display recent values to the user. By implementing this approach, users could directly observe the results and make modifications accordingly without switching between different forms which led to excessive segmentation and resulted in confusion for the user.

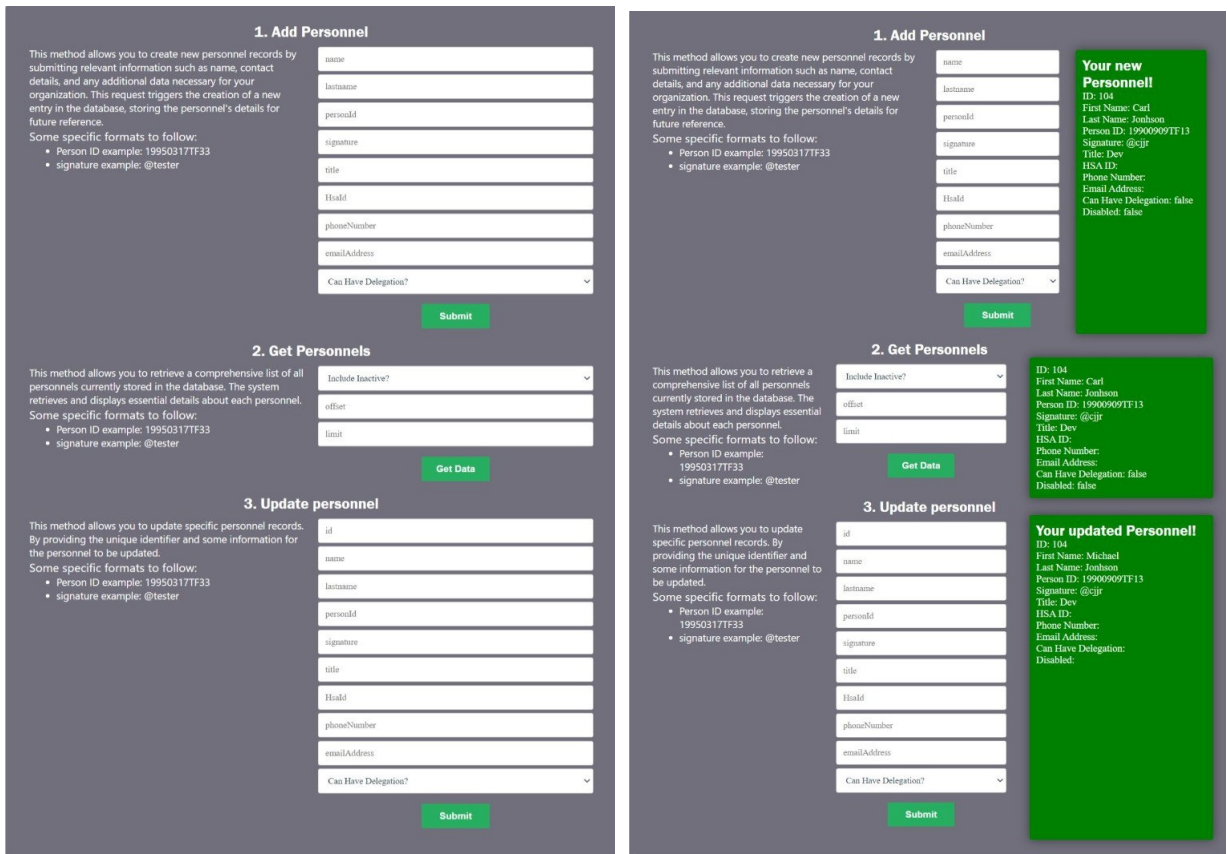


Figure 4.9: Main caption for the figures

More comprehensive descriptions were added to each HTTP request, to provide better instructions to the user about the functionality of some of the needed fields and specify the correct formats for inputs.

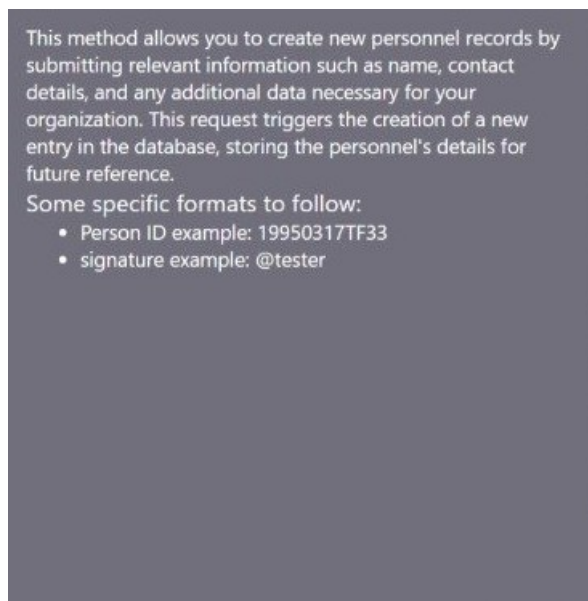


Figure 4.10: Description part

The add personnel method underwent an update to provide a clear representation of the newly added personnel to the user immediately after submitting the form. As a result, the rendered result is now consistently displayed on the page, allowing for further manipulation with other methods.

hide playground

1. Add Personnel

This method allows you to create new personnel records by submitting relevant information such as name, contact details, and any additional data necessary for your organization. This request triggers the creation of a new entry in the database, storing the personnel's details for future reference.

name
lastname
personId
signature
title
Hsald
phoneNumber
emailAddress
Can Have Delegation?

Submit

Your new Personnel!
ID: 103
First Name: Mister
Last Name: Enigma
Person ID: 20020202TF02
Signature: @enigma
Title: Dev
HSA ID:
Phone Number:
Email Address:
Can Have Delegation: false
Disabled: false

Figure 4.11: Add Personnel method

The get personnel method was improved by adding a description field and consistently displaying the retrieved data on the page. This allows for easier utilization of the data with other methods.

2. Get Personnels

This method allows you to retrieve a comprehensive list of all personnels currently stored in the database. The system retrieves and displays essential details about each personnel.

Include Inactive?
offset
limit

Get Data

ID: 103
First Name: Mister
Last Name: Enigma
Person ID: 20020202TF02
Signature: @enigma
Title: Dev
HSA ID:
Phone Number:
Email Address:
Can Have Delegation: false
Disabled: false

3. Update personnel

Figure 4.12: Get Personnels method

The update method was recently enhanced to include a descriptive section, allowing users to provide additional information about the changes being made. Along with the description, a results form was also introduced, enabling users to interactively modify the data and observe the outcomes in real time.

Close

3. Update personnel

This method allows you to update specific personnel records. By providing the unique identifier and some information for the personnel to be updated.

Your updated Personnel

ID: 103
 First Name: Enigma
 Last Name: Mister
 Person ID: 20020202TF02
 Signature: @enigma
 Title: Developer
 HSA ID:
 Phone Number:
 Email Address:
 Can Have Delegation:
 Disabled:

Figure 4.13: Update Personnel method

4.5 The Third iteration

API Playground
Toggle Navigation

ProvideAdministratorV2

The Provider Administrator REST API offers a range of powerful functionalities to manage administrators within your system. With this API, you can effortlessly retrieve information about all administrators or obtain details of a specific administrator by their unique ID. Creating a new administrator is made simple through a POST request, allowing you to provide the necessary details and receive a newly created administrator object with a generated ID in response. Updates to existing administrators can be easily performed using a PUT request, where you can modify their details and receive an updated administrator object in response. Deleting administrators from the system is straightforward as well, achieved by sending a DELETE request with the specific administrator ID. Additionally, the API provides capabilities to manage admin roles. You can add a new admin role to an administrator using a POST request, providing the relevant details, and receiving the newly added admin role object. Modification of admin roles can be done through a PUT request, allowing you to update the details of a specific admin role for an administrator. If an admin role needs to be removed, a DELETE request can be sent with the respective administrator and role IDs. With these comprehensive functionalities, the Provider Administrator REST API empowers you to efficiently handle the creation, retrieval, modification, and deletion of administrators and their associated roles within your system.

Get All Provider Administrators

This request allows you to get a list of all the administrators. No input is needed as the request will load all the available administrators.

Add Provider Administrator

This request can be used to add a new Administrator to the system. To forward the request, you need to specify a person ID (personnummer), first name, last name, signature and title. A role should also be selected among the two provided options: EC.SystemansvarigUtförare or EC.PersonalAdministratör

For testing purpose HsaID, phone number, email and active from/active to can be left empty ad they are not required.

Some specific formats to follow:

- Person ID needs to follow the forma XXXXXXXX-TFXX. This is because for testing, no real personal numbers can be used.
- The signature needs to include "@", ex. "@test"

Figure 4.14: Update Personnel method

The first decision that was made at the start of the third iteration was to integrate the Vuetify framework into our project. This is because the Vuetify framework provides many different pre-made components that are well-tested and offer a large variety of features

that can improve the overall look of the user interface. In particular, the *expansion panel* component was used to hide the testing area of each API request in order to reduce the cluttering on the page and provide a better experience for the user, who can read the description of each functionality and arbitrarily decide if they want to test it by clicking on the expansion panel.

This issue with cluttering was mainly noticed after adding a new API called *Provider-Administrator*, which provided more endpoints compared to the previous API, and therefore the approach taken in the previous iteration proved to not be efficient and caused some unwanted confusion in the overall look of the UI.

The layout for the GET requests was also changed as it cause cluttering for databases that contained many items. The new layout includes a scrollable section where all the data retrieved from the GET request is shown, in order to limit the space taken by the output of the request.

Get All Provider Administrators

This request allows you to get a list of all the administrators. No input is needed as the request will load all the available administrators.



Figure 4.15: Update Personnel method

While using input forms instead of having users manually manipulate the JSON data for each request (as it is done in environments like Swagger) provided an improved user experience, it was also creating a level of abstraction that was too distant from the actual API documentation. For this reason, this iteration also included a live example box, where a reactive variable was used to show the user how filling the input fields of the forms would affect the JSON data in real-time. This feature was considered important as it provided technical depth to the documentation while also providing a simplified testing approach.

Try me!

