UNIVERSITY
OF OSLO

Master's thesis

Designing API documentation for novice developers

A qualitative study of documentation obstacles for novice users

Johannes Skøien

Informatics: Programming and System Architecture
60 credits

Department of Informatics
Faculty of Mathematics and Natural Sciences

Spring 2023

Designing API documentation for novice developers

A qualitative study of documentation obstacles for novice users

Johannes Skøien

Designing API documentation for novice developers

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Application Programming Interfaces (APIs) play a significant role in the software development-process, as they provide access to useful functionality and endpoints in a broad spectrum of relevant systems. Through this thesis, the needs and experiences of novice API developers are studied, in order to identify the needs of said users regarding API documentation, compared to the existing knowledge obtained by studying more experienced users. Even though there are similarities in the needs, there are also some differences that needs to be considered.

The thesis has primarily focused on two elements of the documentation: content and organizing, in order to identify areas of improvement regarding the perception for users without necessarily much preliminary experience using APIs. By looking at what information is presented, as well as how it is presented, weaknesses in current documentation can be identified and improved to better suit users with different experience levels. Content and organizing are also often connected, making them natural to view together to gain an overall understanding.

The study has been conducted as a qualitative study, where a combination of interviews and observations has been used to identify irritations, obstacles and possible improvements when developing with APIs. 15 participants with varying backgrounds have been interviewed in order to gain insight in the mentioned areas from different perspectives. In addition to this, observations of users of the DHIS2 Web API have been conducted to understand actual processes and development, with the aim to gain insight in approaches to solving current experienced problems in actual processes.

Considering facilitating learning and development for a broad range of experience can be beneficial for an API to be perceived as useful by more users. The thesis identifies six points of recommendation for practice in order for API documentation to be useful and intuitive also for novice users. These points are to 1) Consider combining a guide- and encyclopedia-approach, 2) Connect examples to use cases to explain how concepts are used in practice, 3) Make examples interactive, 4) Keep descriptions short and precise, 5) Identify and prioritize presenting the most relevant information first and 6) Gather relevant resources in categories. By employing these points of focus when designing documentation, it is possible that novice users can find the documentation easier to use, thus are more likely to succeed developing with the API.

Keywords: API, API documentation, API usability, novice developers, DHIS2, platform development

Acknowledgements

First, I wish to express my deepest gratitude to my supervisor, Jens Johan Kaasbøll, for his outstanding assistance, support and supervising through the process of creating, working with and finishing this project. Our meetings, discussions and your continuous reviews of my work have been crucial for this thesis to be successful. Your academic guidance has helped me conquer obstacles faced through the process, and helped me steer the thesis in a correct direction.

Second, I wish to thank my project-partner, Ashwin Rajeswaran, for great discussions and support through the whole process of working with this thesis. Conducting data collections, analyzing and discussing findings with you has been of great value, and has helped me a lot through the whole process.

Third, I would like to thank everyone involved in the DHIS2 Design Lab and HISP-group at UiO. Our daily discussions at the Nemko-building has helped both conquer obstacles, and keep motivation up through the whole process of working with the thesis. Being able to present parts of the thesis for you, and getting the thorough feedback you all provided has been of great value.

I would also like to thank all participants in the data collections for this thesis for taking time out of their schedules to talk to me. Without you, this thesis would never have been possible to complete. Thank you to each and every one of you for your valuable contributions to this study.

Finally, I would like to express my deepest gratitude to my girlfriend, my family, my friends and everyone else that has supported and motivated me through the last two years of studying towards the completion of this thesis. Your unconditional support and encouragement has been priceless and crucial for the completion of my work.

Johannes Skøien

University of Oslo

May 2023

Contents

List of Tables

List of Figures

1 Introduction

1.1 Problematization

Application Programming Interfaces (APIs) play a significant role in the software development-process, as they provide access to useful functionality and endpoints in a broad spectrum of relevant systems. According to the SlashData Developer Economics Survey 19th edition, an estimate of nearly 90% of all developers work with APIs (Simpson, 2022). Large software systems can be built up by several smaller sub-systems, often connected by API-connections, making APIs important links in core systems. Learning and understanding how to use APIs may therefore be a crucial part of the education of software development. Resources like documentation and training modules play a major role in the efficiency of self-exploratory learning and development. Especially in educational settings, students having to understand and learn how relevant technologies work during the limited duration of projects requires resources to be carefully considered.

Boundary resources are resources that serve to enable innovation and control development in a desired direction. (Ghazawneh and Henfridsson, 2013). Knowledge boundary resources are described by Foerderer et al. (2019) as boundary resources regarding third parties' knowledge and understanding of the development context, affecting their ability to contribute to a platform. Documentation often tends to serve as an important knowledge boundary resource for understanding and learning new technologies (Wingkvist et al., 2010). Defined both as a platform- and a social boundary resource in previous literature, documentation is artifacts aimed at helping developers accessing and utilizing platform resources (Bianco et al., 2014; Rubleske, 2020). However, the documentation of APIs can often turn out to be complex, overwhelming, and difficult to comprehend. Software developers tend to write their documentation after the development is done, making it a last step of the development process often suffering from a lack of motivation (Parnas, 2010). Badly written documentation can provide a major obstacle for novice developers and API-users, making new technologies hard to conquer and utilize, potentially strongly affecting the motivation to proceed. The consequence of this may be that it does not necessarily fulfill its potential in early stages for novice programmers, potentially leading to developers not being able to utilize API-connected systems to the scope of their potential.

Experienced API-users may have developed personal routines and preferences regarding how to approach new systems. General knowledge regarding naming conventions, what to search for etc. can make the process of gaining a quick understanding of new technologies manageable. For novice developers, however, lack of said experience can make the process significantly more difficult and time consuming. In educational contexts, students are required to study relevant documentation themselves, thus may documentation of poor quality strongly

affect the learning outcome and experience. Also in a professional development environment, poorly written documentation can strongly affect efficiency and progress. This challenge is illustrated by a participant in a previously conducted study:

> *"The biggest hurdle when learning an API is the documentation. If the documentation for an API is good, it solves 99% of your problems"*
>
> Robillard and DeLine, 2010, p. 704

Identifying what is commonly perceived as "good and bad" regarding documentation can however turn out to be difficult, as these are subjective opinions, varying from person to person (Wingkvist et al., 2010). While experienced programmers may be fairly well known with how API are typically structured, beginners may struggle with reading and understanding documentation, making the concept of APIs hard to comprehend. Thorough examination of the target group is therefore crucial in order to focus the resources on the right areas.

During IN5320 - Development in Platform Ecosystems, a master level course running autumn at the University of Oslo, the participating students are faced with the mentioned challenge of having to learn and utilize new technologies during a project with limited time. The students are creating applications for the health-platform DHIS2, using a complex API that requires time and effort to fully understand. They are required to utilize what boundary resources are available, and the quality of these can have a major impact on the learning outcome and feasibility of the project. The presentation of the complex data structure in said system is crucial for understanding and efficient use of the API for project development.

The target group of this study is novice API developers. A novice developer has previously been described as someone who is not experienced at programming (Lau and Yuen, 2009). Thus, a novice API developer in this context is someone without much experience using APIs and the belonging resources. Unlike the mentioned description of a novice developer, novice API developers are not necessarily inexperienced programmers, but rather have limited experience using APIs, with the belonging capabilities and challenges that follow.

1.2 Knowledge gap

Previous literature has covered the process of learning and exploring new APIs, mapping out weaknesses and obstacles regarding the documentation(Aghajani et al., 2020; Robillard and DeLine, 2010; Uddin and Robillard, 2015). Yet, literature offers few insights on how the process of using APIs is experienced and perceived by novice API developers. Most previous studies have targeted experienced professionals, hence getting detailed data from users working with large systems daily. Whether these results are covering also for developers with less

experience using APIs, however, is not yet mapped out sufficiently to conclude. The findings of previous studies may also be relevant for novice developers. It should however be explored more in order to see if these are covering, or if the novice target group requires other additional perspectives and considerations.

The goal of this study is to research common factors of good and bad documentation, which can contribute to the Health Information System Programme (HISP) bettering the learning process of the DHIS2 Web API for current and future students. It can also be relevant for self-educated developers using the DHIS2 Web API outside of education. Seeing how ineffective and poorly written documentation can negatively affect the development, it is beneficial to identify the factors causing the struggles, as well as how to improve these. Good practices when writing API documentation can have a great impact on API-education through organized courses, as well as also bettering the usability and readability of the documentation in general. This thesis aims to provide designers of API documentation insight in considerations that should be made when facilitating learning and development for novice API developers.

1.3 Research question

Based on the background for the thesis, as well as the previous stated purpose of the research, the following research question has been constructed:

RQ 1: *How can API-documentation be designed to facilitate learning and development for novice developers?*

To help answer this question, some more concrete, descriptive sub-question have been formulated:

RQ 1.1: *What are common factors that make existing documentation difficult to understand?*

RQ 1.2: *What are common factors that make existing documentation easy to understand?*

RQ 1.3: *What content is necessary in the documentation?*

RQ 1.4: *What content is perceived as overwhelming in documentation?*

RQ 1.5: *How should the information be organized?*

This thesis aims to contribute to the HISP-project "Developing EDucation with Input from CoordinATED research students" ("DEDICATED"), where UiO in collaboration with universities in Malawi (UNIMA), Mozambique (UEM) and Tanzania (UDSM) aims to develop subjects aimed at teaching development of public health-applications for digital platforms. Through the thesis, the aim is to identify good and bad practices with current API documentation, and thus come up with a proposed answer to the stated research questions. The result will be factors to consider regarding structure and content when writing API documentation in order to ensure usability aimed at novice developers.

To address the research question, and gain knowledge and understanding of how the novice target group perceives documentation as a knowledge boundary resource for gaining knowledge about new technologies, the study will explore the design and content of existing documentation, to look at common factors of well- and bad-formed examples. Researching good and bad factors of existing documentation, may provide insight in how to improve the usability and perception of learning about API-development, both in a general matter and in the context of platform development. The organization and content of the documentation will be investigated, and users' experiences will be explored. From there we will try to map out good practices and guidelines that can be utilized in forming of future, both new and updated API documentation. The scope for the study is related to the DHIS2 Web API, but the findings are probable to be relevant also for other APIs.

The study will be conducted as a qualitative study, where a combination of interviews and observations creates the basis for the thesis. From this, we aim to gain an understanding of the needs, as well as the obstacles novice developers may face when being introduced to new APIs and technologies. The sample group consists of a variety of participants, including lecturers and researchers, students in the course IN5320 - Development in Platform Ecosystems at UiO, developers working daily with DHIS2 as well as developers not affiliated with said system in any way.

The study aims at contributing to the field by identifying what can be improved regarding documentation in order to make API-usage in platform development more manageable for novice developers. In addition, even though this study primarily has a platform-focus, the findings can also be relevant for designers and maintainers of API documentation outside the contexts of platforms, providing relevant insight also for them. To achieve this, it is important to gain an understanding of what novice developers find challenging and lacking with current documentation of APIs as a knowledge boundary resource. Comparing findings from this target group with previous literature's findings, can provide the basis for recommendations that will cover a wider group of developers than just experienced professionals.

1.4 Thesis structure

Chapter 2 - Background gives an overview of relevant concepts to understand for the thesis.

Chapter 3 - Related research presents relevant literature on the topics *usability*, *learnability*, *learning software development* and *API documentation*.

Chapter 4 - Research method describes the methodology for the thesis, the process of data collection and analysis, and describes the sample groups of

the different iterations. It also covers a brief reflection on validity and reliability for the study.

Chapter 5 - Findings is where the findings from the data collections are presented in categories identified during analysis. The findings presented bridge into the next chapter where they are discussed related to existing literature. The chapter is structured in iterations, containing the sections Iteration 1 and Iteration 2. Iteration 1 is created as a joint effort with Ashwin Rajeswaran, thus the chapter may contain some similar paragraphs and partial sections. Iteration 2 is based on common findings from the data collections, but is written individually.

Chapter 6 - Discussion is where the findings are discussed and compared to findings from previous literature. The chapter is separated into two sub-chapters, *Content* and *Organizing*, which contains discussions regarding five categories: *From documentation to practice*, *User-group experience level*, *Continuous documentation*, *Layout and design of the documentation* and *Categorization and organization of the content*.

Chapter 7 - Conclusion addresses the sub-RQs, then addresses the main RQ of the thesis in the sub-chapter *Recommendations for practice*. It also contains a reflection on the contribution, the limitations of the thesis, as well as reflections on possible future research within this area.

2 Background

This chapter aims to provide an introductory understanding of relevant concepts for the thesis. The chapter describes different concepts and terms that are important for the thesis (HISP, DHIS2, DEDICATED, IN5320, API learning), important concepts to understand for the thesis (software platforms, API, DHIS2 Web API), as well as clarification of terms that are used together later in the thesis (Guide vs. encyclopedia, Simplicity and depth). Lastly, the chapter presents descriptions of what's considered as some good examples of existing API documentation identifier by others, as well as why these are considered to be good.

2.1 DEDICATED

DEDICATED (Developing EDucation with Input from CoordinATED research students) is a collaboration between educational institutions in Tanzania (University of Dar es Salaam), Malawi (University of Malawi) and Mozambique (Universidade Eduardo Mondlane), South Africa (University of Western Cape), Ethiopia (University of Gondar), and Norway (University of Oslo). The aim is to develop several new PhD- and master-courses that can make up new or complement existing cross-domain master programs that connect health- and IT-education. The courses target to improve knowledge in public health, community information systems, design, interoperability, and development of applications, as well as usage for data analysis etc.

2.2 HISP

Health Information Systems Programme (HISP) is a research project situated at the HISP-center in Oslo. The project is a multidisciplinary research initiative, aimed at helping low- and middle-income countries develop and deploy digital tools to monitor health, prevent and combat pandemics and manage treatments of different diseases. HISP was started in 1994, and has since then produced a large number of PhD- and master-graduates.

2.3 Software platforms

Software platforms are defined by Tiwana (2013) as "...a software-based product or service that serves as a foundation on which outside parties can build complimentary products or services." (p. 5). Platforms consist of a platform core, with one or more APIs that allow developers to connect to them and develop applications. The API creates connections to applications, creating an ecosystem of applications connected to the same core. Figure 1 displays the relation between a platform core and applications in a platform ecosystem.

Tiwana (2013) defines two types of software platforms, 1) innovation platforms and 2) transaction platforms. Innovation platforms are platforms that facilitate

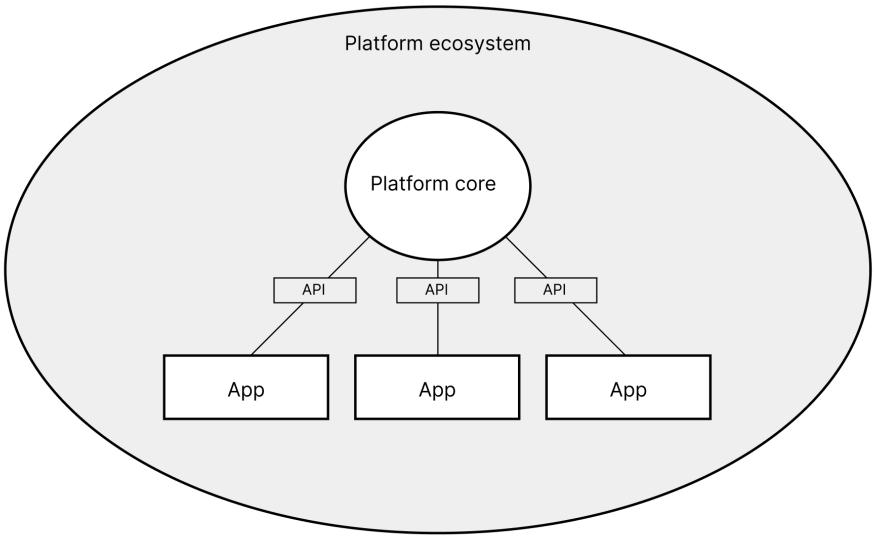


Figure 1: Platform ecosystem visualized
Several applications connected to the same platform core.

innovation, by enabling use of resources for a large group of users. Innovation platforms provide functionality that can be used to develop applications on top of the platform, extending its functionality in different ways. Examples of innovation platforms include iOS (the OS of Apple) and DHIS2. Transaction platforms facilitate interaction between actors in a market, which can be seen as the supply and demand. Transaction platforms allow for a supplier to provide something (e.g. an item or a service), and a customer to purchase said thing. Examples of transaction platforms include eBay, Uber and Amazon.