Person ID 19901219TF39	
First name Test	Last name Test
Signature @test	Title Tester
Hsa ID	Phone Number
Email	
Active From	Active To
Role Name EC.SystemansvarigUtförare	
ADD PERSONNEL	

Example data

```

{
  "personId": "19901219TF39",
  "firstname": "Test",
  "lastname": "Test",
  "signature": "@test",
  "title": "Tester",
  "hsaId": "",
  "phoneNumber": "",
  "emailAddress": "",
  "ProviderAdminStatus": {
    "activeFromDate": null,
    "activeToDate": null,
    "roleName": "EC.SystemansvarigUtförare"
  }
}

```

Figure 4.16: Live example for the POST request

After getting the feedback from the second iteration, it was quickly noticed that users were still not satisfied with error validation. Therefore, in order to provide better feedback to the user after completing each task, this iteration also focused on improving error validation and providing confirmation of success if a task was correctly performed.

Try me!

Something went wrong!

Person ID 199019192039	
Invalid Person ID	
First name Field is required	Last name Field is required
Signature Field is required	Title Field is required
Hsa ID	Phone Number
Email	
Active From	Active To
Role Name Field is required	
ADD PERSONNEL	

Example data

```

{
  "personId": "199019192039",
  "firstname": "",
  "lastname": "",
  "signature": "",
  "title": "",
  "hsaId": "",
  "phoneNumber": "",
  "emailAddress": "",
  "ProviderAdminStatus": {
    "activeFromDate": null,
    "activeToDate": null,
    "roleName": ""
  }
}

```

Figure 4.17: Error validation for the POST request

An alert system was implemented to notify the user when something is wrong and stronger rules for the validation of the input fields were implemented using regular expressions. For example, in the case of the *Person ID* field, for the purpose of testing

no real personal numbers could be used, therefore the input has to follow the format YYYYMMDD-TFXX, and the following set of rules was used for the validation:

```
const personIDrules = {
  required: value => !!value || 'Field is required,
  personId: value => /^(19|20)\d{2}(0[1-9]|1[0-2])
    (0[1-9]|1[12][0-9]|3[01])TF\d{2}$/.test(value) || 'Invalid
  Person ID'
}
```

Listing 1: Validation rules for Person ID

Another issue that was found in the previous iterations is that after completing a POST request there was no confirmation that showed if the request was successfully completed and which ID was assigned to the new object. This caused confusion among the user and disrupted their workflow. In order to address this issue, a new alert that shows the ID of the newly created object was added to the POST request.

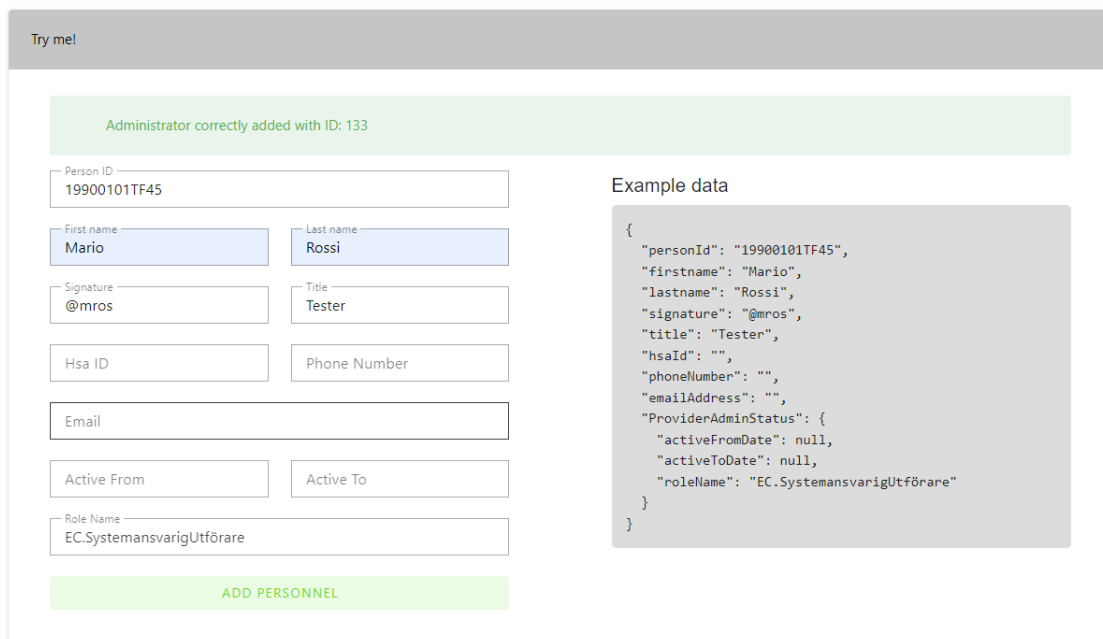


Figure 4.18: Success alert for the POST request

5 Results

The questionnaire for the evaluation has the same question for all three iterations and makes use of a Likert scale with values ranging from 1 to 5, with 1 being the lowest score and 5 being the highest. The full questionnaire can be found **appendix A**.

5.1 The First Iteration

Here is the result of the evaluation of the first iteration:

- **What is your level of expertise with APIs?**

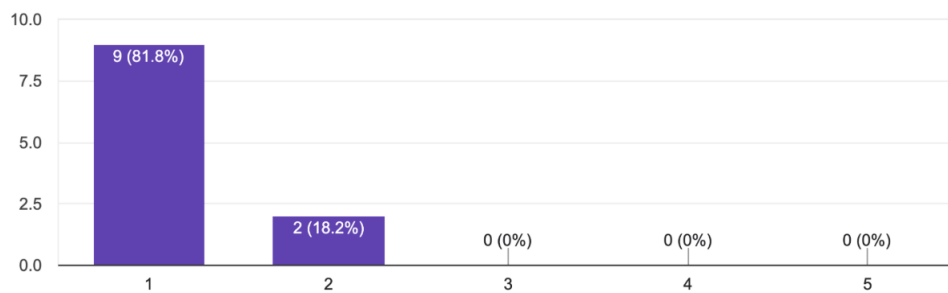


Figure 5.19: Result of the first question

- **How would you rate your overall usage impression?**

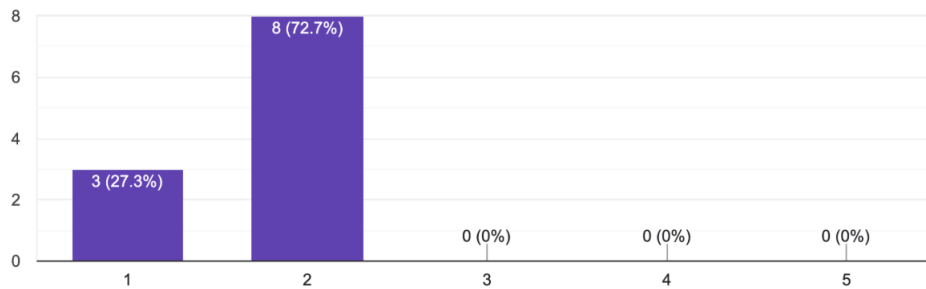


Figure 5.20: Result of the second question

- **Was it easy to navigate through the playground?**

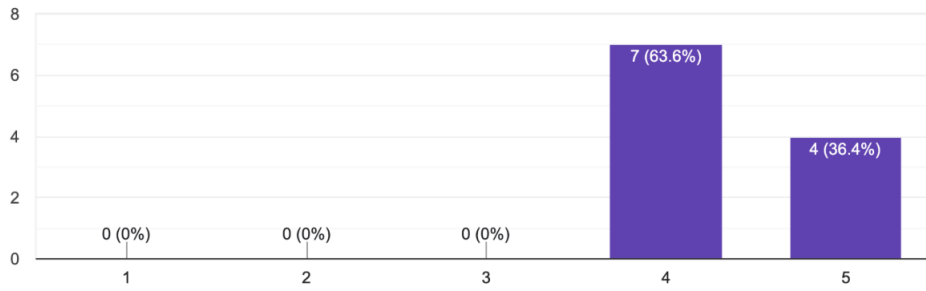


Figure 5.21: Result of the third question

- **How would you rate your efficiency?**

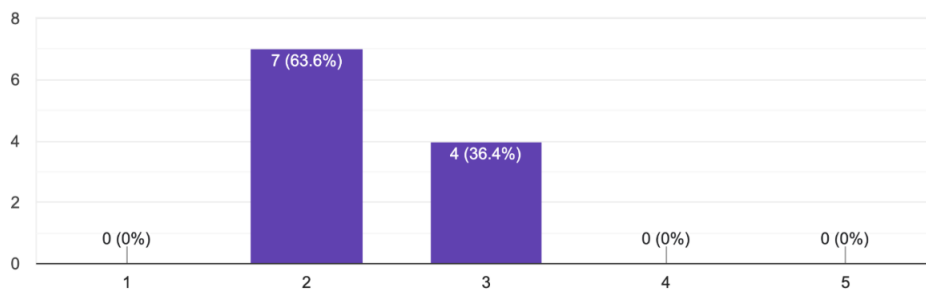


Figure 5.22: Result of the fourth question

- **How would you rate the platform's performance level?**

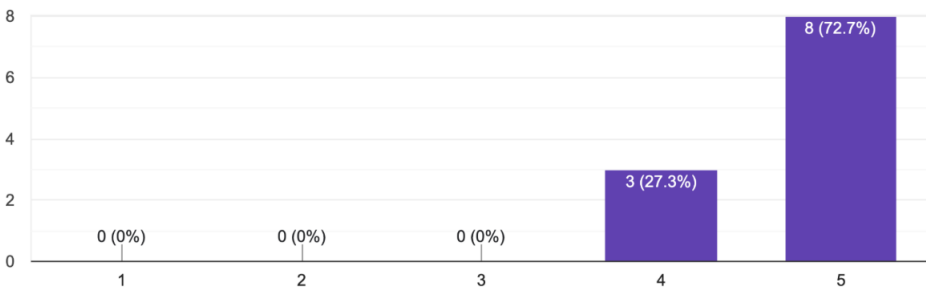


Figure 5.23: Result of the fifth question

- **How would you rate the information presented on the playground platform (tasks descriptions, overall information)?**

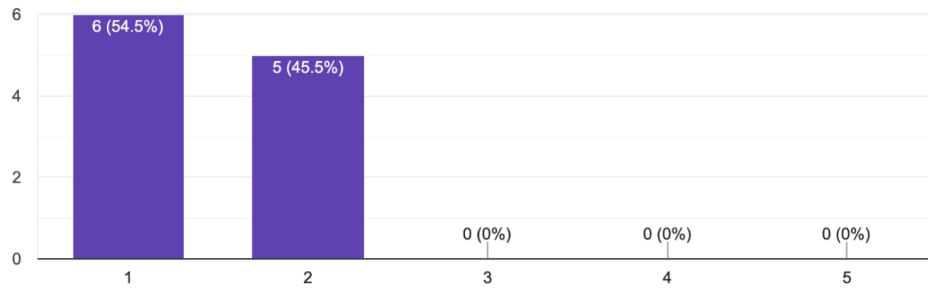


Figure 5.24: Result of the sixth question

• **How would you rate error handling?**

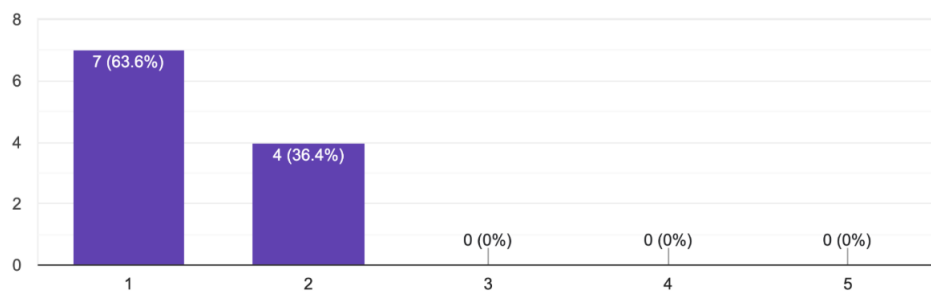


Figure 5.25: Result of the seventh question

• **Did you experience any accessibility issues?**

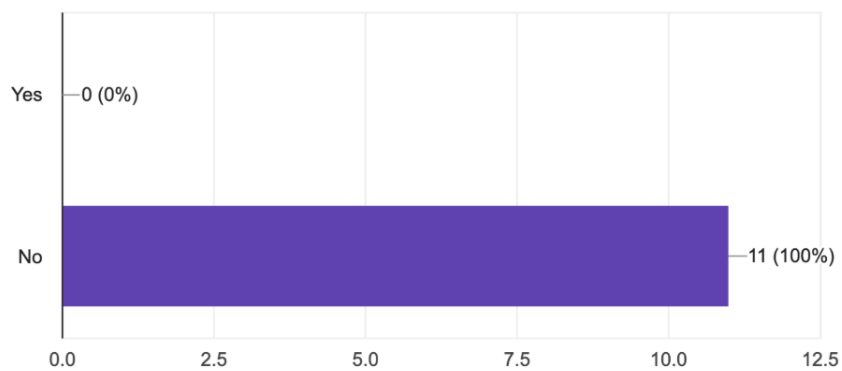


Figure 5.26: Result of the eighth question

• **How satisfied were you with the platform?**

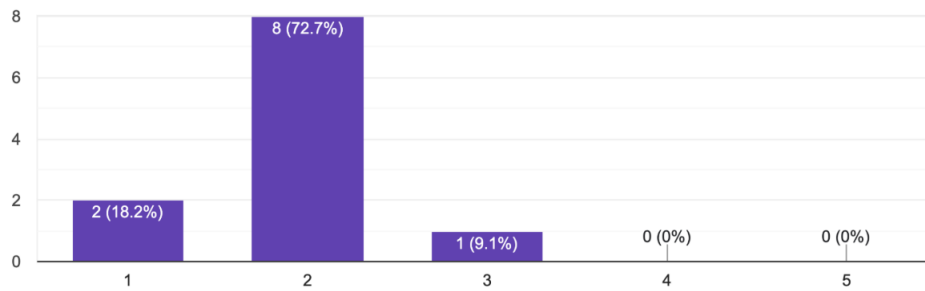


Figure 5.27: Result of the ninth question

- **Did you experience any difficulty using any particular features?**

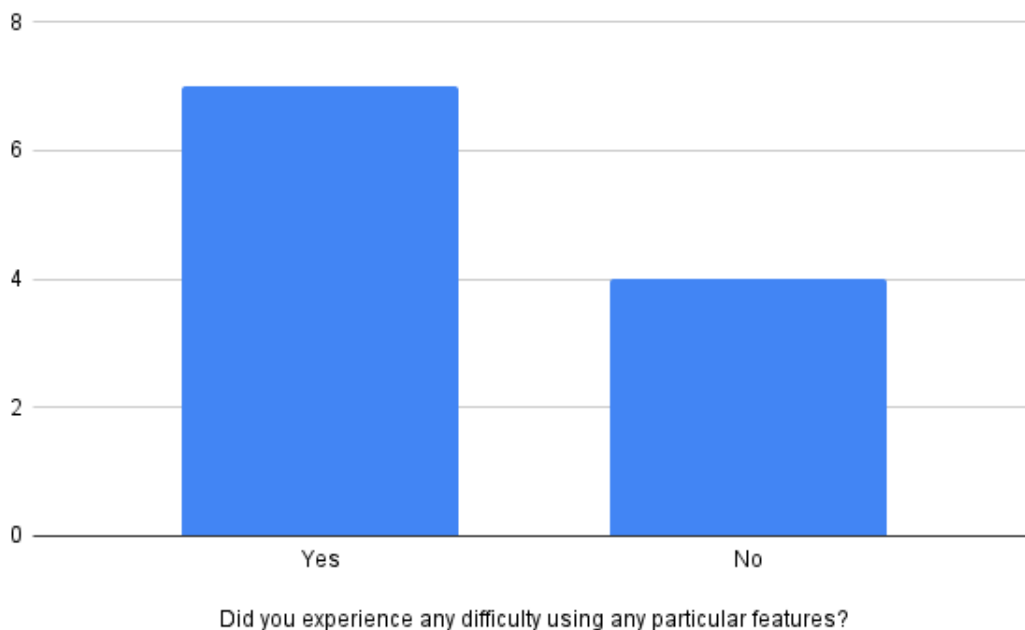


Figure 5.28: Result of the tenth question

- **If you encountered any difficulties with a particular feature, can you closely describe them, please? What feature it was and what was the problem?**

Since this question was not mandatory, during this first iteration we only received four answers to this question. The answer reported that users experienced difficulties with the usage of *Add Personnel* and *Update Personnel*, and in particular one user specified that while trying *Update Personnel* he had trouble figuring out what "id" field of the request was referring to.

- **What additional suggestions or improvements do you have for the platform developers to make it better for users like you?**

One user reported that the functionalities offered by the APIs could use more descriptions.

5.2 The Second Iteration

This section provides questionnaire results for the second iteration:

- **What is your level of expertise with APIs?**

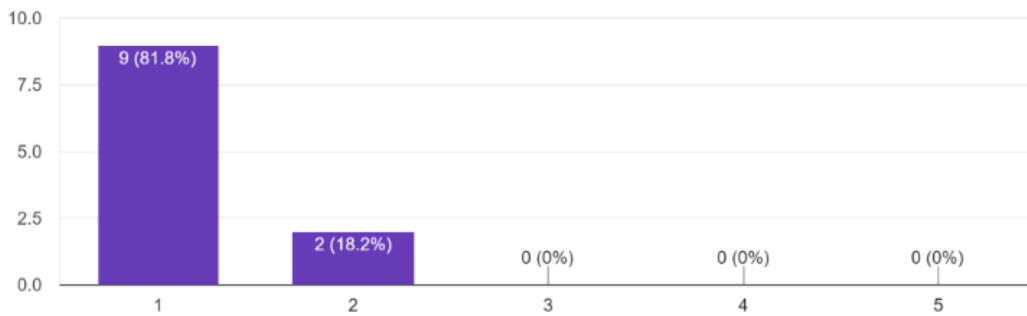


Figure 5.29: Result of the first question

- **How would you rate your overall usage impression?**

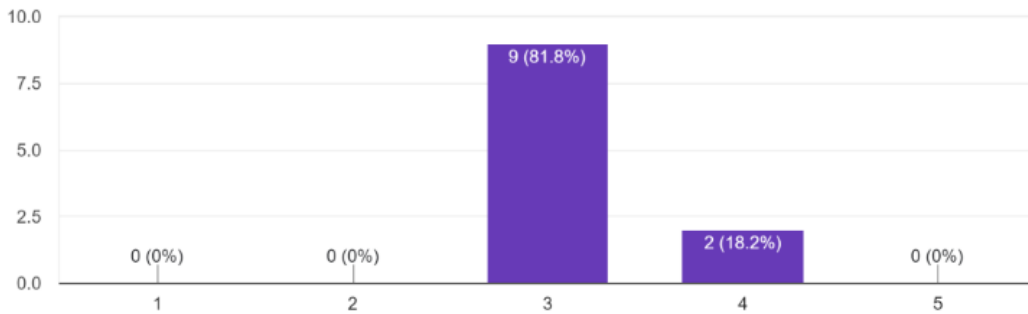


Figure 5.30: Result of the second question

- **Was it easy to navigate through the playground?**

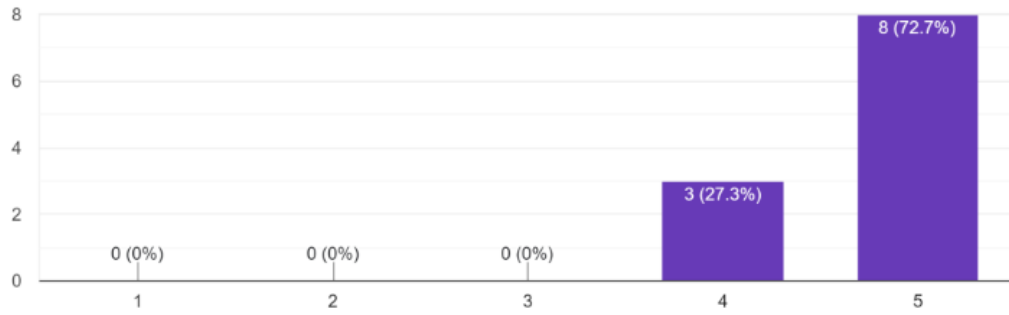


Figure 5.31: Result of the third question

- **How would you rate your efficiency?**

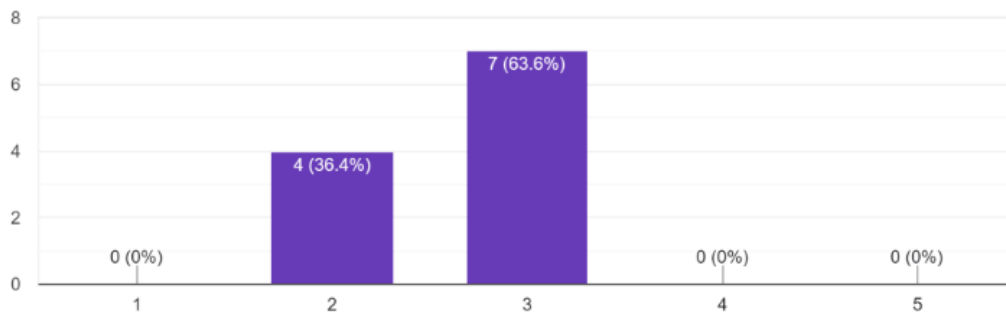


Figure 5.32: Result of the fourth question

- **How would you rate the platform's performance level?**

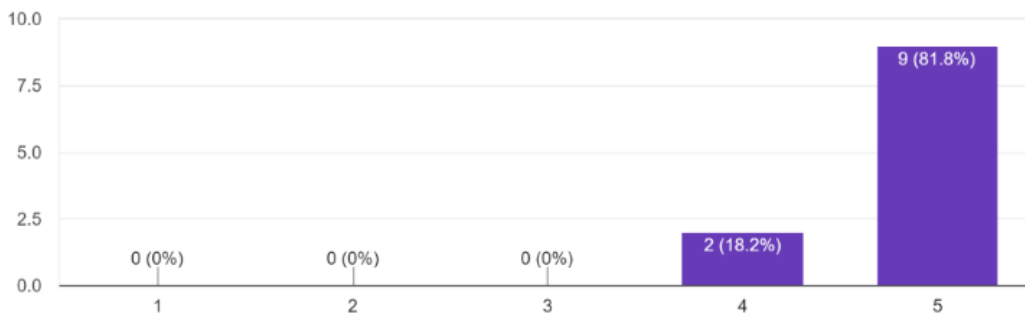


Figure 5.33: Result of the fifth question

- **How would you rate the information presented on the playground platform (tasks descriptions, overall information)?**

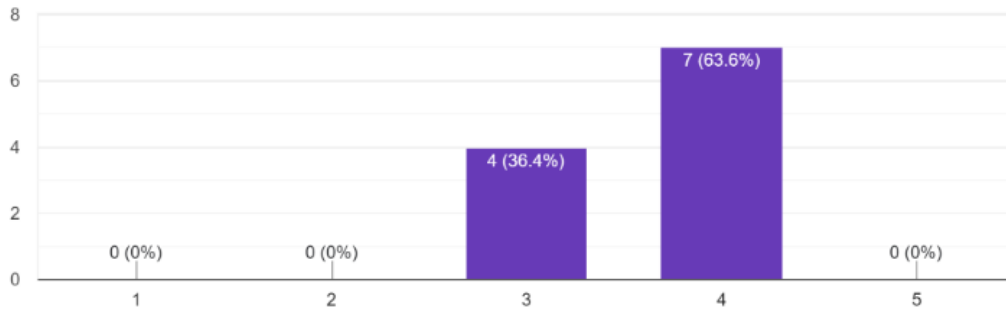


Figure 5.34: Result of the sixth question

- **How would you rate error handling?**

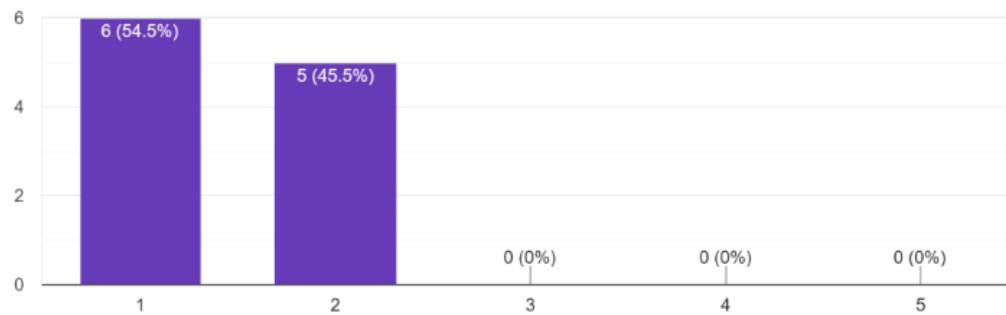


Figure 5.35: Result of the seventh question

- **Did you experience any accessibility issues?**

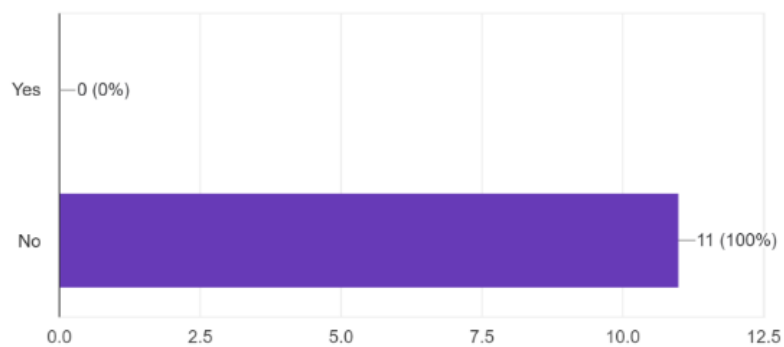


Figure 5.36: Result of the eighth question

- **How satisfied were you with the platform?**

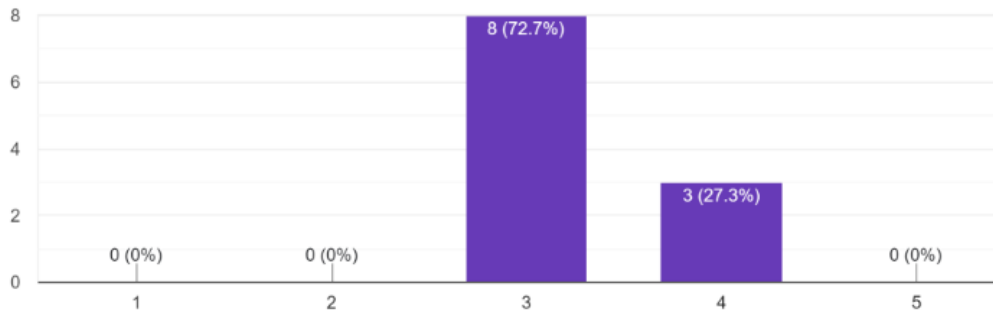


Figure 5.37: Result of the ninth question

- **Did you experience any difficulty using any particular features?**

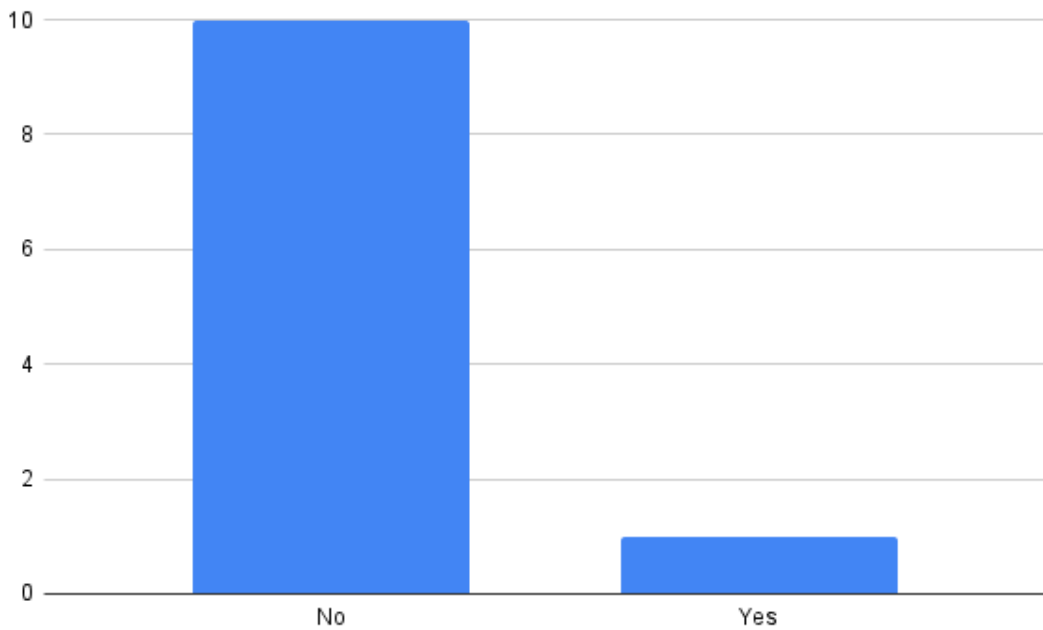


Figure 5.38: Result of the tenth question

5.3 The Third Iteration

Here is the result of the evaluation of the third iteration:

- **What is your level of expertise with APIs?**

What is your level of expertise with APIs?