To support development within a platform ecosystem, platform owners must provide resources to third-party developers, supporting their development work, often referred to as boundary resources (Ghazawneh and Henfridsson, 2013). Boundary resources serve to enable innovation and control development in a desired direction. Ghazawneh and Henfridsson (2013) describe boundary resources as *"... the interface for the arm's-length relationship between the platform owner and the application developer."* (p. 174). A major risk of opening a system, thus exposing it to third parties as a platform, is concerning knowledge. Foerderer et al. (2019) describes what is known as knowledge boundaries: boundaries regarding third parties' knowledge and understanding of the development context, affecting their ability to contribute to the platform. It is important for platform owners to provide resources addressing these knowledge boundaries, referred to as knowledge boundary resources (Foerderer et al., 2019). Knowl-

edge boundary resources is a term that can cover all resources that helps users gain knowledge, and can therefore be a wide covering term. Some examples of knowledge boundary resources are presented in table 1.

2.4 IN5320 - Development in Platform Ecosystems

IN5320 - Development in Platform Ecosystems is a masters level-course conducted at UiO during the fall. The course covers a combination of practical work, where students learn basic frontend-development and develop applications for the DHIS2 ecosystem, as well as a theoretical part, introducing students to theory regarding platform ecosystems. For the fall of 2022, 126 students participated and completed the course.

2.5 DHIS2

The District Health Information System 2 (DHIS2) is an open-source platform, aimed at providing software for data collection, analysis, and management of health data. The platform is developed, coordinated, and operated by the HISP Centre at the University of Oslo (UiO). As of 2022, the platform is used in 73 low- and middle-income countries, and in combination with other NGO-programs this sums up to a usage in over 100 countries, covering the residency of 3.2 billion people (DHIS2, n.d.). As a public health-initiative, the platform is free to use, and offers a variety of software applications to fit the different areas' needs for data handling.

2.6 API

APIs, or Application Programming Interfaces, have been defined in previous literature as *"the interface to a reusable software entity used by multiple clients outside the developing organization, and that can be distributed separately from environment code"* (Robillard et al., 2013, p. 1). They simplify the process of connecting to and utilizing functionality in other systems infrastructure. APIs are often described in a simplified way as the waiter at a restaurant: the customer sees what's available in the menu (documentation), then places the order via the waiter. The waiter brings the order to the kitchen (core system/application), who processes the order and returns it to the customer via the waiter. This is illustrated in figure 2.

API usage is a common part of the programming process, as most large systems utilize libraries and functionality made available through APIs (Uddin and Robillard, 2015). APIs function as extensions of software, and are therefore affected by Lehman's laws of continuing change and increasing complexity (Lehman, 1980). Thus, they will by nature be required to evolve to stay competitive, by offering new functionality, correct security flaws, ease usage for developers and reduce technical debt by removing no longer used functionality (Lamothe et al., 2021). Continuous development of the documentation is there-

Knowledge boundary resource	Elaboration
Documentation	Serves as the primary knowledge boundary resource for software systems (Wingkvist et al., 2010). Offers knowledge regarding technical aspects of a software for people intending to use it. In the context of an API the documentation should present descriptions of the purpose, what functionalities are available and how to use it.
Online courses	E.g. CodeAcademy, Khan Academy etc. Online based courses that aim to teach users fundamentals and advanced features for use of a technology.
Video tutorials	Videos created with the aim to teach aspects of or full technologies through a video, where concepts are explained and can be presented in practical use cases.
Workshops	A gathering where several people in a common context are gathered to work, develop or discuss concepts to gain knowledge and a common understanding of something.
Helpdesks	A resource users can approach to get one-to-one assistance to help solve a problem.
Community resources	E.g. Stack Overflow, DHIS2 Community etc. Forums where technical questions can be asked, discussed and answered in order to gain a deeper knowledge and solve problems.

Table 1: Examples of knowledge boundary resources

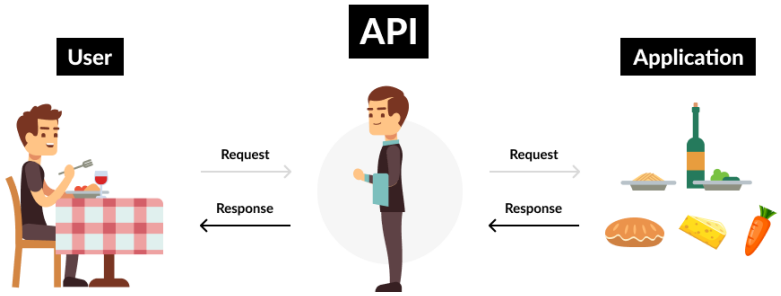


Figure 2: Visual representation of the role of APIs
Note: Figure from "A is for Application: API Basics" (Layne, 2020)

fore crucial for the service to stay usable and competitive.

By definition, APIs can be anything that connects different parts of systems to each other. E.g., when programming in an operating system, it is likely that the operating system offers some sort of functionality that allows the program being written to communicate with the OS, making the OS provide functionality internally via an API. Thus, API can be a wide covering term. However, for the purpose of this thesis, the term API is used as a description of Web APIs: APIs that are made for communicating via the web ("Web APIs", n.d.).

There are three main protocols used for Web APIs, REST, SOAP and RPC (Bigelow, 2023). REST APIs (REpresentational State Transfer) are used when handling data, and depend on a client/server relationship. The client sends a request to the server, the server handles the request and returns the data in a response. The request is not saved in states, thus each request is independent of other previous requests. REST transmits information through the HTTP-protocol, limiting it to only transferring text-information.

SOAP APIs (Standard Object Access Protocol) follow more strict rules than REST, in the sense that it allows responses only in XML-format, compared to REST being able to respond also with JSON- and HTML-objects. This allows for more predictable and secure communication, but results in a more steep learning curve, as well as more complex maintenance. SOAP also allows for communication using other protocols, e.g. TCP/IP and SMTP, as well as HTTP.

RCP APIs (Remote Procedure Call) calls for methods rather than data, to trigger actions in the server it communicates with. The response is a status of

whether the request was successful or failed, as well as error messages explaining why it failed, compared to a document with data which is returned by REST and SOAP APIs. These APIs are usually internal or private APIs, as they trigger actual changes in the server with requests, and thus potentially presents a security threat if provided to the public.

The most common protocol used in development today is REST APIs, as these requests are simple to create, resource effective and allows for response-data in different formats, fitting more experience levels than only responses in XML, which may be complex to understand.

Communication with APIs can be done in different ways. The most common is by sending URI-requests, like the one displayed in figure 3. Here, the request provides a base-URL (where the request is headed), a path from the base-URL to the location of the endpoint, as well as different parameters specifying the search. This is how developers communicate with REST APIs, as these requests are strictly required to be sent through HTTP. This is also a common way to communicate with SOAP APIs, including the required payload as parameters.

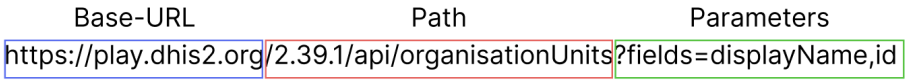


Figure 3: URI API request
Example of URI request-structure when communicating with DHIS2.

Sometimes API designers create SDKs (Software Development Kits), that can either be imported into projects, or downloaded and installed as versions of text-editors (e.g. Android Studio, a version of the text-editor IntelliJ tailored for developing applications for the Android-ecosystem). This way, developers are able to call functions instead of sending URI-requests, making it seem more like a part of the actual program, and possibly explaining the call in the function-name (e.g. getUserInfo(...)). Figure 4 displays an example of how SDKs simplify the process of communicating with APIs.

The DHIS2 Web API uses a combination of these approaches. Developing for DHIS2, custom React-hooks are available that handles core-issues like authorization etc. The requests must still be created somewhat like the first mentioned URI-approach, but are sent through the mentioned hooks, handling rudimentary things like error-handling etc., and relieving the user of having to handle complete URLs, which can become long and complex in some systems. Figure 5 displays how the DHIS2 Web API can be used in a React-project.

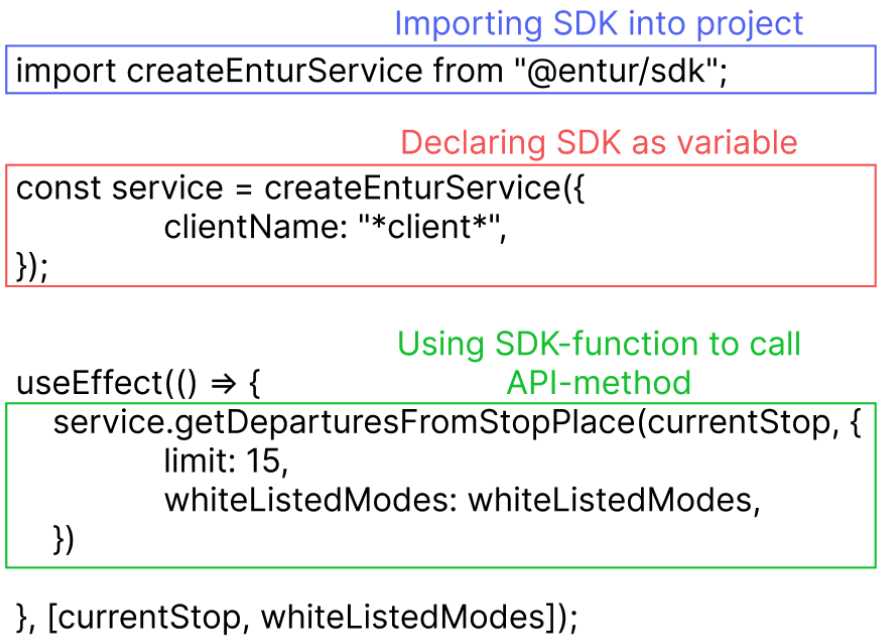


Figure 4: SDK API request
Using an SDK for communicating with the EnTur-API in React.

2.7 DHIS2 Web API

One way to communicate with DHIS2 is through the DHIS2 Web API. This API facilitates development of capacities for tracking and analyzing data, allowing for extending the functionalities of the platform regarding collection and analyzing new and existing data. The API provides access to aggregate and individually-collected data, giving developers the opportunity to develop applications using the data for analysis using indicators representing various sizes of samples.

A key part of the DHIS2 Web API is the comprehensive use of metadata to create a basis for how data is collected and analyzed. Metadata-identifiers in the DHIS2 Web API include a variety of variables, e.g. orgUnit, identifying the connected organization, dataElement, identifying the specific piece of information, period, indicating what period of time the data is representing etc. The metadata is essential for connecting values to their corresponding origin, thus creating a basis for presentation and use of the data. The metadata-identifiers allows the API to keep different endpoints needed to access limited, but do however create potential for complex compositions of different identifiers to navigate to the data needed. As illustrated in figure 6, dataValues, e.g. numbers

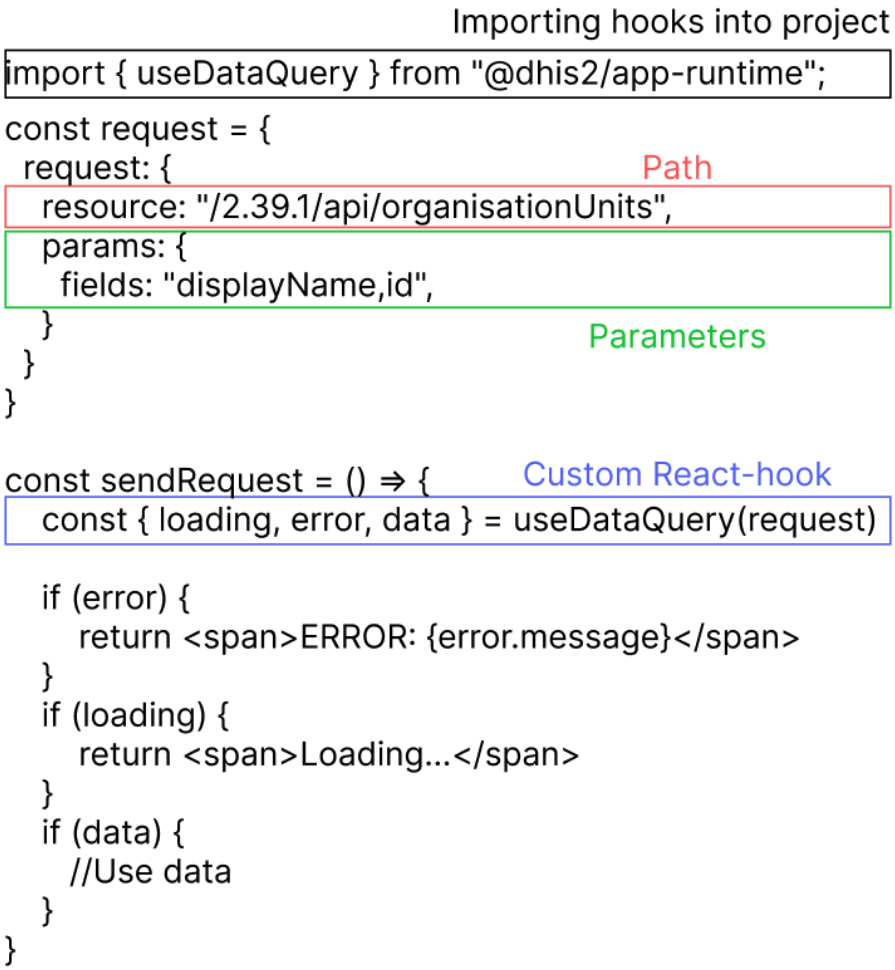


Figure 5: DHIS2 API request
Example of fetching data from the DHIS2-dataset "organizationUnits" using custom React-hooks from DHIS2.

representing recorded amount, are connected to a dataElement, an organizationUnit and a period. Each of these three metadata elements can contain one or more dataValues, e.g. an organizationUnit can contain several dataValues (e.g. dataValues representing different periods). However, a dataValue can only contain one dataElement, one organizationUnit and one period, as it represents only one registration of an occurrence (e.g. value can only be valid for one period, one orgUnit etc.).

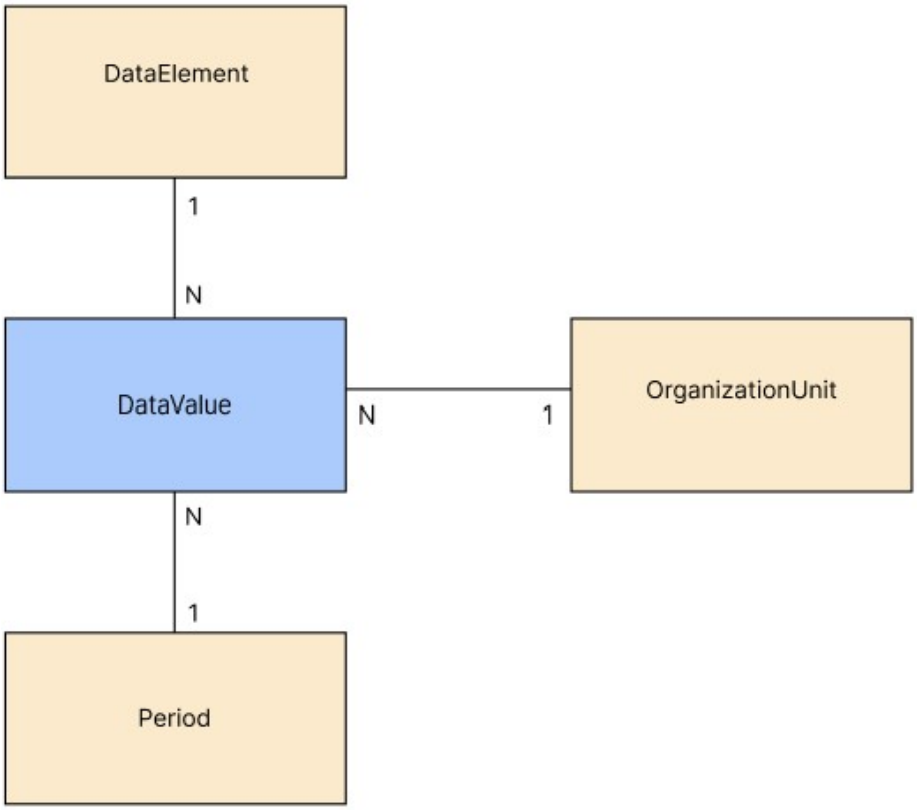


Figure 6: DHIS2 data-relations
The relation between DataElement, OrgUnit, Period and DataValue in the DHIS2 data model.

2.8 API learning

In order to learn a new technology, relevant boundary resources are essential for understanding. Different APIs have different knowledge boundary resources available, supporting and assisting the developer through their process. The primary resource for most APIs is the documentation, as this functions as the first line of support for many developers (Wingkvist et al., 2010). This resource is supposed to provide an overview of the functionalities within the system, and provide descriptions allowing developers to use them. Other typical resources for API learning can include guides, web-tutorials, communities etc.

In the case of DHIS2, the platform provides a learning resource known as "DHIS2 Academy". The resource provides courses at several levels, covering a variety of different levels, from fundamentals to more advanced use of tracker and analytics. DHIS2 Academy facilitates both online and in-person courses

that allow users from all over the world to attend and learn about DHIS2. Examples of available levels are displayed in figure 7

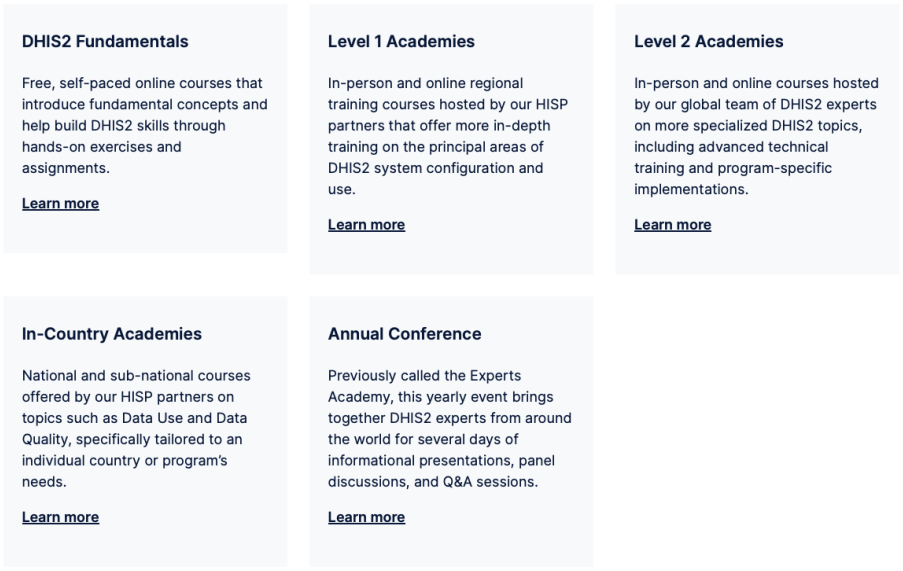


Figure 7: DHIS2 Academy
Different courses provided through DHIS2 Academy.

2.9 Guide vs. encyclopedia

Two different ways of presenting information are presenting it as a guide, and presenting it as an encyclopedia. The two approaches have their strengths and weaknesses and can serve different purposes. For example, a guide-approach can be beneficial for a beginner, as it is focused on describing a process or action step-by-step. This can be done by describing each step isolated, e.g. by having a separate page for each step, relieving it of noise created by external information not directly related to the particular action. Thus, beginners can follow each step of the interaction, being guided through the process while gaining understanding little by little. However, a guide can be slower to navigate, thus making it less effective as the user gains knowledge and understanding of important concepts, and rather wants to look up certain information quickly than to be led through all steps of a process. Opposite to a guide is the encyclopedia-approach, where information is presented in a way that can make it quick to locate e.g. by searching. This can be done in different ways, by presenting all information on one page, making it easy to search using "CTRL-F", or by gathering related information and presenting it together. However, this way of presenting information can quickly become overwhelming for beginners, as a large amount of information can make it difficult to understand where to begin searching. Thus,

even though it may be efficient for some users, others may find it difficult to use and create more confusion rather than help. Oftentimes, encyclopedia-style presentations have more focus on presenting information than describing the approach, thus potentially requiring some degree of initiatory knowledge of typical steps for different actions. Possible presentations of both approaches are presented in figure 8.

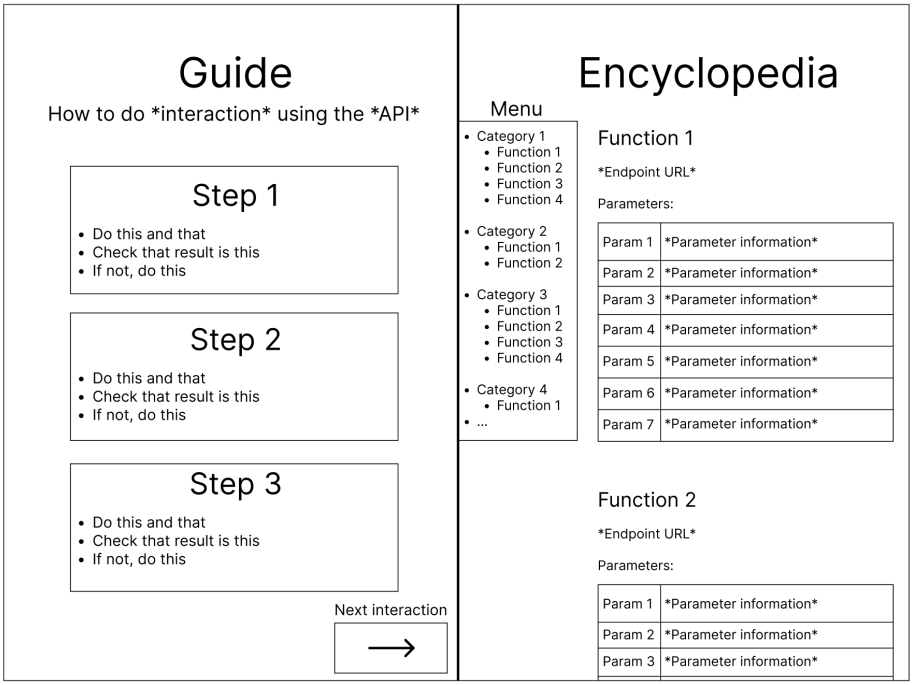


Figure 8: Guide vs. encyclopedia
Example displaying the difference between a possible guide-layout and a possible encyclopedia-style documentation.

2.10 Simplicity and depth

Two important terms that will be used during the thesis are the terms "simplicity" and "depth". In the context of this thesis, the term "depth" refers to information about something that is presented thoroughly in detail. For example, depth in the description of a car could describe each part of the engine, chassis, interior etc. in detail, describing how it works as well as why it works this way, giving the reader a thorough and complete understanding of each component after reading. By providing detailed information about every little detail, the user may gain a thorough understanding of what is being described as a whole. By putting all of this information together, this approach allows for

an extremely detailed understanding of the product as a whole. However, the effort required to obtain this detailed understanding can be high, and require some previous knowledge of what is being presented.

On the other hand, another term that will be used is "simplicity". Even though it is used as a contrast to depth in this thesis, simplicity does not necessarily mean descriptions not touching upon detailed information at all. Instead, what is meant by simplicity in the context of this thesis is that something is described in a simple way, by avoiding information that is not strictly necessary to understand something. Thus, related to the formulation in RQ1.3, simplicity would imply that the presented content is necessary for understanding, making the two terms closely connected. However, this does not necessarily apply the other way around, as something being necessary does not have to mean that it is presented simply.

In the example mentioned above, a description of a car presented with focus on simplicity could instead of describing every component and bolt of the car, rather focus on the essential parts for understanding what a car is and how it works. Instead of going into detail regarding spark plugs in the engine, their role and how they work together with other components to create energy in the engine, a simplicity approach could abstract this away and leave the reader with the essentials. A car consists of a chassis, four wheels and an engine to create power. With this knowledge, the user is able to understand the basics of something, and can then proceed further into more detailed information as the needs grow.

2.11 Examples of good API-documentation

Bush (2019) presents a number of APIs which have what he perceives as good examples of documentation, listing the characteristics of what makes them good. He also presented lessons to take from each case. Especially three mentioned examples were interesting:

2.11.1 Twitter

The API documentation for Twitter was described as a good example of how information should be presented, as it was described as feeling more like an interactive guide instead of a blog post, making it easier and more engaging for the user to follow than traditional documentation. Each page was described as having a "Next step"-indicator, connecting relevant parts of the documentation together in the correct order of use. It was also described how almost every page has an FAQ, allowing the documentation to remain simple, while more detailed information can be found in the FAQ. This way, the documentation is kept short and understandable for the users. The lesson mentioned for this documentation was to "Be flexible with how you present information", referring to the value of keeping information simple with the possibility of presenting more detailed information in a connected FAQ.

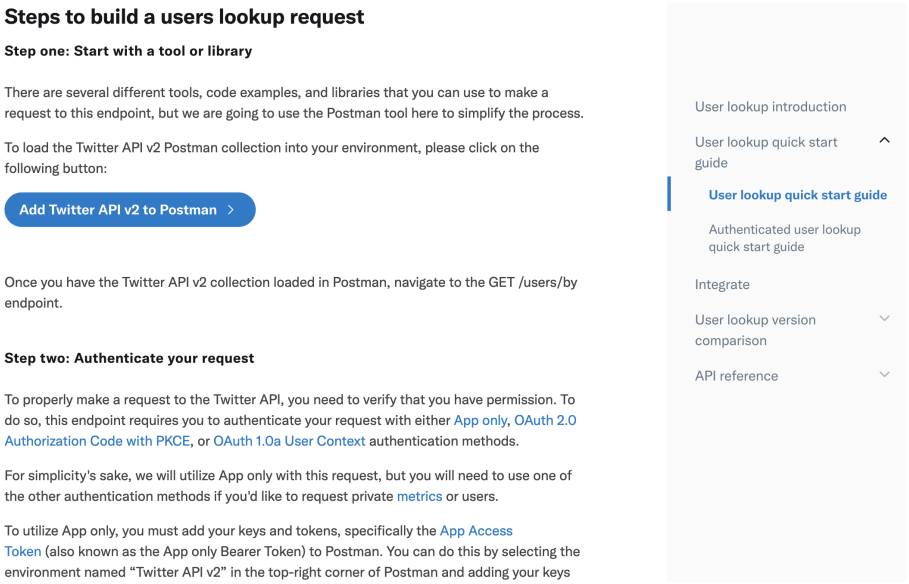


Figure 9: Twitter API Documentation
Displaying how an endpoint is accessed interactively.

2.11.2 Dropbox

Dropbox was mentioned as a good example of tailoring documentation to the different programming backgrounds, by allowing the user to choose the preferred programming language from a list, thereafter presenting examples in this language. This way, it is easier for users to understand the examples and put them to use in their own project. The documentation was also mentioned as a good example of simplicity in the descriptions and layout, making the documentation easy to understand. The lesson mentioned from this documentation was to "cater to unique dev-backgrounds", referring to the ability to choose what programming language the developer wants to get the examples presented in.

2.11.3 GitHub

The API documentation for the GitHub-API was mentioned as a good example of how to keep things simple, by avoiding having too much information in one place. As with the other examples, this was mentioned as a way for developers to easier comprehend information and avoid confusion. The documentation provides links to relevant reference-material, guides and libraries in the top menu, allowing developers to easily and quickly navigate to relevant resources without it being a part of the current documentation, avoiding overflow of information. Documentation is written in a blog-post style, but awareness of simplicity and avoiding presenting too much information still keeps the documentation clear

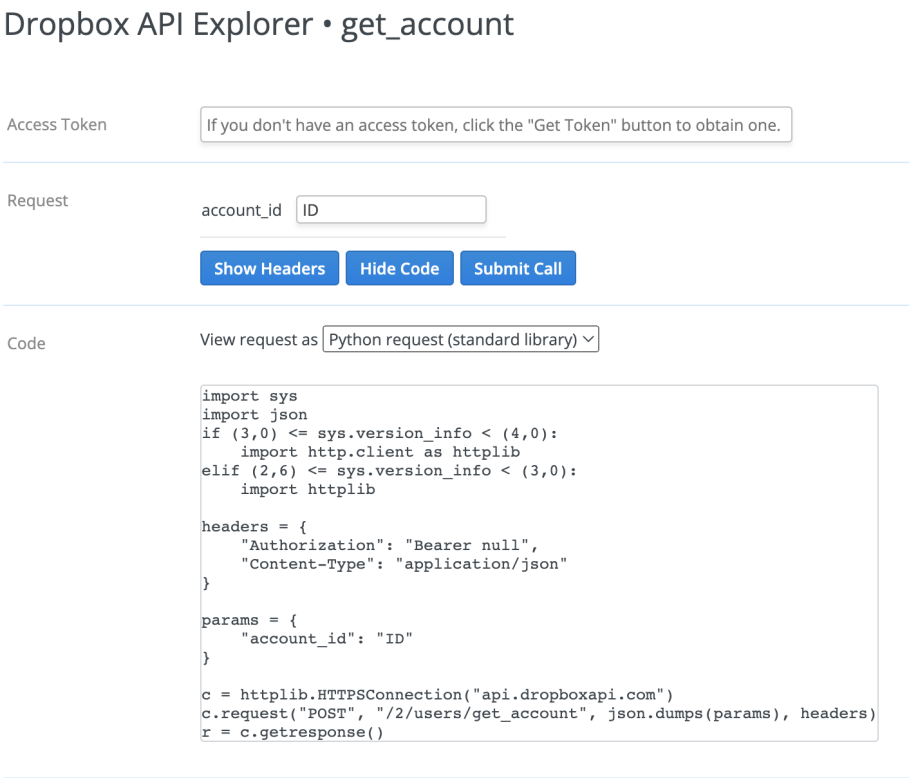


Figure 10: Dropbox API Documentation
Displaying an example of endpoint presentation in the API Explorer.

despite the presentation structure. They were also praised for having a widget in every page of the documentation that allows the user to see the status of the API, allowing them to quickly identify whether problems that appear are caused by programming errors or system-failure. The mentioned lesson from this example was to "save developer time wherever you can", referring to the status-widget saving developers time in debugging when they face obstacles, as well as linking to relevant resources.

The mentioned examples have some different characteristics and functions that make them good. However, some things were found in all three examples, and can be seen as a common denominator for documentation to be perceived as good. First, they all have an authentication- and quickstart-guide, where gaining access and starting to use the documentation is described step-by-step to allow users to quickly begin using the API. Second, good endpoint definitions and descriptions were mentioned as good characteristics of all three documentations, and lastly, that they all provide code-examples with example responses were mentioned as important for them to be perceived as good examples of API

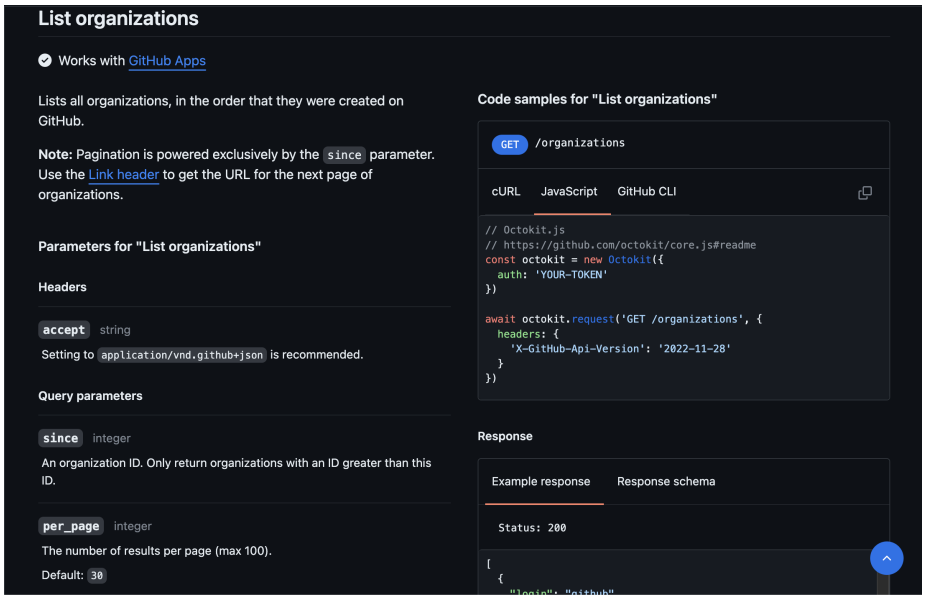


Figure 11: GitHub API Documentation
Displaying an example of endpoint presentations.

documentation, allowing the users to see how the API can be used in practice.