11 responses

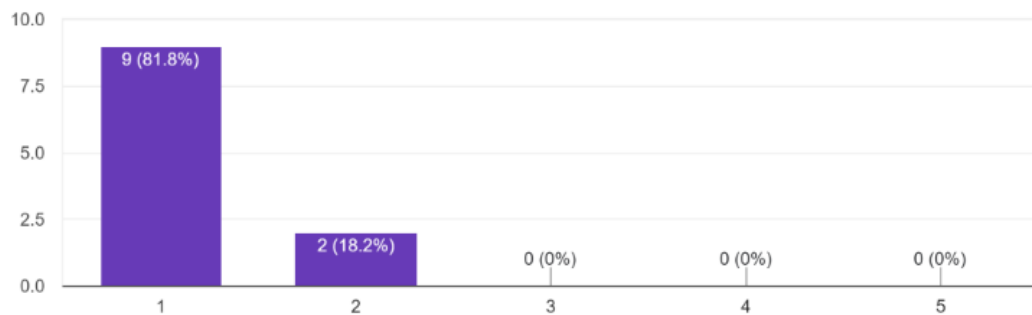


Figure 5.39: Result of the first question

- **How would you rate your overall usage impression?**

How would you rate your overall usage impression?

11 responses

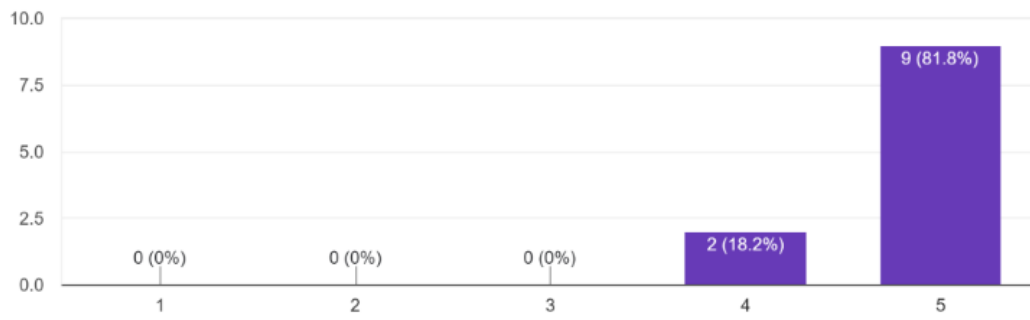


Figure 5.40: Result of the second question

- **Was it easy to navigate through the playground?**

Was it easy to navigate through the playground?

11 responses

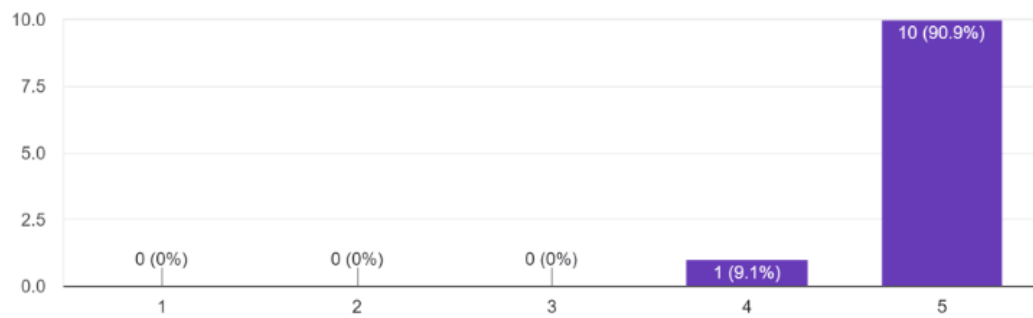


Figure 5.41: Result of the third question

- **How would you rate your efficiency?**

How would you rate your efficiency?

11 responses

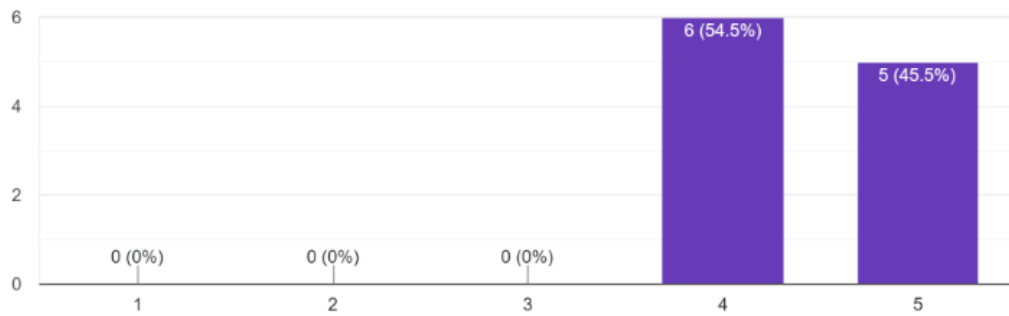


Figure 5.42: Result of the fourth question

- **How would you rate the platform's performance level?**

How would you rate the platform's performance level?

11 responses

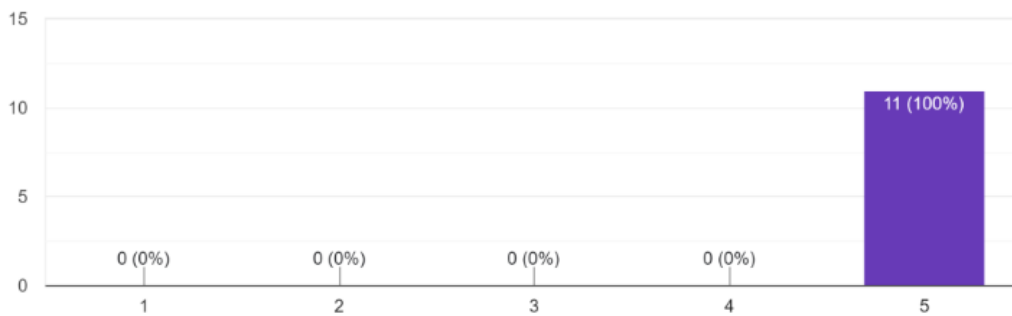


Figure 5.43: Result of the fifth question

- **How would you rate the information presented on the playground platform (tasks descriptions, overall information)?**

How would you rate the information presented on the playground platform (tasks descriptions, overall information)?

11 responses

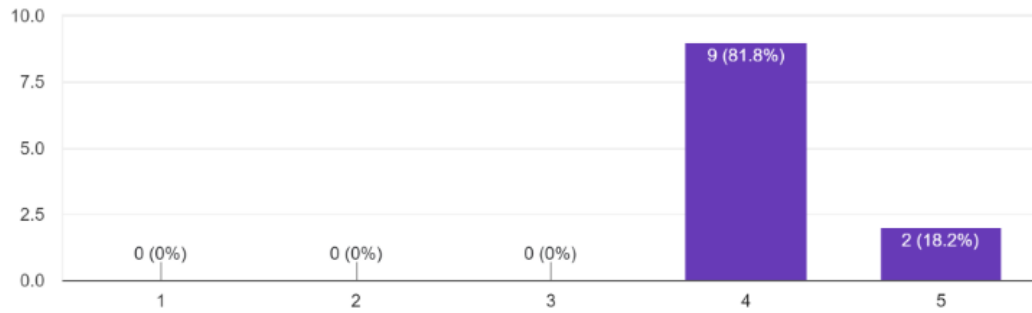


Figure 5.44: Result of the sixth question

- **How would you rate error handling?**

How would you rate error handling?

11 responses

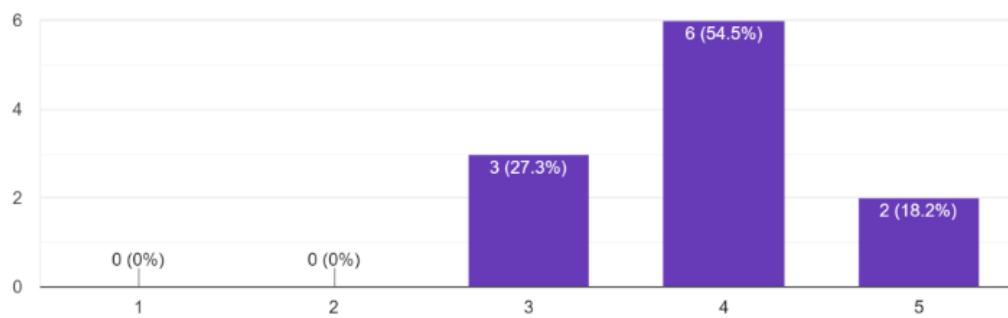


Figure 5.45: Result of the seventh question

- **Did you experience any accessibility issues?**

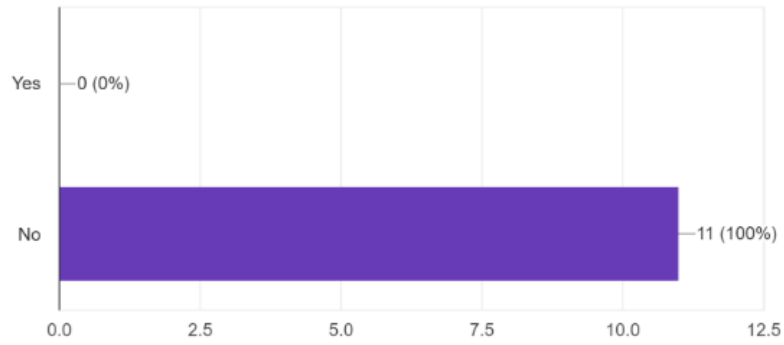


Figure 5.46: Result of the eighth question

- **How satisfied were you with the platform?**

How satisfied were you with the platform?