3 Related literature

This chapter presents literature that is relevant, and that is used for discussion of findings later in the thesis. First, literature related to usability is presented, followed by literature touching upon learnability. Further, literature discussing the process of learning software development is presented, before presenting relevant literature regarding previous research on API documentation and related topics.

3.1 Usability

An important factor for a system to be perceived as useful, is that it provides usability for the user. Usability is defined in a number of ways. Shackel (1981) is said to have created the first formal definition of usability, by defining it as *"... the capability to be used by humans easily and effectively"* (p. 24). Bevan et al. (1991) argues that it should be defined as *"... the ease of use and acceptability of a product for a particular class of users carrying out specific tasks in a specific environment"* (p. 1). How to measure usability may turn out to be a challenging task, as it has to consider whether to focus on subjective or objective measurements. Bevan et al. (1991) presents four perspectives on measuring usability: 1) The product-oriented view, measuring the ergonomics of the product, 2) the user-oriented view, measuring the attitude and effort of the user, 3) the user performance view, measuring the task performance and interaction of the user, especially considering ease of use and acceptability and 4) the context-oriented view, considering the user group, tasks and environment.

Hornbæk (2006) highlights the challenges regarding measuring usability of a system, including the trade-off between objective and subjective measurements. It is important to separate the term usability from usableness (Gluck, 1997), as measuring usableness can refer to reviewing functions regarding whether something is possible to use, while usability measures the effect of using the said thing. For something like documentation, usability measurement would consider how the information is perceived, understood and used by the user, while usableness would simply measure whether the documentation is possible to use or not.

Several factors can affect the usability of an artifact. Malhotra (1982) and Moy et al. (2018) discusses the effects of large amounts of text and information on one page in learning. They find that as the amount of information increases, the cognitive ability to follow decreases. Large amounts of information also affects the ability to distinguish between what's relevant and what's not negatively. Moran (2020) finds that people tend to scan pages rather than read everything, suggesting that information should be presented in a way that allows for understanding without necessarily reading everything thoroughly. This can be achieved by separating content into sections with explaining headlines allowing the reader to quickly locate relevant information for their use. This is further emphasized by Pernice (2019) who presents four typical reading patterns that

are typically used by people when reading information, the layer-cake-pattern, the commitment pattern, the F-pattern and the spotted pattern. Figure 12 displays visualizations of the different patterns.

Figure 12: Reading patterns
The four different reading patterns, from top left: the layer-cake-pattern, the commitment pattern, the F-pattern and the spotted pattern. Note: Figure created with images from "Text Scanning Patterns: Eyetracking Evidence" (Pernice, 2019)

Consistency and standardization is important for the user to be able to understand connections between different components and parts of a page. Lidwell et al. (2003), Preece et al. (2019) and Schlatter and Levinson (2013) point out the importance of consistency as a design principle, underlining its value for creating a successful solution for user understanding and intuition. However, Preece et al. (2019) also mention a risk that can occur when trying to achieve consistency: when trying to design an interface to be consistent with something,

one faces the risk of making it inconsistent with something else, emphasizing that it is not necessarily as easy as it may seem to achieve.

Furnas et al. (1987) discusses what they call "The terminology problem", covering the issue regarding how the choice of words can affect the availability and accessibility of information. They find that *"Many, many alternative access words are needed for users to get what they want from large and complex systems."*(Furnas et al., 1987, p. 971), emphasizing that finding one word that provides access for all users is difficult. Their findings show that a single word (access term) created by one designer only has a hit rate of only 10-20 percent, underlining the difficulty of selecting accessible words, as well as the importance of defining alternative terms.

Some web-resources, e.g. Wikipedia, utilize community contribution for the users to be able to contribute to the content with additional knowledge they possess. This way everyone is able to contribute, thus possibly broadening the knowledge and providing increased quality content to all users and providing more words for each technical term, thus increasing the chances of a hit when searching. Xu and Li (2015) discusses two types of motivation for contributing to a public source of information: content contribution and community participation. The first mentioned motivation covers how providing extra/extended content may be motivated by credit or a feeling of recognition (e.g. Wikipedia displaying the top recent contributors), or by the wish for self-development. Community participation however, may be motivated by more intrinsic factors like a feeling of belonging or ideological factors like the ideology of free knowledge.

3.2 Learnability

An important part of a functioning API documentation is the learnability: the ability to easily learn and understand the content. Nielsen (1994) argues that learnability is a fundamental attribute in usability. Learnability is especially important for novice users (Dzida et al., 1978).

There is a consensus that learnability exists as an aspect of usability (Grossman et al., 2009). However, it is important to understand that learnability is not necessarily the same as usability, as the two terms may address different issues. Learnability has been defined in several different ways, emphasizing different areas of the learning process. For example, Nielsen (1994) refers to learnability as the experience of the initial learning curve for novice users, while Shneiderman defines it as the time it takes *"... typical members of the user community to learn how to use the actions relevant to a set of tasks"* (Shneiderman, 1997, p.16). Mifsud (2011) discusses the difference and relationship between usability and learnability, shedding light on the common wrong use of the two terms interchangeably. If we rephrase learnability to "ease of learning", and usability to "ease of use", it becomes more clear that learnability primarily addresses the initial process of learning, while usability refers to the easiness of repetitive use,

separating the two terms and the issues they aim to address.

3.3 Learning software development

In the case of learning software development, studies have shown positive results when working in pairs of two when performing tasks, so called pair programming. This approach can result in better understanding and knowledge transfer than solo programming (Vanhanen and Lassenius, 2005), possibly beneficial for novice developers. Skill level serves as an important factor for the effectiveness of programming (Salleh et al., 2011), with the less experienced part of the developer community having the best effect and most satisfaction of performing pair programming (Arisholm et al., 2007; Hannay et al., 2009). Figure 14 displays the effect of pair programming for users of different experience levels, and shows how pair programming strongly increases the correctness for juniors, while also decreasing time used for intermediates.

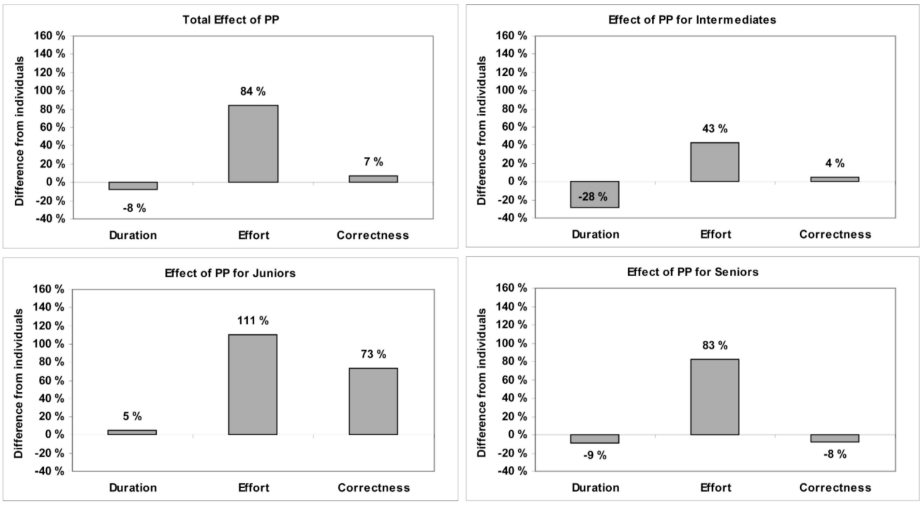


Figure 13: Effect of pair programming for different experience levels
Note: Figure from Arisholm et al.(2007, p.74)

Trial and error can also be an efficient way of learning and exploring. Jones et al. (2010) discusses the benefits of trial and error as an approach to problem solving, and found it to be a significantly better approach than the compared approach of errorless conditions. When facing complex concepts and structures, textual descriptions and explanations can be difficult to comprehend, thus affecting the overall understanding of a system. Raiyn (2016) discusses the effectiveness of visual elements in learning and their effect on learnability. He argues that use of visual elements can have positive effects on students' understanding and learning, thus using it can improve the learning outcome of students in various fields.

3.4 API documentation

Documentation is an important knowledge boundary resource when learning and using APIs, as they tend to be the first line of support when facing obstacles (Wingkvist et al., 2010). However, documentation is often written as the last step of a development process, after the software has been written, rather than parallel with development. Thus, the documentation often suffers from a lack of motivation from the developers when it comes to creating it (Parnas, 2010). Poor documentation is the primary reason for system quality degradation (Kajko-Mattsson, 2005), thus may the lack of motivation writing documentation lead to significant consequences for the system as a whole in the long term.

Several articles have explored API documentation in an attempt to identify factors that make documentation good, and what is lacking to improve it. Through "How API Documentation Fails", Uddin and Robillard (2015) explores struggles experienced by developers and other employees at IBM, first in Ottawa and Toronto, then Canada and Great Britain. They tried to gain an impression and overview of the different issues in API documentation today, and how these are perceived by developers in said company. Results from the study are summed up in figure 14.

Robillard and DeLine (2010) explores learning obstacles when facing new APIs for employees at Microsoft. Their study was conducted in three stages: (1) a survey to map out the most common obstacles to create the basis for further research, (2) qualitative interviews to gather detailed information on issues and perception of these, and (3) a second survey to confirm findings. The study highlighted five important factors to consider while creating and designing API documentation, to improve efficiency:

1. **Documentation of intent:** The lack of intent can make developers uncertain of the provider's purpose of the API. This can have a great impact on the system performance, and lead to developers using the API in ways it is not intended. The intent must however involve a tradeoff, as overflow of information can bloat the documentation. Not all participants consider intent to be of significant help either. It is therefore important to evaluate the cost/benefit, as well as the scope of the descriptions when deciding whether to prioritize this.

2. **Code examples:** Code examples are important for efficient API learning. They show the learner concrete utilizations of the different functionalities and possibilities, and can help make the API tangible, hence easier to understand for the developer. There is, however, a balance that must be considered regarding complexity of the examples displayed. If the examples are too comprehensive, they can be easy to copy but difficult to understand. If they are too simple, respondents indicate that they lose their value. Finding a universal agreement on what is the best trade-off between these can be difficult to achieve, as perception of information

API documentation problems reported in the exploratory survey.

Category	Problem	Description	E*	D*
Content	Incompleteness	The description of an API element or topic wasn't where it was expected to be.	20	20
	Ambiguity	The description of an API element was mostly complete but unclear.	16	15
	Unexplained examples	A code example was insufficiently explained.	10	8
	Obsoleteness	The documentation on a topic referred to a previous version of the API.	6	6
	Inconsistency	The documentation of elements meant to be combined didn't agree.	5	4
	Incorrectness	Some information was incorrect.	4	4
		Total	**61**	**57**
Presentation	Bloat	The description of an API element or topic was verbose or excessively extensive.	12	11
	Fragmentation	The information related to an element or topic was fragmented or scattered over too many pages or sections.	5	5
	Excess structural information	The description of an element contained redundant information about the element's syntax or structure, which could be easily obtained through modern IDEs.	4	3
	Tangled information	The description of an API element or topic was tangled with information the respondent didn't need.	4	3
		Total	**25**	**22**

Figure 14: Findings from "How API Documentation Fails"
Note: Figure from Uddin and Robillard (2015, p. 70). E* = number of examples that mentioned a problem, D* = number of developers who reported a problem

amount is highly subjective. A possible solution to this, presented in the study, is to use simple examples in combination, to exemplify certain actions and combinations that can be used to reach a goal.

3. **Matching APIs with scenarios:** As a way of concretizing the different functionalities, the point of matching scenarios with the function pattern that leads to the relevant endpoint is mentioned as one of the points. This can be connected to the proposed solution from 2., where an action is presented using the pattern of functions that is used to reach that goal.

4. **Penetrability:** Penetrability is described as the fine line that the developer needs to balance on, between an over-exposure of the APIs internal elements (which violates the principle of hiding information), and a design that makes the behavior of the API impenetrable (making it hard to learn). Poorly designed error handling and lack of insight makes working with the API a difficult task. A common work-around of this, is for developers to either inspect the source code, or perform micro-experimentation with the API, both being complex and time-consuming tasks.

5. **Format and presentation:** The last factor that is presented is the

> format of the documentation. A modularized, separated presentation, displaying the different functions on separate pages, is more time consuming than having all information gathered on one page. The respondents also experienced frustration when reading documentation that merely rehashed the name of the methods, bloated the presentation with derived information or provided overly trivial examples that only showed a single method call. The presentation of the documentation thus plays a major role in the perception of the developers.

Even though the two previously mentioned articles covers relevant topics, like common irritations and struggles when it comes to reading and using API documentation in general (Uddin and Robillard, 2015) and obstacles when learning new APIs (Robillard and DeLine, 2010), their target group is what separates them from our study. Their participants have respectively 13 and 9.8 years of experience, making them outside the scope of this study, targeting novice developers. They do however present findings that may be relevant also for the target group of this study.

Garousi et al. (2013) finds that there is a difference in needs regarding the documentation, depending on the experience level, as well as difference in use of documentation. They state that experienced users tend to refer less to documentation than novice users, indicating that the needs of the novices should be prioritized when creating documentation in order for it to be successful. This study is conducted as a hybrid, action research study, where they combine mining of project data from an internal system for analysis with results from a questionnaire-based survey eliciting expert's opinions to compare what's being said and what's actually being done regarding documentation. The survey was sent to a group of 135 software engineers, where 25 responded (18.5%). They do compare documentation-use and perception based on experience level, but do not inform about the respondents' distribution of years of experience. However, since the study is conducted in a professional industry-setting at a large, Canada-based company, it is likely that the distribution of experience between the respondents is comparable to other previously mentioned studies of documentation in similar settings. In said cases the mean years of experience has been 9-13 years, thus separating it from the target group of our study.

Wyner and Lubin (2011) presents a study exploring how MBA-students with varying technical backgrounds perceive learning coding and using a simple API during a 1-week, intensive lab-seminar. They are first given three days of tutoring covering basic programming, before expanding the knowledge by utilizing the skills using a simple restaurant-API. The paper does not touch a lot upon the documentation of APIs specifically, but describes how a tailored API and documentation simplifies the process of learning. The documentation is described as short and readable, tuned to the students' experience, and updated every time the API had changes made. It implicitly underlines the importance of a documentation made with the experience of users in mind, for them to successfully utilize its functionality during a short span of time. It does also underline the

importance and usability of the APIs role in software development. Seeing how the general response to the course was positive, it is clear how important and useful API knowledge is from early stages of learning programming.

Through surveying 146 participants, Aghajani et al. (2020) identify issues from the practitioners perspective that should be considered when creating documentation. They identify a series of different issues categorized as What (what content is found in the documentation, e.g. inappropriate installation instructions and outdated version information), How (how information is presented, e.g. regarding readability, usability and format/presentation) and Process/Tool-related (covering the issues of tools and processes when creating documentation, e.g. lack of time to write documentation). Like the previously mentioned articles regarding API documentation, their respondents leaned towards the higher end of the experience curve, with 60% of the participants having >10 years of experience, and 80% having 5+ years of experience. Thus, the findings may be affected by their level of experience, making the results not necessarily representative for the target group of this study. However, as with the other articles, the findings may still be relevant to some degree, making them interesting to compare to findings from this study.

McLellan et al. (1998) highlights the importance of examples for understanding APIs. Their study emphasizes the important role of examples for developers to be able to use the different functions and endpoints of an API by presenting what's possible to do and what's not. As mentioned by a participant, *"... if there are examples of it, they're worth their weight in gold."* (McLellan et al., 1998, p.80). Zinovieva et al. (2021) discusses the use of "Online coding platforms" for remote learning, describing how they can contribute to learning by presenting tasks and an environment where the user can try to solve problems without having to implement the code themselves. Combining examples and development environments in a sandbox (Arntzen et al., 2019) can be beneficial for allowing users to explore the possibilities of the API.