11 responses

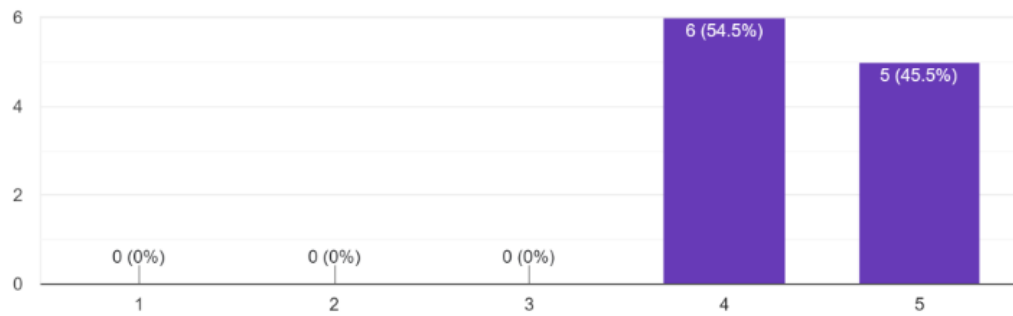


Figure 5.47: Result of the ninth question

- **Did you experience any difficulty using any particular features?**

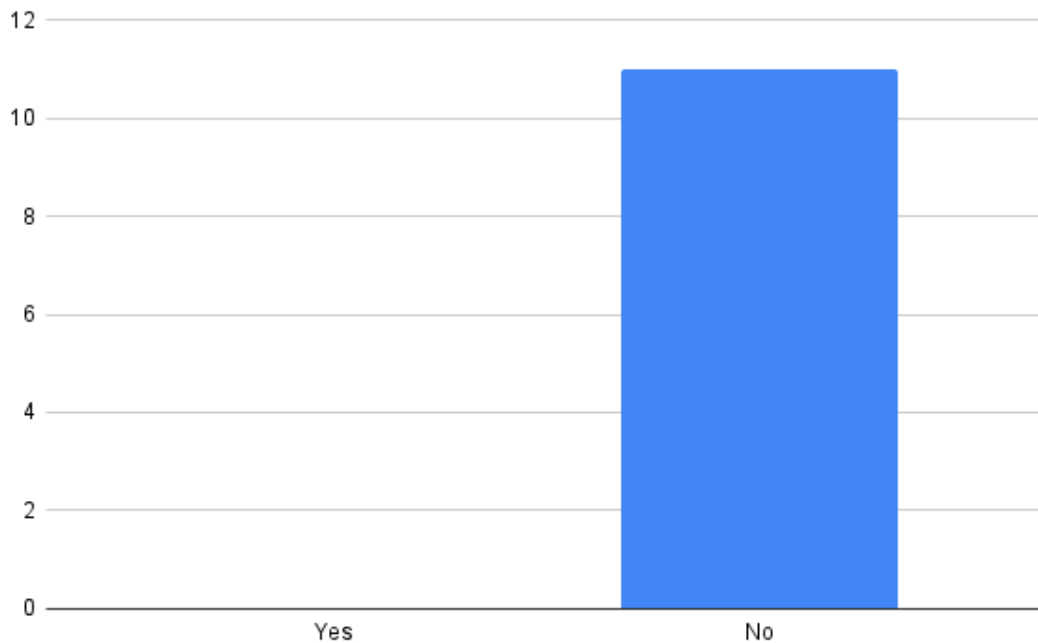


Figure 5.48: Result of the tenth question

5.4 Final interview with developers

This section will provide a transcript of the interviews conducted with two senior developers at the company. While the user questionnaire was focused on the usability of the API, these interviews were focused on getting the developer's view on whether this approach could be beneficial or if the costs would outweigh the benefits.

What are your initial impressions about the API Playground?

- Developer 1: It looks really simple. I think it feels a bit simpler than something like Swagger since you're able to fill in all the data through a form rather than directly editing the JSON like you would in Swagger. So I think it would probably be easier for non-technical users to try it out and understand what you need, like a bit about how the API works. And that's my initial impression. And I also think this approach makes the documentation clearer overall for people that do not have much experience with REST APIs. Really simple for the user feels like.
- Developer 2: I think it looks good. I like how you provide a live example of how the JSON data of the request will look next to the forms. Everything in general felt very reactive and smooth. It was also useful that the POST request shows a confirmation when an object is correctly added and shows the ID so that users can easily use it to try and play around with the other functionalities.

Did you find the interface intuitive and user-friendly?

- Developer 1: Yeah, I find that it's intuitive and user-friendly.
- Developer 2: Yeah, I think so.

Were there any specific features that you found useful or lacking?

- Developer 1: I think, like, the simplicity is the key feature of the key sort of thing that that is good with it. So I don't think it's so much about having a lot of features. What is important it just that you can try out all the functionalities that the API offers and that everything is really simple and intuitive. I think error handling is important. And maybe that's something that can be improved. It should be very clear to the user what they have done wrong and how they should fix it. I guess that's something that can be implemented better later on.
- Developer 2: Probably the live example of the JSON data is the functionality that stood out the most to me so far. Does a good job of showing more technical details while still keeping the approach simple enough for non-technical users to understand.

How well did the API playground integrate with the API documentation?

- Developer 1: Yeah, that's a good question. Yeah, I guess I mean, for very technical persons that are very used to working with APIs, maybe the swagger is what they want to use because it's what is kind of a standard solution. And this might be more specific, for the non-technical, more domain persons that want to understand what they can use. And ideally, sort of the same information could be reused in both so that some of the descriptions could be taken from the swagger. But I think I think it can work well together. Because this is another level, of user-friendliness than the swagger thinking. I think that that's, that's useful. So yeah, I think so.
- Developer 2: I think it integrates well. Maybe it should provide some links that forward to the more detailed documentation, that should provide more coverage for all use cases.

Do you think that the Playground correctly displays the APIs functionalities?

- *Developer 1*: What I've seen so far it feels like it's correctly displayed. I mean, you have both representations both This input field but you can also see exactly the data that will be in the JSON format there. So that fields. Correct. And as you said,

if you, when you have posted something, you probably want to get the ID that's generated. And so I mean, when you have that, and it's you also get told, like all the data's? No, I feel that, that it's correct.

- Developer 2: Yeah, I think the description overall is pretty useful. Of course, they are simplified, but they still provide a good enough representation of the API.

What future improvements would you like to see?

- *Developer 1*: One thing that I'm just thinking about this is what the database for a platform like this one could look like, what what what will this be run against? One thing could be that there was maybe some Azure environment when you have this kind of playground environment, where maybe the data is reset every night or something like that. The back-end solution for this should be designed in a way so that people can play around without ever destroying anything. Like, you want to be able to try to create stuff and remove stuff. Maybe there could be a reset button, or I don't know, there are probably a lot of possibilities to use, like containers and other tools to able to have a safe kind of sandbox for the playground. But to me, I think this is a nice beginning to have this really user-friendly interface and to get validations. Then I guess also one thing is it would be nice if the documentation is tied together with the swagger and the code so that when you change the code, this should ideally change and that you don't have to remember to do some other stuff here. It is very important for the documentation to actively follow each iteration
- Developer 2: It's hard to say, but it feels like it could be somewhat easier. But nothing specific really comes up to my mind at the moment.

API documentation is often hard to maintain. Would you consider implementing a solution similar to the one presented in this study, or do you think the costs would outweigh the benefits?

- *Developer 1*: Yeah, it really depends on the implementation. And because today, we have some other resources that are hard to maintain. We have some PDF documents that sort of list all endpoints that we have and what API packages are like they are sold in different packages. So customers will buy a package with some endpoints. And that kind of document has proven to be not a good way because it doesn't get updated with the code. So I think that an approach like this could be beneficial. But I think it has to be done In a way that is tied together with the code with the other documentation. And that it's not a lot extra, I think that maybe you need to have

some more detailed explanation text that explains an API a little bit clearer here for a non-technical person. But I still think that that information should be very close to the code. Maybe that should be somewhere in the controller class or something so that when you do code changes, you see that text, it's not in a totally different place in the code, so you need to remember to go to some completely different place and do some updates. That's kind of hard to remember. So I think it can be done in a way that it's not too hard to maintain. But it is definitely a process that has to go strictly alongside the development process, so that it is not disconnected from the rest, because if it's something completely on the side of everything else, then it will be too much maintenance, and after a while it might end up not showing the complete truth about what the API does, because we will have some changes that we have forgot to make in the documentation.

- Developer 2: I think as long as this is easy to like, update and like, keep what will say like it keeps up with the code then it needs to be some fast connection to the code. So this can be maintained. Because it's often easy to like what we often do like we implement something, then it's some documentation. And then there are code changes and the documentation doesn't change. That can always forget the documentation like the boring porch. So some like this, it should be easily connected to the code somehow. So it can be kept up to date.

6 Analysis

In this section, the result of each iteration will be cross-examined, in order to closely analyze the effects of the change that were made after each iteration.

6.1 Comparing the outcome of first and second iteration

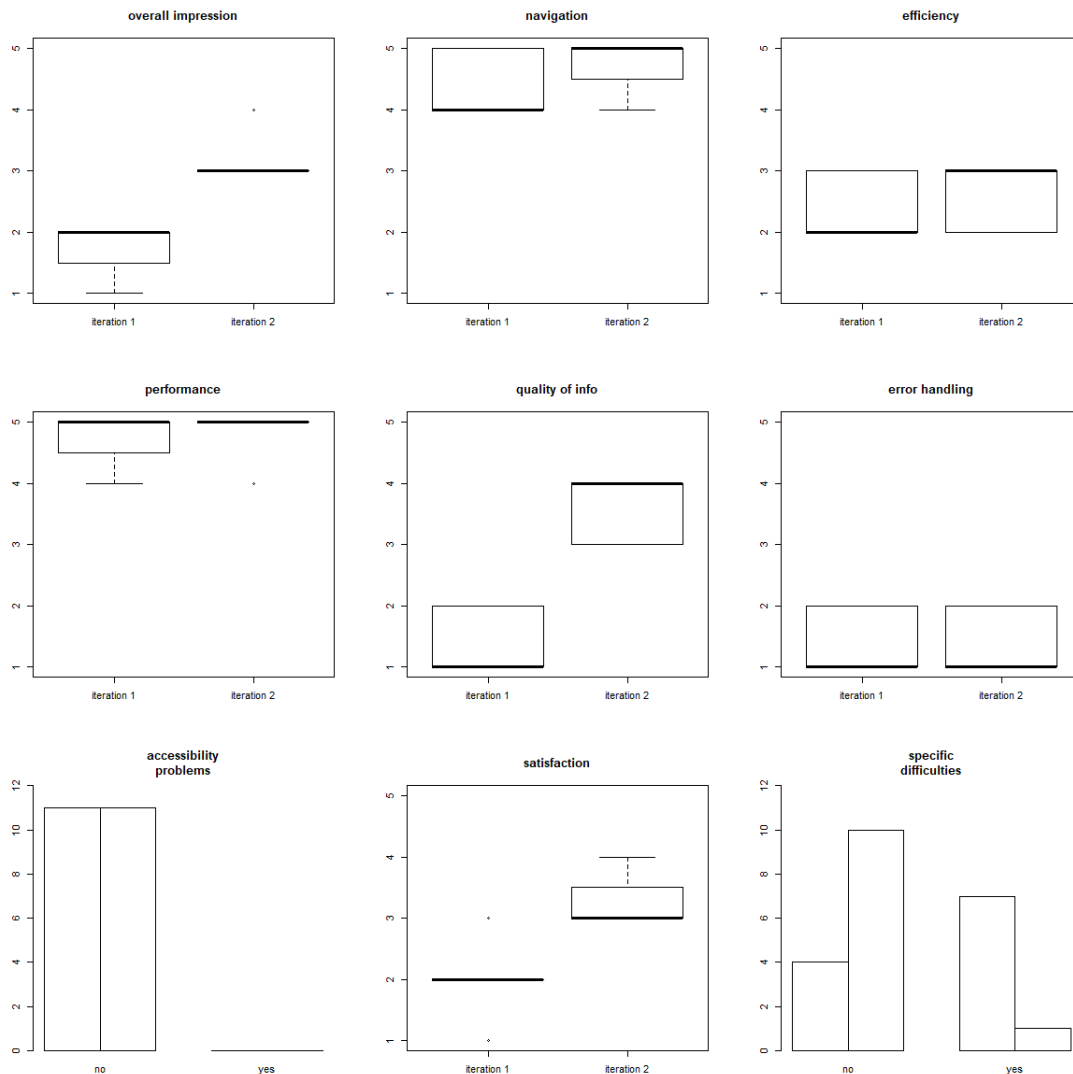


Figure 6.49: Result of the first two iterations

The result for the overall usage impression was rather low, given the very basic initial implementation. Nevertheless, the changes that were made as a result of the evaluation of the first iteration resulted in an improvement in this category.

The navigation was already considered efficient from the first iterations and while there is a slight improvement that can be observed in the second iteration, given the reduced scope of this project these results do not provide much value to the issue under

analysis.

The users reported low-efficiency scores, which suggests that they had trouble figuring out how to correctly use the functionalities. While the result of the changes made to the second iteration does show a small improvement, the outcome is still not desirable.

Similarly to the question about the ease of navigation within the page, this question provide a high score immediately from the start, since the lightweight nature of the application at this point of the development process was not a threat to the performance of the platform. For this reason, the results obtained from the responses do not provide reliable results.

The amount of information and description given in the first iteration was limited, which explains the low score obtained after the evaluation. The additional information provided in the second iteration showed an observable improvement in the perceived quality of the content.

The simple overall structure led to users across both iterations did not experience any accessibility issues.

The overall satisfaction, similarly to the quality of content and the overall usage impression reported poor scores in the first iteration, which then observably improved as a result of the changes made to the second iteration.

As a consequence of the improvement made in the second iteration, the number of difficult occurrences for particular features was considerably reduced. In the first iteration, users gave more details about which feature was causing the biggest issue, with most of the participants reporting issues with the *Add Personnel* and *Update Personnel* due to confusion about the needed fields and formats. These issues were addressed and solved in the second iteration with the improvement of the descriptions.

In conclusion, the introduction of a navigation bar for switching between APIs, the addition of more detailed descriptions, and the optimization of the task completion flow effectively addressed some of the issues that users presented after testing the implementation of the first iteration. But while these results are promising, they have yet to represent the desired outcome of the project.

6.2 Comparing the outcome of all the iterations

For the final analysis of all three iterations, a box plot was chosen. As it provides tidy and focused information in one place.

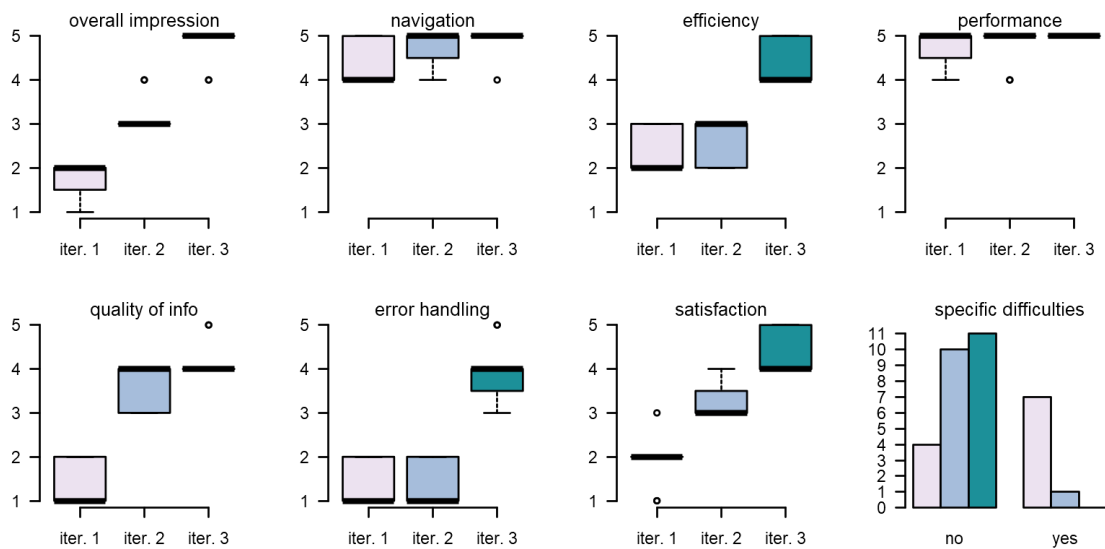


Figure 6.50: Result of the three iterations

For the entire development period, a consistent increase in outcomes was observed across all the specific aspects that were thoroughly examined.

First of all, the overall impression increased from a median value of two during the first iteration to a steady value of five by the third iteration. By the 3rd iteration, it was possible to achieve the optimal look of the visual aspect as well as the overall performance, which contributed to a noticeable increase in the values precisely between the second and third iterations

The overall median values related to the quality of navigation did not noticeably change, since in the first iteration the overall results were quite high. By the final iteration, it was possible to achieve a confident almost maximum possible result. However, following hardly noticeable differences, the responses do not provide reliable results.

The amount of information and description in the first iteration was limited or was not provided at all which led to the lowest possible median value. Detailed attention was paid to this aspect, leading to improvements in both the quality and overall content, which resulted in a noticeable enhancement. However, despite these efforts, the same dramatic growth could not be achieved by the third iteration since the overall result by the second iteration was already high. Although some users did observe a general improvement, the median result remained unchanged.

During the first and second iterations, the overall efficiency remained relatively low, indicating that users encountered significant issues when utilizing the general functionality. However, by the conclusion of the third iteration, significant and positive improvements were attained. As a result, the overall median score reached 4 points, and a substantial number of users rated its effectiveness as 5, which is a perfect outcome within this

category.

Similarly to what was observed in the case of the navigation aspect of the platform, the feedback obtained in regard to the performance of the application was very positive right from the start, due to the low overall complexity of the application which proved not to be a burden for the system. Therefore, while the results were positive, they do not provide relevant qualitative data and the performance aspect of this type of application should be investigated further in a higher complexity environment.

The results for the error handling aspect were the same in the first and second iterations. Initially, error handling was implemented in a basic manner as it was not deemed crucial. However, in the third iteration, specific focus was given to this aspect, resulting in significant improvements. The median rating for error handling reached an excellent score of 4, moreover, a smaller number of users rated it at the highest level.

The overall satisfaction with the project increased markedly with each successive iteration. The median values gradually increased from 2 in the first iteration to 3 in the second and 4 in the third, accordingly. This step-wise growth directly reflects the effectiveness of the changes that led to a noticeable improvement by the final iteration.

Another positive result was shown in the number of reported difficulties with specific tasks. In the first and second iteration users encountered issues while performing certain operations on the platform, with the POST and PUT requests being regarded as the most complicated tasks. During the third iteration, more focus was directed at solving these issues and this was reflected by the feedback received after the third iteration, where no specific difficulties were reported by the participants to the evaluation questionnaire.

In summary, the final iteration witnessed significant improvements primarily attributed to the introduction and implementation of a new layout and structure. Notably, the integration of a comprehensive error handling system, encompassing more detailed rules and representation, had a positive impact on the project.

6.3 Analysis of interviews with developers

The developers' initial impressions were noteworthy, they both found the interface intuitive and user-friendly and underlined the simplicity as the key feature. One developer expressed that the application presented a simplified approach compared to conventional Swagger implementations, as users interacted with intuitive input forms rather than directly manipulating JSON files. Another developer took note of the application's overall form reactivity and its smooth user experience. He also highlighted the convenient presentation of request confirmations, including the user ID, which greatly facilitated further interactions.

In terms of future enhancements, the first developer offered valuable recommendations on updating the back-end solution, as the current one used endpoints to an existing API. According to their suggestion, the back-end solution could leverage the Azure environment, enabling users to interact with real-time data as a sandbox model. At the same time, both developers underlined that future development should be mainly focused on maintains aspect. Since API code and its documentation usually update separately.

7 Discussion

This project focuses on trying to improve the accessibility of API documentation for non-technical users. The company that contributed to this study, highlighted the difficulty of communicating their services to potential customers, who often lack the technical expertise needed to accurately interpret traditional documentation. Thus, a case study was run on some of the RESTful APIs offered by the company, with the objective of implementing an API playground environment that could offer simplified live examples of the API's usage, in pursuance of a possible solution to the identified problem.

In order to answer the first research question (**RQ1**), an initial small-scale literature was conducted with the purpose of analyzing what previous research has identified as the most important factors contributing to good API documentation design, with aspect such as quality of the descriptions, ease of navigation and live examples being regarded as the most deciding factors for the success of API documentation.