4 Research method

This chapter will describe the process of collecting and analyzing data for this thesis. First, the methodology that has been used is presented. Then, the method used for data collection is described, as well as why this was chosen instead of alternative approaches. Further, the sample group for the study is presented, as well as a description of why they were asked to participate. After this, the validity and reliability of the study is briefly discussed, before the process of collecting data through literature review and the two conducted iterations are described. Lastly, the process of analyzing the data is described, before presenting the findings from the rounds of open and axial coding.

The study has been conducted as a joint effort with Ashwin Rajeswaran. Originally, we started off with a similar task and somewhat similar RQs. We have therefore conducted all data collection efforts together. Analysis has also been carried out together, allowing us to discuss findings and categories as we worked our way through the process, identifying relevant categories and topics. The focus of our theses has since been separated, splitting our work into two separate contributions to the field. As a result, the perspective mentioned through this chapter is from a group perspective, hence why decision makers are mentioned as "we".

4.1 Methodology: Qualitative study

This study has been conducted as a qualitative study. According to Creswell (2006), qualitative studies are suitable for exploring issues where there is a need for studying people and identify variables to be considered. It allows for understanding and knowledge-gathering in situations where thorough and complex understanding of the issue is needed. Alternatively, the study could have been conducted as a case study, but seeing how we did not have an identified case that we wanted to focus on and research specifically, a qualitative study allowed us to study more broadly and discover the relevant perspectives during the process.

4.2 Method

4.2.1 Qualitative data collection

The study is largely based on interviews, combined with some observations of the system in use. This way, we were able to gain qualitative insight in the issue, providing detailed information about obstacles and suggested improvements. Previous studies, e.g. Uddin and Robillard (2015) have conducted quantitative studies by sending out questionnaires to a large number of people. Even though this approach could have given us a larger amount of responses than we collected, we wanted a purely qualitative approach to gain detailed insight from a smaller selection rather than general information from more respondents. An issue with quantitative methods like surveys is the reduced possibility to ask

follow-up questions, thus possibly affecting the chances of gaining detailed information.

A possible alternative approach could be to base the study to a larger degree on observations. This could have led to detailed insight in actual development processes. However, seeing how documentation is a knowledge boundary resource, aimed at supporting developers, it is not necessarily used in every development session, and the amount of use can also be very limited, depending on the tasks. Hence, in a worst case scenario, a period dedicated to only observing in a natural context (e.g. at the developers place of work) could result in few to no findings, making it a method that requires sufficient time to provide detailed information. It could be possible to conduct observations in a controlled environment, but as the aim is to explore how API documentation is perceived in actual development settings, this could have led to findings being less representative relative to actual cases. Surveys, as mentioned, could also have been used as a method. However, with limited time to conduct the data collections, and our wish to collect qualitative data to gain detailed insight, this thought was abandoned. We found survey to be a less efficient way of collecting data for our study, as too broad questions could be difficult for participants to answer (e.g. "How do you find API documentation"), and specific questions may face the risk of being perceived as leading (e.g. "What do you like about *system*"). Also, finding a sufficiently large number of participants fitting our target group, both providing a breadth in participants and still ensuring that they fit into the experience-level that is studied, could have been more difficult than the benefits following this method.

4.2.2 Sample group

The sample group consisted of a combination of lecturers, students, DHIS2 developers and developers without experience using DHIS2. The participants for the different iterations of interviews and why they were asked to participate are presented in table 2. Table 3 presents who were observed in the study. We wanted to gain insight in API development from different perspectives, thus collecting data from a broad range of experiences seemed beneficial. Given that previous research has already focused on experienced professionals, we rather wanted to focus primarily on the less experienced user group, as well as roles supporting this group's development.

The participants from the academic part of the sample group for iteration 1 were asked to participate because they either lecture or have experience lecturing in courses using different APIs for higher education. Thus, the target was to gain insight in development processes for novice developers, which students often are, from an outside perspective. As our supervisor, Jens Johan Kaasbøll, has broad experience within the educational-/research-field, he reached out to some possible contacts lecturing in courses that seemed fitting for the thesis by email, forwarding further contact to Aswhin and me. The participants were briefly presented with our case, and why they were contacted. Five possible partici-

pants were contacted, and four responded that their curriculum and experiences could be relevant and that they were positive to participate.

The non-academic were chosen to gain some initial insight from the perspective of users, instead of only approaching further data collections with expert insight. The participants were approached in person, as they were found around locations where we usually either work or study. They were briefly presented the purpose of the interview, why we wanted to speak with them, and then planned a time for an interview. All three approached wanted to participate in the study.

For iteration 2, Ashwin and I conducted data collections individually, with an interview guide that had been made together beforehand. This way, we were able to conduct data collections in different contexts, collecting data that was discussed and compared when we met up after the iteration. By preparing a common interview-guide, we were able to ensure that participants were asked the same base-questions regardless of who and where they were, ensuring that the data could be used by both of us, and possibly strengthening the reliability of the study. The participants within the DHIS2 developer-group were working in a workshop, where Ashwin and the purpose of our study was briefly introduced. The participants were invited to come over for a talk after their session of work was finished. Notes were taken during their process of work in the workshop, regarding how they communicated and approached problem solving in teams. Four participants approached Ashwin after the workshop and participated in separate interviews where they were able to elaborate thoughts about development within DHIS2, as well as the documentation as a tool.

In the same time period, the students were messaged during the last process of their project work in IN5320. These were students that I had assisted through the course, and I therefore had some preliminary insight in their process so far in their project. They were explained the purpose for the interview, and asked if they were interested in participating. Two groups of students were approached, and members of both groups participated in the data collections (1x 1 participant, 1x 2 participants). In addition to this, a participant working with the DHIS2 Core was approached, and participated in an interview, which allowed us to gain insight in the issue from the perspective of a representative for the platform core.

4.3 Validity and reliability

4.3.1 Validity

Validity can be important to assess in order to consider whether the results are credible and evaluate the quality of the study. Stølen (2023) defines validity by stating that *"An evaluation is valid if it evaluates what it is meant to evaluate"* (p.115). He continues to separate validity into four sub-concepts: external validity, internal validity, construct validity and conclusion validity.

Iteration 1		
Group	Who?	Why did they participate?
Academics	4 participants from different universities and colleges, lecturing in and/or researching relevant topics for the thesis	To gain initial insight in API-development for novice developers, from the perspective of the lecturing part
Non-academics	3 participants with varying development-background - both with and without experience developing using the DHIS2 Web API	To gain insight in the use of APIs from the developers perspective, both directly connected to DHIS2 and APIs in general
Iteration 2		
Group	Who?	Why did they participate?
DHIS2 Developers	5 developers working daily with development within the DHIS2 ecosystem, with varying years of experience	To gain insight in the perception of the API and documentation from the perspective of users working with the platform daily, and their perception of the onboarding-/ learning-process
Students at UiO	3 students at UiO, participating in the course IN5320 - "Development in Platform Ecosystems", where they develop applications using the DHIS2 Web API	To gain insight in the perception of the API and documentation from the perspective of users dependent on the API to create applications in an educational setting, during a restricted period of time

Table 2: Participants in the interviews of the study

Observations		
Group	Who?	Why did they participate?
Students participating in the course IN5320 - Development in Platform Ecosystems	126 students at UiO on master's level, with varying background regarding development	To gain insight in the struggles of starting to use the DHIS2 Web API for students without necessarily much experience using APIs in an actual development setting, and to lay the foundation for following interviews
DHIS2 Developers	12 DHIS2-developers participating in a workshop	To gain insight in the perception of DHIS2 and its available resources from developers working with the system daily.

Table 3: Participants in the observations of the study

External validity covers whether the study is generalizable and relevant also outside of the actual case. As qualitative studies aim to research specific issues for a certain population (Leung, 2015), naturally their focus is to gather detailed information rather than quantifiable data that can be considered generalizable. Thus, generalizability can be difficult to achieve using qualitative methods, as answers and opinions can vary significantly between participants. Semi-structured interviews, the method used for the data collections for this study, are subject to varying answers, thus resulting in difficulties attempting to generalize. Naturally, as the study focuses specifically on DHIS2 as a platform with its belonging Web API, its characteristics and peculiarities can form the results, separating them from being generalizable. Seeing how DHIS2 provides an API that is heavily dependent on a complex combination of metadata, the documentation can quickly become difficult to grasp for novices without proper thought regarding presentation aimed at them, thus providing different results than other public APIs with simpler logic.

We tried to research a diverse sample group, with varying backgrounds, levels of experience and roles, in an attempt to achieve an understanding of the target groups' experiences. By exploring perspectives of participants also outside of DHIS2, without experience with said system, the aim was to gain some general insight in addition to the specific DHIS2-focus. However, the results are still difficult to generalize, as our sample group is still of limited size. This, combined with qualitative data collection methods, which provide varying results as a result of subjective opinions and experiences, the results are not necessarily possible to safely generalize. By combining interviews with other methods like

observations and literature review, thus triangulating, it is possible to approach some sort of generalization, but a larger sample group and further triangulation could be necessary to argue that the results are generalizable for all users.

Internal validity covers the concern of causal relations between variables. In the case of this study, a discussion regarding internal validity would cover the connection between understanding and usability of an API, and the documentation. Even though the connection between these factors are difficult to measure, both previous studies and responses in this study presents documentation as an important and highly relevant resource for understanding and using APIs. Therefore, it is probable that there is a connection between users' understanding of APIs and the documentation as a resource. In many cases, the documentation is the primary, sometimes only resource API creators provide as a resource for developers being able to use and understand their system. However, other factors may play in and affect the ability to understand usage of APIs. Programming experience can play an important role for the ability to understand, making it difficult for users to utilize the API. It is however likely that, by providing a well thought out documentation targeted towards a specific user group, that the users will be able to understand something that can make them able to use the system.

Construct validity covers whether the operational measures represent the researcher's thoughts and investigation according to the research question. The research question aims to gain a broad understanding of how API documentation can be designed to improve learning and development for novice developers. The sub-questions aim to support the main RQ, by providing insight into factors that make existing documentation difficult and easy to understand, what content is necessary and what is perceived as excess, as well as thoughts on how information should be organized to be perceived as logical and facilitate learning and understanding. As mentioned, the method was chosen to be able to collect information by allowing the participants to speak freely, within the borders set by the interview guide. The questions were formed in a way that allowed us to gain insight in the experiences of the participants, and provide an understanding that would allow us to answer the sub-RQs. The formulations of the sub-RQs were chosen as contrasts, to gain insight into both what is good and bad, thus identifying what is perceived as things that should be included and not. Therefore, the questions were constructed in a way that was in harmony with the thought purpose of the study. As mentioned in the background-chapter, the terms "simplicity" and "necessary" from RQ1.3 are closely connected. Having simplicity implies that the content presented is necessary. However, necessary information is not always presented with focus on simplicity, making the terms connected in one way but separated in another.

Lastly, conclusion validity covers the connection between findings and the conclusion. In this thesis, the conclusion is presented as a sum-up of the discussion, where findings are discussed up against existing literature. Thus, it can be justified to indicate that the thesis is conclusion valid, as the conclusion presents the

findings that were most frequent and thus can be said to be the most important consideration mentioned through the study.

4.3.2 Reliability

Reliability in quantitative research concerns whether exact replicability of the study and results is possible, and is thus challenging and can be counter-intuitive to evaluate for qualitative studies (Leung, 2015). Following, he mentions that the essence for reliability in qualitative studies is connected to consistency. In the case of this study, examining reliability could cover an evaluation of whether the results would be the same if a similar study was to be conducted by other researchers. Reliability can be assessed by evaluating whether other researchers would come to the same conclusions as this study presents (inter-rater reliability) or by examining whether the questions used in interviews would produce consistent results (test-retest reliability) (Golafshani, 2003).

This study is based on qualitative methods, aimed at identifying subjective opinions, thus by nature making it difficult to reproduce identical results. Regarding inter-rater reliability, it can be realistic to believe that other researchers would come to the same conclusions by reproducing the study. As we have explored a diverse sample group, the collection of different opinions and insight can create the basis for common conclusions, even though concrete answers may vary. Regarding test-retest reliability, semi-structured interviews are by nature difficult to replicate identically, as the purpose of the method is to allow the participants to freely talk about their opinions and experiences within the boundaries set by the interviewer. In addition to this, the method allows the participants to elaborate their answers, as well as the interviewer to ask follow-up questions not necessarily prepared beforehand. Therefore, it is not certain that the study is reliable according to the test-retest reliability. However, if the questions used in the interviews were used in a more structured context, it is possible that they would provide reliability.

4.4 Data collection

4.4.1 Literature review

To gain initial understanding of what has already been touched upon and explored regarding issues with existing API documentation, much time was used to review existing literature, identifying common areas, findings and indicators, as well as what separates existing research from what we are trying to identify. Through literature review, we uncovered important areas and points of interest to bring into the following data collections. The primary focus was on literature touching directly upon the documentation of APIs and documentation as a resource. However, other relevant literature was also reviewed to gain an understanding of the underlying concepts as well, such as usability and learnability. Relevant findings from the literature review are summed up under related research.

4.4.2 Iteration 1

As we started planning the first iteration of our data collection effort, we identified interviews as beneficial to conduct in order to gain a qualitative insight into the case we wanted to explore. We wanted the participants to be able to speak freely in order for us to gain as much insight as possible, and semi-structured interviews were therefore conducted. The sample group for the iteration was chosen in order to gain a general initial understanding of issues regarding API documentation for novice developers. We wanted to begin collecting data covering issues not necessarily related only to DHIS2, but rather to gain more of a broad initial understanding of the issue, and thus found lecturers to be interesting to interview. As they have an outside-in view on educational projects, and experience with planning both processes and resources for courses, we found that their experiences may be interesting to explore to gain initial understanding. Before approaching API users, we found that this might be beneficial to have as introductory knowledge going into further data collections. In addition to this, we wanted to explore the experiences and perceptions of developers both with connection to DHIS2 and not, to gain initial understanding of their perceived issues. We wanted to broaden the scope also outside of DHIS2 initially, as more general API-struggles may provide interesting insight that may be beneficial to bring into the next round of data collections. Therefore, we chose to interview a combination of DHIS2-developers, non-DHIS2-developers and users with some previous experience with DHIS2, but not necessarily heavily involved on a daily basis. Common for the last two mentioned participants was the limited experience developing using APIs, separating them from the sample groups of previous studies.

The academic participants were asked questions about their experiences with students' struggles working with development projects, as well as their own experiences, both in a lecturing setting and outside of educational work, if they have industry experience as developers as well. The interview guide that was used for said interviews can be found in appendix 1. When talking with the developers in this iteration, we wanted to approach in a way that allowed the participants to speak freely about their experiences. Inspiration for topics discussed were chosen from the same interview guide that were used in the talks with the academics, but instead of following the guide slavishly, we let the participant lead the way with a basis in the categories from the guide. As some introductory questions were aimed specifically towards the academics' educational experience (e.g. "What does the participant lecture in?"), not everything was directly relevant for the developers. Instead of constructing a new interview guide, we decided to follow the existing guide, but skip the questions relevant only for academics. This way, we were able to gain a broad understanding of their experiences, providing valuable insight, while still collecting data that could be compared to the academics' answers during analysis. During the interviews, all participants were presented some sketches of our initial ideas of improvement for general API-documentation, and asked to comment on their

thoughts on strengths and weaknesses with the concepts presented. The sketches were inspired by findings from previous studies, and are displayed in figure 15.

Figure 15: Sketches
Sketches presented to the participants in iteration 1, presenting possible documentation-presentations.

Following the interviews, before proceeding with the next interaction, some observations were conducted in the setting of IN5320 at the University of Oslo. During late summer/early fall, the students in said course are approaching a "Self-paced online course" (UiO, n.d.), introducing them to web development, APIs and the DHIS2 platform. As a teaching assistant in said course, I was able to gain insight into their perceived issues, irritations and struggles through both the said self-paced course and the introduction to work with their project developing an application for the DHIS2 platform.

4.4.3 Iteration 2

Planning for the next iteration, we used the findings and categories from the first iteration to create a guide for the next rounds of interviews. Unlike the more general focus of the previous iteration, the goal for this iteration was to gain a deeper understanding of the experiences directly connected to development using the DHIS2 Web API.