These findings were then used as guidance for eliciting the requirements and objective of the API playground platform, and by following the guidelines proposed by Hevner et al. in [10], a design science approach was used to structure the iterative development process of the application. With the purpose of answering the second research question (**RQ2**), after each iteration a questionnaire was used to guide the evaluation process, as the selected group of participants tested the application and gave their feedback by answering the questions. Finally, at the end of the development process, an interview with senior developers from the company was conducted, in order to reason about the possibility of integrating a similar solution into their system.

When analyzed, the final outcome of this study shows overall positive results and provides valuable insights for improving the accessibility of API documentation for users with varying levels of technical expertise. The results demonstrate a visible growth in the appreciation of the platform across each iteration. The data gathered after each iteration is overall in line with the findings from the small-scale literature study, as it highlights that the introduction of higher quality descriptions and the usage of interactive live examples improves the overall impression and usability of the platform, similar to what was observed by Uddin et al. in [4].

While adding an extra layer of abstraction by having users interact with forms that they are more familiar with, provides a more intuitive experience, it is also important to still show some degree of technical details in order to prevent the platform from misrepresenting the APIs. This reasoning led to the implementation of a live example of the JSON data that is updated in real-time while the user interacts with the form; a feature that, as demonstrated by the evaluation of the third iteration and the interview with the

senior developers, represents a promising solution to the problem.

The implementation of more detailed validation and error handling is also deemed to be important by the users, as it provides valuable feedback while performing the tasks offered by the platform, which may prevent the users from misunderstanding some of the offered functionalities.

Another important aspect observed by previous research is the need to guarantee ease of navigation across the documentation, by allowing direct access to the resources sought after by the user [16]. Although the results of the evaluation show positive feedback about the navigation across the application, the scope of this project was very limited in terms of size and therefore did not provide the optimal environment to successfully evaluate the ease of navigation. For this reason, the project does not provide definitive guidelines for improving the navigation of API documentation.

In addition to that, this project was limited to the implementation of documentation for REST APIs, which implies that some of the results and findings may not be directly applicable to other types of APIs. For example, while the suggestion of using interactive live examples to add some level of abstraction may be generalized, the specific use of input forms is in most cases not extendable to other APIs.

8 Conclusion

This research aimed to identify key factors that could possibly affect the quality and effectiveness of API documentation and to optimize it specifically for non-technical users. As a result of the conducted studies and implementations, the users' feedback underlines the main aspects that contribute to the overall user experience. To be more detailed, the feedback demonstrates that factors such as description quality and error handling directly affect it.

Since the project was a collaboration with the company and they are interested in the results. It is relevant to their purposes and the industry overall. Since, this data could be used as a short clue for further development processes, i.e. what details should be paid more attention to.

One of the limitations of this study was the relatively small sample size, as only a limited number of participants were available for testing. Due to time and resource constraints, we were only able to involve a small number of testers in the evaluation process. This limited sample size may have impacted the diversity of perspectives and experiences represented in the feedback received. With a larger group of testers, we could have obtained a broader range of insights and identified potential issues that might have gone unnoticed with the limited pool of participants. Different backgrounds, skill sets, and usage scenarios could have provided a more comprehensive understanding of the API documentation's strengths and weaknesses.

8.1 Future work

As expressed in the previous sections, this study offered some good insight that could provide a good starting point for future studies toward improving the accessibility of API documentation for non-technical users. But the limitations expressed by the narrow scope of the application and the small-sized group of participants prevented the project from offering a comprehensive list of guidelines geared towards the solution of the problem. Therefore, a more extensive study is needed in the future in order to solidify and extend the findings of this project.

References

- [1] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk, “Api change and fault proneness: A threat to the success of android apps,” 08 2013. [Online]. Available: <https://doi.org/10.1145/2491411.2491428>
- [2] M. Meng, S. Steinhardt, and A. Schubert, “Application programming interface documentation: What do software developers want?” *Journal of Technical Writing and Communication*, vol. 48, no. 3, pp. 295–330, 2018. [Online]. Available: <https://doi.org/10.1177/0047281617721853>
- [3] G. Bondel, A. Cerit, and F. Matthes, “Challenges of api documentation from a provider perspective and best practices for examples in public web api documentation,” in *International Conference on Enterprise Information Systems*, 2022.
- [4] G. Uddin and M. P. Robillard, “How api documentation fails,” *IEEE Software*, vol. 32, no. 4, pp. 68–75, 2015. [Online]. Available: <https://doi.org/10.1109/MS.2014.80>
- [5] M. Meng, S. Steinhardt, and A. Schubert, “How developers use api documentation: An observation study,” *Commun. Des. Q. Rev.*, vol. 7, no. 2, p. 40–49, aug 2019. [Online]. Available: <https://doi.org/10.1145/3358931.3358937>
- [6] M. Meng, S. M. Steinhardt, and A. Schubert, “Optimizing api documentation: Some guidelines and effects,” in *Proceedings of the 38th ACM International Conference on Design of Communication*. Association for Computing Machinery, 2020. [Online]. Available: <https://doi.org/10.1145/3380851.3416759>
- [7] M. P. Robillard and R. DeLine, “A field study of api learning obstacles,” *Empirical Software Engineering*, vol. 16, pp. 703–732, 2011. [Online]. Available: <https://doi.org/10.1007/s10664-010-9150-8>
- [8] S. Clarke, “What is an End User Software Engineer?” in *End-User Software Engineering*, ser. Dagstuhl Seminar Proceedings (DagSemProc), M. H. Burnett, G. Engels, B. A. Myers, and G. Rothermel, Eds., vol. 7081. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2007, pp. 1–1. [Online]. Available: <https://drops.dagstuhl.de/opus/volltexte/2007/1080>
- [9] W. Maalej and M. P. Robillard, “Patterns of knowledge in api reference documentation,” *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1264–1282, 2013. [Online]. Available: <https://doi.org/10.1109/TSE.2013.12>

- [10] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Q.*, vol. 28, pp. 75–105, 2004. [Online]. Available: <https://doi.org/10.2307/25148625>
- [11] C. R. B. d. Souza and D. F. Redmiles, "On the roles of apis in the coordination of collaborative software development," *Computer Supported Cooperative Work (CSCW)*, Sep 2009. [Online]. Available: <https://doi.org/10.1007/s10606-009-9101-3>
- [12] K. Thayer, S. E. Chasins, and A. J. Ko, "A theory of robust api knowledge," *ACM Trans. Comput. Educ.*, vol. 21, no. 1, jan 2021. [Online]. Available: <https://doi.org/10.1145/3444945>
- [13] D. Kramer, "Api documentation from source code comments: A case study of javadoc," in *Proceedings of the 17th Annual International Conference on Computer Documentation*. Association for Computing Machinery, 1999, p. 147–153. [Online]. Available: <https://doi.org/10.1145/318372.318577>
- [14] D. Hoffman and P. Strooper, "Api documentation with executable examples," *Journal of Systems and Software*, vol. 66, no. 2, pp. 143–156, 2003. [Online]. Available: [https://doi.org/10.1016/S0164-1212\(02\)00055-9](https://doi.org/10.1016/S0164-1212(02)00055-9)
- [15] J. Bloch, "How to design a good api and why it matters," in *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications*. Association for Computing Machinery, 2006, p. 506–507. [Online]. Available: <https://doi.org/10.1145/1176617.1176622>
- [16] Q. Fan, Y. Yu, T. Wang, and H. Wang, "Why api documentation is insufficient for developers: an empirical study," *Sci China Inf Sci*, October 2020. [Online]. Available: <https://doi.org/10.1007/s11432-019-9880-8>
- [17] C. Parnin and C. Treude, "Measuring api documentation on the web," in *Proceedings of the 2nd International Workshop on Web 2.0 for Software Engineering*. Association for Computing Machinery, 2011, p. 25–30. [Online]. Available: <https://doi.org/10.1145/1984701.1984706>
- [18] T. Lethbridge, J. Singer, and A. Forward, "How software engineers use documentation: the state of the practice," *IEEE Software*, vol. 20, no. 6, pp. 35–39, 2003. [Online]. Available: <https://doi.org/10.1109/MS.2003.1241364>
- [19] G. Uddin, O. Baysal, L. Guerrouj, and F. Khomh, "Understanding how and why developers seek and analyze api-related opinions," *IEEE Transactions on*

Software Engineering, vol. 47, no. 4, pp. 694–735, 2021. [Online]. Available: <https://doi.org/10.1109/TSE.2019.2903039>

- [20] G. Uddin, F. Khomh, and C. K. Roy, “Mining api usage scenarios from stack overflow,” *Information and Software Technology*, vol. 122, p. 106277, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950584920300276>
- [21] M. Piccioni, C. A. Furia, and B. Meyer, “An empirical study of api usability,” in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 2013, pp. 5–14. [Online]. Available: <https://doi.org/10.1109/ESEM.2013.14>
- [22] A. Cummaudo, R. Vasa, and J. Grundy, “What should i document? a preliminary systematic mapping study into api documentation knowledge,” in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2019, pp. 1–6. [Online]. Available: <https://doi.org/10.1109/ESEM.2019.8870148>

A Appendix A

Here you find the full questionnaire used for the evaluation process

1. What is your level of expertise with APIs? (1-5)
2. Can you describe your background in more detail, please? (Not required)
3. How would you rate your overall usage impression? (1-5)
4. Was it easy to navigate through the playground? (1-5)
5. If you had any problems with the navigation process, can you please briefly clarify where and what you have experienced?
6. How would you rate your efficiency? (1-5)
7. How would you rate the platform's performance level? (1-5)
8. If you encounter any performance difficulties, can you describe them, please? (Not required)
9. How would you rate the information presented on the playground platform (tasks descriptions, overall information)?
10. How would you rate error handling? (1-5)
11. Did you experience any accessibility issues? (Yes/no)
12. If you encounter any accessibility issues, can you closely describe them, please?
13. How satisfied were you with the platform? (1-5)
14. Did you experience any difficulty using any particular features? (Yes/No)
15. If you encountered any difficulties with a particular feature, can you closely describe them please? What feature it was and what was the problem?
16. What additional suggestions or improvements do you have for the platform developers to make it better for users like you?