To gain the insight we sought for this round of data collection, we decided to shift the focus onto a purely DHIS2-related setting. Therefore, we proceeded to study a collection of DHIS2 developers in their daily work, conducting interviews to enhance our understanding of their struggles and daily tasks. We also interviewed a number of students at UiO, conducting their project developing

an application using the DHIS2 Web API. As they are required to learn, understand and utilize the DHIS2 data structure and API during the span of the project, interviewing them to study their perception of the process and encountered issues could unveil interesting findings. We also approached a developer associated with the DHIS2 core, to explore struggles and different development- and documentation-aspects from the perspective of the maintainers of the platform core.

The interviews were conducted as semi-structured, with a guide developed based on the findings from iteration 1. The interview guide can be found in appendix 2. The participating students were asked questions about their project: how they had approached the task, which obstacles they had faced as well as how they had conquered these. The DHIS2-developers were asked about their daily work, their process of onboarding when they initially joined DHIS2, what obstacles they faced then and about obstacles they face in daily work. In addition to the interviews, observations of the students conducting their project in IN5320 were also performed, to gain insight into their issues and struggles of learning and utilizing the DHIS2 Web API in a project during a limited period of time.

4.5 Analysis

The analysis for this thesis has been a process of exploring and gaining insight from existing literature, and analyzing and comparing this to empirical data collected through the study, to identify similarities and differences leading up to the result constructing the final contribution of the thesis. The process of analysis has been based on thematic analysis (Braun and Clarke, 2006), for sensemaking of empirical data and attempting to find links between findings and previous literature, shedding light on the similarities and differences. Thematic analysis is often used to analyze qualitative data, as a way to search for repeated patterns and identify themes across a data set (Braun and Clarke, 2006).

4.5.1 Open coding

After each iteration, we collected all interview notes in a common Miro-board, to gain an overview of the findings. We then proceeded with open coding of each interview, to categorize the information we had been presented through the interviews. The findings were presented separated into categories within each isolated interview. In Miro, we constructed an affinity-diagram to help analyze and categorize the data. Categories from open coding of the iterations are presented in table 4 and 5. This way, we were able to categorize internally within the scope of each interview, laying the foundation for further coding and analysis. Figure 16 displays how we organized and presented interview notes from each interview in Miro.

Category from open coding	Descriptions
Examples and use cases	Examples can be important for understanding how to use a system, by presenting for the user how to use the different functionalities available. Use cases can be important for understanding the practical use of a system, connecting examples and descriptions to cases that are likely to occur when using the system.
Level of expertise	The needs of the user group can depend on their level of expertise.
Balance simplicity and depth	Even though there is a need for thorough information, there needs to be a balance between the simplicity and depth in the documentation.
Step-by-step descriptions of the API in use	In order for the user to understand how the system can be used in practice, showing actions step-by-step may be beneficial.
Handling version updates	Core-updates may be frequent, and needs to be addressed clearly.
User involvement	User involvement can help the documentation fit the needs of the actual users better.
Getting inspiration from other APIs	Designing a documentation accepted by all users from scratch can be difficult to achieve. Getting inspiration from established API's documentation may be beneficial.
One page vs. modularized sections	Large portions of text can be perceived as overwhelming for the user. It can be beneficial to consider the trade-off between having information collected on one page and separating it in a modularized layout.
Organizing the content	The content must be organized in a logical way for the user to be able to easily find the necessary information.

Table 4: Categories from open coding of iteration 1

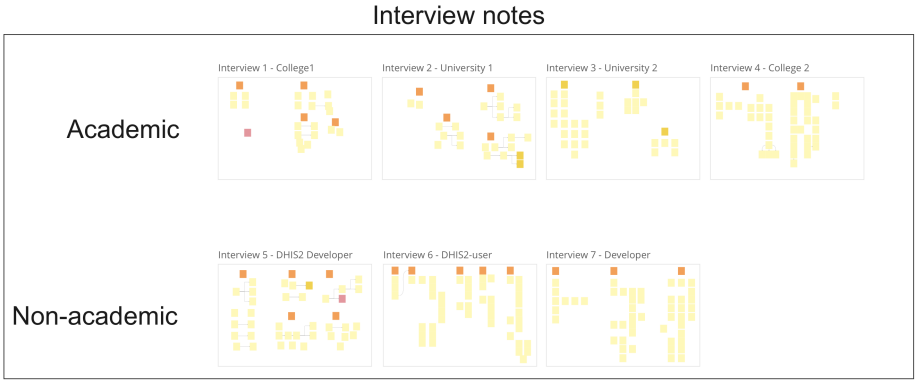


Figure 16: Open coding - findings
Each interview collected in a separate section with findings (yellow notes) categorized internally through open coding (categories represented by orange notes).

4.5.2 Axial coding

After the initial open coding, findings from the different interviews were compared and collected in common categories with axial coding. We looked at the categories from the open coding, and identified common categories for the results. By doing this, we were able to identify common denominators between the different interviews, thus identifying categories relevant to present under findings, initiating the discussion. The relevant notes from each interview were collected, creating an overview of the collective opinions. From iteration 1, the following categories were identified: Potential for improvement, Pedagogics, Content and Structure. As we were conducting interviews together, we collected data relevant for both our problems. Thus, the category "Pedagogics" was relevant for Ashwin, but not for this thesis. Therefore, it was not brought on as an important part further in this study. The category "Potential for improvement" turned out after some review to be possible to merge into "Content" and "Structure", which ended up being the categories of focus of the iteration. Figure 17 displays how categories were collected in the Miro-board.

From the interviews in iteration 2, we identified three main areas of interest: Approach to problem solving, Obstacles faced when problem solving, and Improvements for problem solving. As the previous iteration, also the findings from these data collections revolved around the two important categories, "Content" and "Structure". The findings were therefore categorized and collected in these categories for this iteration.

Within the two main categories, the initial coding of the iterations revealed a large number of smaller categories, as seen in table 4 and 5. Entering the discussion, I therefore decided to proceed with further axial coding, to combine

Category from open coding	Description
Inadequate examples	Examples are perceived as inadequate for the user. Examples are difficult to read due to unknown syntax.
Unclear information	Information is perceived as inadequate, lacking relevant details Information is perceived as too theoretical and difficult to put into practice.
No provided information about platform-/core-issues	The documentation lacks clear information about issues and updates .
Unclear categorization	The content is perceived as uncategorized.
Design not standardized through all parts of documentation	The documentation is made up of several parts, made with different designs, possibly making it difficult to understand relations and navigate.

Table 5: Categories from open coding of iteration 2

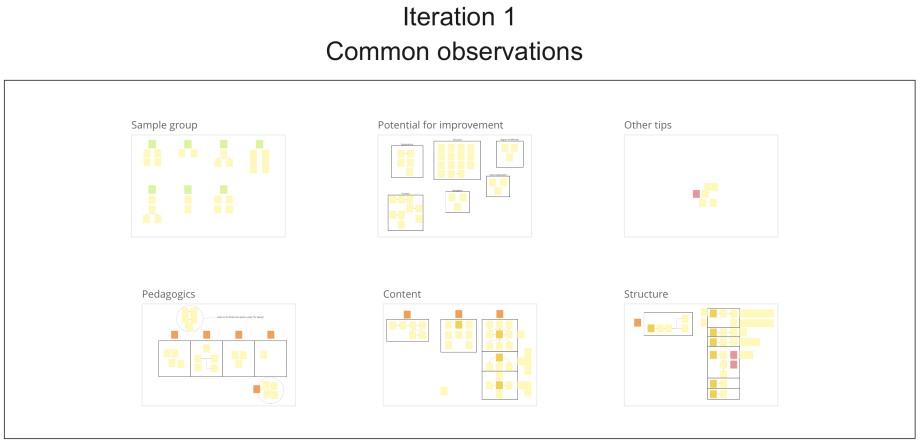


Figure 17: Axial coding - findings
Results of axial coding after iteration 1

identified categories into fewer, larger categories. By looking at the findings, identifying similarities and creating more broad umbrella-terms covering several smaller categories that are touching upon similar areas, the discussion was cleaned up by containing five main categories instead of the original 15, while

still containing the important information. The new sub-categories are presented in table 6.

Category	Sub-category	Covering findings
Content	From documentation to practice	Examples and use cases
		Inadequate examples
		Step-by-step descriptions of the API in use
	User-group experience level	Level of expertise
		Unclear information
		Balance simplicity and depth
		User involvement
	Continous documentation	Handling version updates
		No provided information about platform-/core-issues
		User involvement
Organizing	Layout and design of the documentation	Getting inspiration from other APIs
		Design not standardized through all parts of documentation
	Categorization and organizing of content	One page vs. modularized sections
		Unclear categorization
		Organizing the content

Table 6: Revised categories leading into discussion

5 Findings

This chapter will present the findings from the different iterations of data collections conducted through the study. The chapter is divided into two main sections: iteration 1 and iteration 2. Iteration 1 presents the findings from the introductory interviews and observations, conducted with participants with varying backgrounds, including academics and developers, as described in research method. Iteration 2 presents the findings from interviews conducted with participants involved with DHIS2 in different settings. The iterations are summed up in a table for each iteration, presenting the findings from the iteration.

5.1 Iteration 1

This sub-chapter is written as a joint effort with Ashwin Rajeswaran. Thus, the chapter may contain some similar paragraphs and partial sections.

Iteration 1 revealed two main points of interest for this study: Content and Organizing. Each of these points contains several findings that fall under the respective category.

5.1.1 Content

5.1.1.1 Examples and use cases

There was a broad agreement among all participants that examples simplify and enhance the understanding of functionality of the API. Seeing the technical details put in practice makes actual use easier to picture and understand, making the documentation seem more focused on actual use than only presenting and describing technical information. Some participants mentioned that, when shown the current version of the DHIS Web API documentation, the documentation could greatly benefit from using more practical examples. Figure 18 displays an example from the DHIS2 Web API-documentation that presents an example. As the participants pointed out, the examples do not contain use cases that connect them to actual use.

When discussing how examples may be applied, participants expressed that examples without connection to realistic use cases do not yield the developers much. In other words, if the examples do not reflect circumstances that may be plausible to face for developers, or use cases that are common and likely to be faced for the system, it becomes difficult to understand what the examples are trying to represent. It also becomes more difficult to understand the connection to actual use for different endpoints, thus imagining how to use the API in actual projects, which may be important for some developers when planning their projects and use of APIs to achieve the goals of their project. Relevant use cases can be used in order to explain the API in an actual setting, rather than describing the use in a generic way. By doing this, designers can ensure

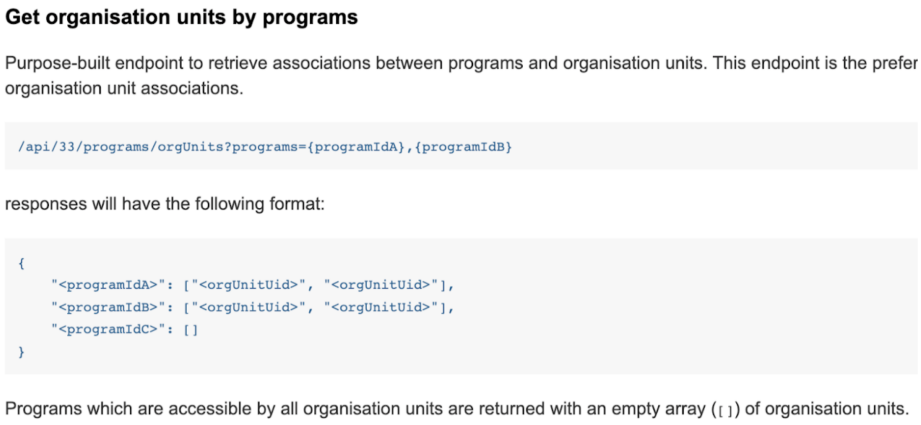


Figure 18: DHIS2 endpoint description
Example of endpoint-description in current DHIS2 Web API documentation.

that users understand how to actually use and utilize the API in the intended scenarios, potentially reducing the amount of misunderstandings and wrong use. Use cases can be an efficient way to describe how to conquer common difficulties and obstacles, regarding the said issue of struggling to connect and convert examples to actual use, hence helping the user utilize the API optimally.

The value of making the examples interactive was mentioned multiple times, as only being presented the solution does not necessarily lead to actual understanding and ability to use the API. Instead, users face the risk of taking shortcuts by only copying rather than understanding use of different components in said scenarios. Connecting the examples to a "sandbox", an area where users can try to use different endpoints in an already set up environment, allowing them to play around with and test the effect of changing different parameters, was mentioned as something that could greatly improve the learning outcome of the documentation. The examples should provide typical responses: what they should look like, what response codes the user can expect to receive etc. The aim of these examples are simply to give the user a low threshold opportunity to test the API in use, without necessarily having to set up their own program for testing it. The examples do not necessarily have to provide actual responses with real data, but they should illustrate how the user can expect to communicate with the API in actual use, thus presenting data that are representative of what the user can expect to receive as a response.

In addition, some participants expressed that the endpoints need to be clearer with what capabilities and requirements they have in order to understand what you can do with them. To do so, they suggested that the documentation should provide information about what parameters are optional, required etc., and in addition provide examples of how the data in responses may change corresponding to the parameters changes. This capability is already present in some parts

of the documentation, for example under metadata identifier schemes, but they do not provide example requests that illustrate how the endpoint communication may look and changes to different parameters affect responses. Figure 19 displays how a metadata identifier scheme from the DHIS2 Web API documentation presents different parameters.

A prominent challenge for students of the course IN5320 at UiO is trying to understand what the responses from different endpoints actually represents and provides, as a complete response in DHIS2 often requires communication towards several different endpoints linked at different parameters in order to be able to make use of the content in the responses. This is due to the interconnectedness of the various metadata in the DHIS2 data structure, which connects different datasets and makes the user dependent on communicating with several endpoints to fetch all data necessary. Figure 20 displays how different endpoints (dataSets and dataValueSets) must be combined to fetch all relevant data for a data element. The example fetches information about commodities from the "Life-saving commodities"-dataset for an organization unit from a specified period.

Data values query parameters

Query parameter	Required	Description
period	No	Period to use, will be included without any checks.
orgUnit	No	Organisation unit to use, supports multiple orgUnits, both id and code can be used.
comment	No	Should comments be include, default: Yes.
orgUnitIdScheme	No	Organisation unit scheme to use, supports id | code.
dataElementIdScheme	No	Data-element scheme to use, supports id | code.

Figure 19: Metadata identifier scheme
Metadata identifier scheme for fetching data from the dataValues dataset.

5.1.1.2 Level of expertise

An important thing that was mentioned several times during the first iteration of data collection, was the importance of identifying the group that is targeted with the documentation. Experienced users may have different needs and requirements than novice users, thus should the documentation be adjusted to the primary user group in target. The findings show that while experienced users often may prefer large documents with the possibility to "CTRL-F" (search) through the document, novice users can benefit from a more step-by-step, guided approach. Searching through a large document requires some degree of understanding of the possibilities that the system provides, as well as some knowledge about what to look for in order to find what's relevant. Novice users may have a need for a more practical approach, providing examples and visual components rather than plain text- and table-content style encyclopedia in the documentation.

```
export function fetchDataQuery(){
  return {
    dataSets: {
      resource: "dataSets/ULowA8V3ucd",
      params: {
        fields: [
          "dataSetElements[dataElement[name,id,categoryCombo[categoryOptionCombos[name,id]],dataElementGroups[name, id]]]",
        ],
      },
    },
    dataValueSets: {
      resource: "dataValueSets",
      params: ({ orgUnit, period }) => ({
        dataSet: "ULowA8V3ucd",
        orgUnit: orgUnit,
        period: period,
      }),
    }
  }
}
```

Figure 20: DHIS2 - example from project
Function fetching information from a DHIS2-dataset from a project where the DHIS2 Web API was used.

One apparent claim made by the participants having experience with DHIS2 was that the degree of difficulty and learning curve for DHIS2 is very steep. One of the participants stated that the content *"never hits where I am currently standing"*. In other words, the participant expresses that finding content adjusted to their knowledge level about the underlying logic of the system can either be too hard to grasp, or too simple for their use cases. As their knowledge increases, they must proceed further into the learning resources and documentation, as what they have already read is below their current level of knowledge. However, the next stages are perceived as too advanced, creating the mentioned issue of struggling to locate resources at the stage the user is currently at. The participants expressed that, in the case of DHIS2, the data model is the most challenging component of the API to understand, suggesting that the data model and metadata needs more attention in onboarding and coaching efforts for new developers to be able to settle in the system with ease.

Some participants also suggested that the content's level of difficulty should be grouped in some way. A few participants noted the necessity to take more "baby steps" because the documentation's material can quickly become advanced and, consequently, less user-friendly. This was either proposed as a way to differentiate between content for novices and content for advanced users, or as a way to modularize the documentation's content into smaller steps that allow for gradual mastery. As part of the latter participant's idea, they also reported that incorporating gamification techniques could boost the content's engagement and learnability by making the distinct modules more comprehensible to the target audience.

5.1.1.3 Balance simplicity and depth

Some participants mentioned that the documentation could be falling short because it is trying to address complex information textually, which may be easier to comprehend if presented differently. For structural and architectural understanding (e.g. the data model in the case of DHIS2), figures and visual presentation like diagrams may be beneficial in order to understand connections and relations between different parts of the system. The academic participants expressed that visualizations and illustrations of important concepts like data structure etc. may facilitate further understanding of the metadata, which in turn may help the user understand how to utilize the API best possible in projects. These elements can supplement text-descriptions, thus making technical concepts easier to understand for the user, as well as allowing the documentation-designers to compress textual descriptions, and rather refer to the figure.

However, not all information can be presented visually. Where text-only-descriptions are used to explain details, it is important to focus on writing good and precise descriptions that keep the target group in mind regarding formulation. To ensure understanding and avoid misunderstandings, the language in use should be adjusted to who the description is intended for. However, deciding what is good and precise can be difficult to achieve, as all users possess a subjective opinion regarding what is understandable and precise. It is therefore important to consider what information is important to present, and what may be excess, to keep the descriptions precise and relevant.

5.1.1.4 Step-by-step descriptions of the API in use

When targeting a novice user group, providing step-by-step descriptions of the possible actions and interactions may be beneficial. There are different opinions on what should be presented in which order, but there's a broad agreement that simple explanations of the different steps in an interaction may help to simplify the process of utilizing the API more efficiently. "Steps" in this context can be understood as the different actions and activities that, put together in a sequence, make up a complete interaction. Actions to cover could include common activities that all users will have to do, e.g. authentication/getting access and/or an intro to getting different data from the API. It is important to ensure that these descriptions are well formulated and written, so that the user can both follow along and understand the process which is being done. Another important consideration is that the descriptions must be complete and include everything that needs to be done in order to achieve the described goal. Participants were however clear that external resources, e.g. descriptions of how to utilize the underlying technologies (e.g. REST) should be linked to their respective resources. If the designers of the documentation include all information in one place, users may experience the documentation as overwhelming and thus perceive the documentation as less useful and usable.

The step-by-step interaction-descriptions should focus on including a brief explanation of different endpoints, as well as what parameters they accept. Exactly how the descriptions should be designed depends on what is being presented. However, participants suggested that tables can be used for a simple overview and understanding of the different endpoints. The descriptions should include what the endpoint expects to receive (what is required), as well as what it accepts (what is possible to send). It should also include a short description, explaining what the role of the endpoint is, what its intended purpose is and what it provides. Figure 21 presents a possible presentation of an endpoint in documentation. The example contains a title, a short description of the endpoint (what it is used for), the base URL-path to communicate with the endpoint, a scheme of parameters with a required (Yes/No)-field, and an example displaying the endpoint in use. The example in the figure is thought to be possible to change by using the left dropdown-menu to select between different use cases. The example is thought to be possible to get presented in different programming languages, for easy adoption into projects, by selecting the preferred language using the right dropdown-menu.

5.1.1.5 Handling version updates

One significant challenge for both the lecturers in the educational settings, and developers in development settings, was keeping up to date with new version updates and changes as they occur. Whenever the API receives an update, the documentation must be updated accordingly, in order for it to stay up-to-date and relevant for users. One of the academic participants mentioned the challenge of creating educational resources that are up to date, because APIs often can change rapidly due to innovation and fixes of different features. A concrete example mentioned by the participant, was that creating course material connected to the Android development API can be challenging, because most content becomes outdated before the end of the term, due to rapid updates. Thus, creating resources supporting development using the API can prove itself difficult. They mentioned that creating guides that utilize third party APIs will require very frequent updates, as without updates the developers may try to implement a request and try to communicate towards an endpont that has been phased out or become legacy. In other words, it can become harder to teach best practice of the use of said API when some endpoints may be removed and others may be changed to a point where existing applications may break, without proper information.

Even though the mentioned example covers the Android API, it can still be relevant also for DHIS2, as updates of the API are necessary to keep the system relevant and avoid issues. From the perspective of the developer, issues regarding obsoletion of both API-functionalities and documentation may present major issues. Figure 22 presents an example of a case where a section of the documentation has been deprecated, thus no longer serving the same purpose even though it is still available online. This presents new obstacles for developers,

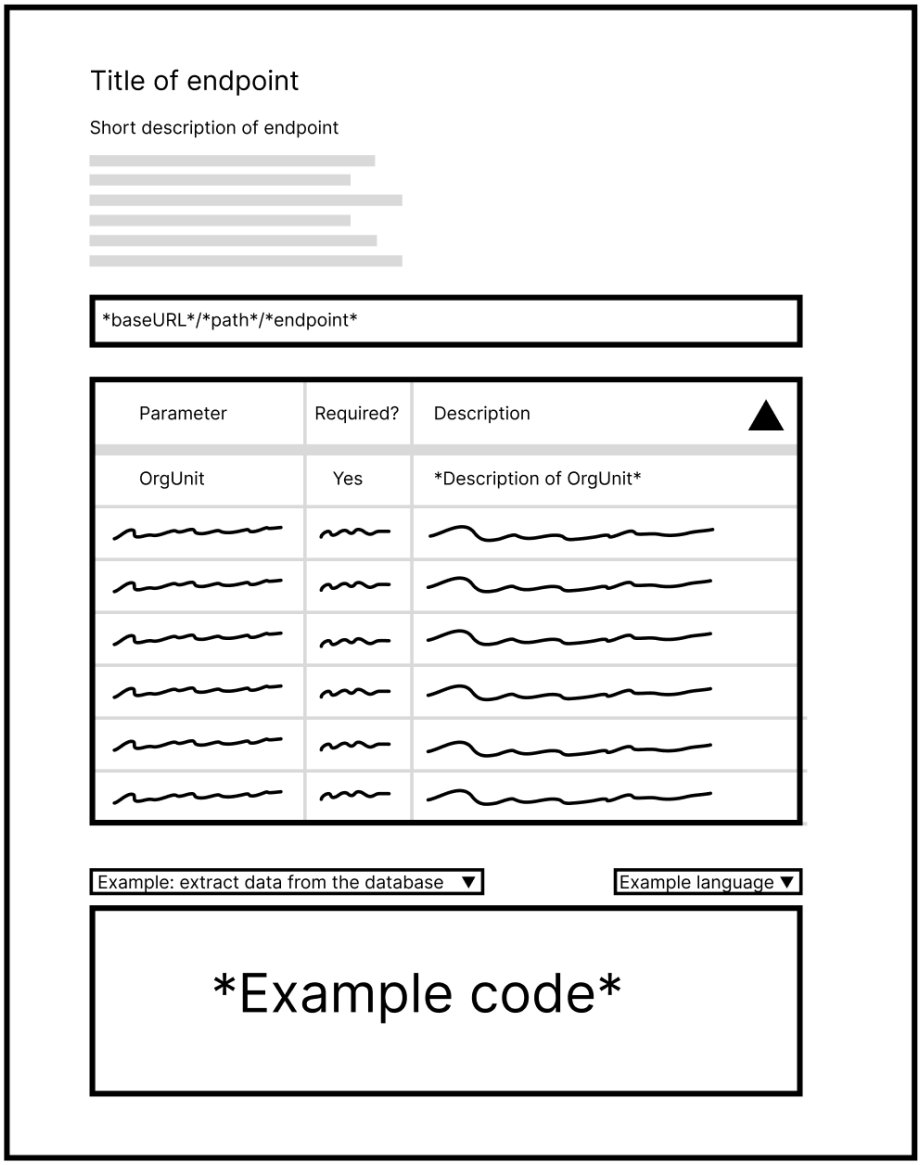


Figure 21: Possible endpoint-layout
A possible layout of endpoint-documentation.

as their process is highly dependent on being able to find the reason for errors. After debugging the code, a natural next step when developing with APIs may be to approach the documentation for possible reasons and solutions. However, if cases like this example are not communicated clearly, developers may find it more difficult to error-handle and find correct resources.

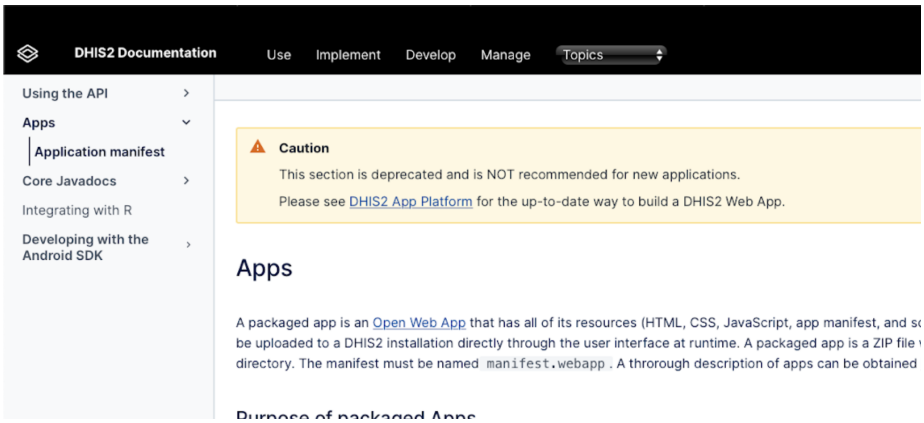


Figure 22: DHIS2 - deprecated section
Example of an outdated section from the DHIS2 Web API-documentation.

5.1.1.6 User involvement

One participant mentioned the possible value of allowing users to contribute to the documentation by making suggestions to changes or additions from experience they have gained. After using a system for a period of time, users gain experiences that may be beneficial for others to learn from, both regarding practical approaches and possible improvements regarding factors like formulations and presentations in resources like the documentation. Therefore, by allowing users to participate and contribute with their experiences, other users, such as novices, may benefit from learning alternative approaches, possibly simplifying their process of understanding how to use an API. This means that users can contribute by developing alternative methods for achieving a goal – for example suggesting multiple ways to fetch data from endpoints. By allowing users to suggest changes in the written documentation, it is also possible to expand the number of phrases used in the documentation, possibly enhancing the likelihood that individuals will find what they're seeking by searching.

5.1.2 Organizing

The conducted interviews revealed three main areas of focus regarding organizing that can influence the users understanding and efficiency when approaching the documentation of a new API.

5.1.2.1 Getting inspiration from other APIs

Some participants suggested looking at existing documentations by large corporations (such as Google's Android development documentation, Twitter etc.) for ideas on how to design the user journey and user interface of API documentations in order to enhance the current DHIS2 Web API documentation. As

described by participants, these large corporations have APIs with documentations that are public and exposed to large amounts of users internationally. Thus, it is likely that they have grasped the variables that determine the success or failure of documentations. As a result, it may be good to draw inspiration from them when developing recommendations for other API documentations. Instead of "reinventing" how documentation should look from scratch, fetching inspiration from the documentation of established, public APIs with broad user feedback, inspiration can be collected and create the basis for documentation, which can then be tailored to the specific needs of the API.

5.1.2.2 One page vs. modularized sections

The majority of the sample group stated that the content's disorganization confuses readers. In other words, there is so much information available that readers feel like they are "drowning" and have few tools to help them find what they need. Participants stated that there should be a trade-off between having a one page, plain text-presentation, and a presentation of content in separated, modularized sections, where the different categories of information are placed apart from each other (e.g. authentication, fetching data, posting data etc.).

The participants were unified in the opinion that this modularization of content should not fully replace the plain text-presentation of the documentation, but rather supplement it. There is a tension regarding how modularized the documentation should be, as too much modularization may impair the experience of locating information if there is too much to navigate. Seeing how experienced users may appreciate the possibility to search ("CTRL-F") through the documentation rather than navigate through a modularized structure, fully replacing the complete overview can potentially affect the efficiency and satisfaction of the experienced programmers significantly. However, the majority of participants stated that dividing the documentation into multiple pages can result in better organization, which may assist users in locating the desired content. New and novice users may not be familiar with APIs, their structure or naming conventions, making the mentioned preference of experienced users potentially difficult for novice users. Without sufficient knowledge of said factors, searching through a large overview of all available endpoints may be very difficult and time consuming.

It was also mentioned that searching doesn't always provide the result that you're looking for. Sometimes searches for certain things are done via Google, and in these cases, a modularized, separated content-presentation may turn out more beneficial. For internal search functionality, said obstacle of not finding the result you're looking for from a search may be a large irritation. A possible approach to avoid this presented by participants was the possibility to have a "similar"- or "Did you mean ..."-functionality for searches, when searches don't provide a result but there are similar results that the user might have been searching for. This way, it is possible to explore and possibly understand more

of the documentation without the thorough understanding of the underlying structure beforehand.

5.1.2.3 Organizing the content

As a continuation of the previous point, the participants touched upon how to organize the content. The participants identified four main areas of focus regarding organizing:

1. **Structure**
   The content can be collected on a single page, but this requires some considerations. In order for this kind of collection to be understandable for novice users, the page requires some sort of structure, making it manageable for new users. A large page filled with information faces the risk of being perceived as overwhelming, hence creating a thought out structure is important to avoid this.

   Another way to avoid an overflow of information, is simply to compress, or "hide" everything that is not in use or directly relevant, so that only the relevant information for each case is displayed by default. This can be solved by separating content into collapsible items, creating sub-sections etc. Avoiding large collections of information reduces the risk of the documentation being overwhelming, thus possibly making it more manageable. It can also be important for designers of documentation to consider what information is relevant and what is not, to hide information that's not necessary for that specific use. By reducing information that is not directly relevant for each case, but rather more of "nice to know"-information, reducing the feeling of information overload for the user can be easier. It is also important to consider the target group when designing the documentation. Beginners may have different needs than experienced users, thus it is important to separate "beginner-tutorials" from more advanced descriptions.

2. **Separate documentation from side-resources**
   For relevant side-documentation that may be beneficial to include in the context (e.g. resources abour REST APIs, frameworks etc.), it can be wise to link these rather than describing the external tools and resources in the documentation. It may also be beneficial to separate API-documentation and other system-documentation, e.g. separate the technical documentation from the comprehensive data-structure- and descriptive documentation in the case of DHIS2.

3. **Prioritize after importance**
   Participants mentioned that developers often want to get started quickly, without having to read more material than necessary. It is therefore important to organize the documentation according to the relevance and importance of the different sections. Given that the documentation begins

by explaining the most important, basic aspects of the system (authentication/getting access, getting started etc.), it is possible that new users will experience a quicker and less complicated introduction to the system. By ensuring this, it is easier for them to get started and gain a basic understanding of the API, then instead of reading everything before starting, rather return when they face concrete issues as those appear.

It is important to introduce what the API does and what overall functionality it provides. E.g. the purpose of an API can be to provide data about public transport routes, about food recipes etc. This must be communicated clearly in order for the user to understand what the purpose of using the API is. This should however be done carefully and in a short, informative and precise format, only presenting information directly relevant and important to understand the API, to avoid the user becoming uninterested before the more important content for use is reached. This introduction can contain a summary of what data/functionality the API can provide, as well as briefly how it provides the different results. Further, it is important to begin focusing on usability - how the user can begin using the API. Describing the key actions that facilitate use of the API is important for the documentation to serve a purpose for new users. Making this easily available early in the documentation will simplify the process of getting started. These descriptions should include brief explanations of the different endpoints, as well as how to use them. Exactly what it should contain depends on the individual case. However, it should contain information that is directly relevant for the use, allowing the user to get a quick introduction to how it should be used.

At last, after the most important content regarding the endpoint for the developer has been described, the documentation can continue describing what participants mentioned as "less important information". Exactly what this mention covers depends on the API and its purpose, but typical examples that were mentioned were e.g. information about rate limit, pagination etc.

4. **Define terms**
   For novice developers, technical terms can be difficult to understand. It was mentioned by participants that the writers of documentation should make sure to define important terms, in order to avoid confusion. However, including this must be done carefully, as too much information may exacerbate the previously mentioned risk of information-overflow. A solution to this risk proposed by a participant could be to link to a separate resource with relevant descriptions of terms and concepts. This way, it is possible to ensure understanding of relevant terms while still avoiding the risk of cognitive overflow.

5.1.3 Iteration summary

The first iteration of data collections was conducted to gain ground knowledge regarding perception of documentation, from a combined view of API-users and academics. The sample group consisted of seven participants, mixed between lecturers and developers, with and without knowledge of DHIS2. The iteration revealed several topics of interest. Relevant findings for this case are summed up in table 7.

Category	Finding	Description
Content	Examples and use cases	Examples can be important for understanding how to use a system, by presenting for the user how to use the different functionalities available. Use cases can be important for understanding the practical use of a system, connecting examples and descriptions to cases that are likely to occur when using the system. Lack of examples created from use cases can make documentation more difficult to use.
	Level of expertise	The needs of the user group can depend on their level of expertise.
	Balance simplicity and depth	Even though there is a need for detailed information, lack of balance between simplicity and depth in the documentation can make it difficult to use.
	Step-by-step descriptions of the API in use	Step-by-step descriptions of functionalities can make use of the API easier to comprehend for novice users.
	Handling version updates	Core-updates may be frequent, and needs to be addressed clearly.
	User involvement	Allowing for user involvement can help the documentation fit the needs of the actual users better.
Organizing	Getting inspiration from other APIs	Designing a documentation accepted by all used from scratch can be difficult to achieve. Getting inspiration from established APIs' documentation can help designers succeed.
	One page vs. modularized sections	Large portions of text can be perceived as overwhelming for the user. Modularizing some content into separate sections can be beneficial to consider.
	Organizing the content	Organizing of the content can have a significant impact on the user experience for the documentation, and needs to be considered in light of the user group in target.

Table 7: Findings from iteration 1

5.2 Iteration 2

To supplement the findings from iteration 1, providing knowledge about API usage in general, we proceeded with a more targeted approach towards the DHIS2 Web-API in iteration 2. The data collections were conducted with DHIS2-developers, as well as students participating in the course IN5320 - Development in Platform Ecosystems at UiO, as a combination of interviews and observations. The goal was to gain insight in the development-process using DHIS2, identifying different approaches, obstacles experienced, as well as possible improvements to problem solving.

5.2.1 Approaches to problem solving with DHIS2

From the interviews and observations, two main approaches to problem solving with DHIS2 were identified: "Exploring" and "Use of documentation". In addition to this, a number of other different approaches were mentioned, covered under "Other approaches".

5.2.1.1 Exploring

Even though the participants approach problem solving with DHIS2 differently, many responses were similar and can be collected in one category: exploring. Inspecting requests and responses was mentioned as an effective way of exploring the DHIS2 API. Seeing how DHIS2 has quite a complex data structure, learning how to use the structure in practice by exploring can make understanding relations easier. This also allows for the API to be explored without much reading of background- and technical information. Once you have a basic functioning request, changing the parameters quickly provides a response on whether this change works/is allowed or not. The participants noted this as an important part of their process of getting to know new APIs, gaining initial understanding of basic relations.

All participants mentioned trial and error as a go-to approach when beginning to use a new technology, including getting to know the DHIS2 API. Using API-tools such as Postman to send requests, while changing the different parameters to see the effect, was highlighted as a natural early stage activity of exploring new APIs. This way, the participants learn what works and what does not work by exploring at their own pace, adjusted to their own skill level. A mentioned approach to trial and error was to experiment with different requests test-driven, by creating a set of tests initially which create the goal for further programming, which allows for a more targeted approach than just testing without a concrete purpose. Creating initial tests to be passed, then approaching the API iteratively like shown in figure 23, the developer gets the opportunity to explore the API by adjusting requests on the go to pass the tests.

Exploring freely like mentioned above was described as a practical way of learning. Seeing how not everyone necessarily views theoretical learning as

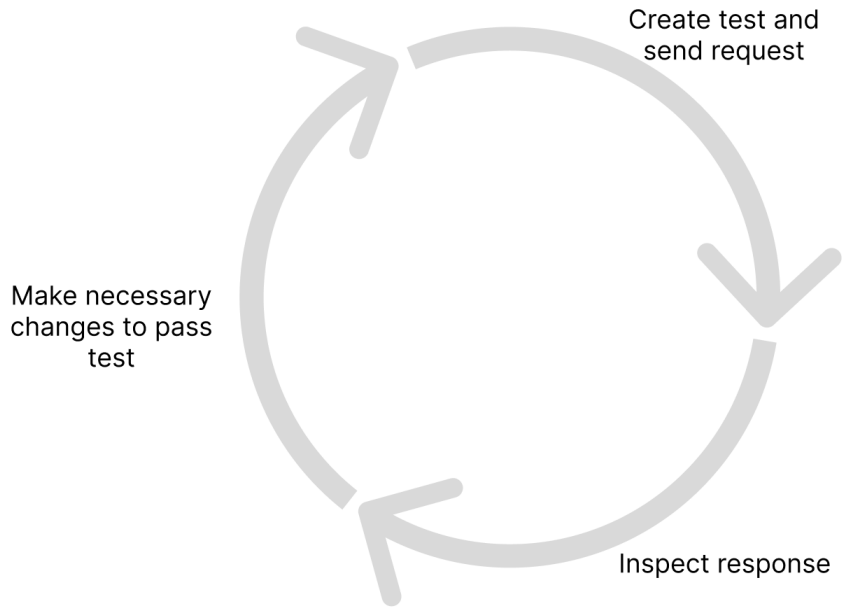


Figure 23: Iterative approach to API-development

the most efficient way to gain an understanding, a more practical approach like self-exploring may be more beneficial. The participants actively working with DHIS2-development on a daily basis mentioned how they look at the data through what they described as the "3 dimensions of DHIS2": Where we are trying to collect data, the period (time and place) the data is situated in, and what data is being collected. These three dimensions create a basis for the said participants to categorically approach the fetching of data, saving both time and effort in the process. The dimensions are presented in figure 24

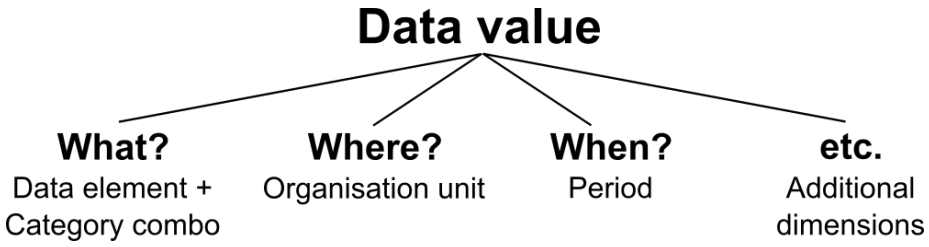


Figure 24: Dimensions of DHIS2
3 dimensions of DHIS2. "Additional dimensions" describes everything not covered by the other three dimensions, e.g. fields etc.

One participant mentioned this approach as an effective way to establish what was mentioned as a "minimum viable payload" - the minimum required to collect the relevant data. By exploring the opportunities the API provides, as well as what changing different parameters does, the user is able to create a request that fetches the basic data needed to solve a task. "Minimum viable payload" can be seen from two different perspectives: the user and the API. A minimum viable payload for the user can be seen as a request that creates the basis for a more concrete request, by allowing the user to gradually filter out excess data, making the request gradually less resource demanding. E.g., a minimum viable payload in DHIS2 from the user's perspective can be a request that fetches all data from a specific district, allowing the user to proceed with narrowing down the search to fetch only the necessary data. Another participant emphasized the importance of this process, as gathering more data than required exposes the API to unnecessary data-traffic, possibly affecting the performance of the API for other users.

"One have to understand what kind of data is being collected and what needs to be collected"

- Participant from iteration 2

From the APIs perspective, a "minimum viable payload" can be seen as something that filters down the search enough to let the API respond with just the exact information requested. The two perspectives are illustrated in figure 25.

5.2.1.2 Use of documentation

Documentation was a resource mentioned as a natural starting point for the approach to problem solving with DHIS2, as well as a resource that developers return to when problems appear later in the development. A prerequisite for exploring the API as mentioned above, is to have access to the API - a possibility to send requests. The documentation often creates the basis for this, by providing requirements for authentication as well as endpoints to begin exploring. The documentation was mentioned as an important resource to see the opportunities available when beginning to use an API, such as DHIS2. Whether it be the structural descriptions or the technical documentation, this lays the foundation for understanding of the possibilities the API provides, and what it can be expected to provide.

5.2.1.3 Other resources

A number of other different resources were mentioned in the interviews. The students mentioned teaching assistants (TAs) and course-specific forums as an essential part of their understanding, while the DHIS2-developers mentioned the DHIS2-community as important for their process of problem solving. Even

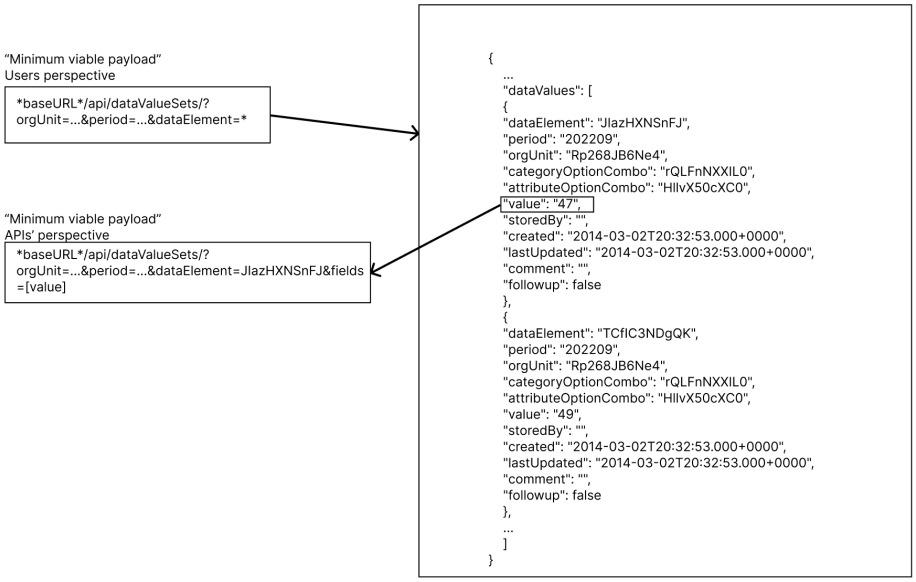


Figure 25: Perspectives on "Minimum viable payload"
Presentation of the different perspectives on "Minimum viable payload". Note that the example is purely for illustration and is not necessarily an accurate request/response.

though these are different, they represent a similar resource: community resources. Both TAs in course settings, and forums/different community resources allow for internal sharing of knowledge within the context, which can help others understand and pass faced obstacles. An area where users can ask questions regarding obstacles they face, and get feedback from others who have conquered the same issues, allow for sharing of knowledge regarding issue-handling and sharing of experience. Internal sharing of knowledge was highlighted as important for individual understanding, as seeing how other developers do things in practice was mentioned as an important part of understanding how to put the structure into practice. What is covered under "internal" depends on the context, but was by the participants describing the community working with the same system or project (e.g. DHIS2 community for DHIS2-related work). The students delegated the responsibility of understanding the API to one person, while the others performed other tasks. After some time, they practiced knowledge-sharing, by working together to understand how to use the API, led by the person responsible for learning the API from the beginning.

Observations from both the students' project, as well as the DHIS2-developers at work, revealed that pair/mob programming and discussions in groups is a common approach to problem solving. As people possess different experience, sharing of knowledge and experience by working in pairs or groups can lead to

better overall learning and performances.

External resources, resources not specific for the system or context, are also important when facing issues and obstacles. Resources of importance that were mentioned by several participants were Stack Overflow, YouTube and Google. Stack Overflow is a popular resource for developers, providing a forum where technical discussions can be conducted amongst developers. Defined as a Community Question and Answer-platform by Ahmed and Srivastava (2017), Stack Overflow serves as an important resource in problem solving technical projects. With more than 10 million users, 92% of all questions answered and a median answer-time of 11 minutes, Stack Overflow is a natural resource for help with technical issues (Meldrum et al., 2020).

Participants also mentioned YouTube as an important resource for learning APIs, DHIS2 in particular. The DHIS2 YouTube-account provides a series of different tutorials, webinar recordings etc., making it a possibly important part of the learning process of DHIS2, explaining data-structures, usage of different parts of the system and community-updates ("DHIS2 - YouTube", n.d.). Other technologies also often have tutorials and other video resources available through YouTube. Resources are often made either by the creator of the technology, or independent video-creators (e.g. Web Dev Simplified for web-development, Programming with Mosh for various technologies etc.), making YouTube an important resource for information seeking when problem solving. Figure 26 displays some examples of videos available through the DHIS2 YouTube-channel.

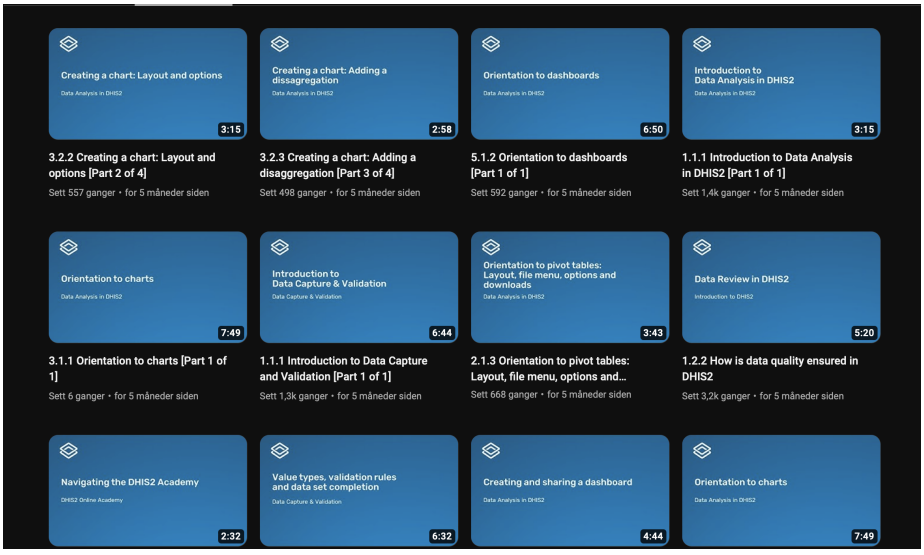


Figure 26: DHIS2 YouTube-channel
Examples of videos available on the DHIS2 YouTube-channel

To find the different resources available, Google was mentioned as an important tool to locate what's needed. The students mentioned Google as the "only" way to locate some resources, as navigating through the official documentation sometimes provided outdated or not-sufficient resources. For example, they experienced that the official documentation directed them to an outdated version of the UI-documentation, while Google provided links to a working version. Whether it be API-specific resources, or other supporting resources, using established and wide-covering search engines like Google can be an efficient approach.

5.2.2 Obstacles and possible improvements to problem solving with DHIS2

5.2.2.1 Content

Obstacles An obstacle regarding content, especially in the case of the interviewed and observed students, is that the current documentation is perceived as lacking adequate examples with relevant info. Examples in the documentation are some places written as cURL-requests, a format the students expressed little knowledge about. Seeing how DHIS2-applications are often written in React, especially in the case of IN5320, examples displaying requests formatted for use in said context could possibly be more informative and relevant. Figure 27 displays a request in cURL, and the same request taken from the context of a DHIS2 React-project. Curl-requests are perceived as more aimed at experienced developers, making them effective, but less understandable for user groups with less experience. The students expressed that examples like the cURL-request displayed in figure 27 below, were difficult to comprehend. They also struggled to convert the examples from the documentation into practical applications for use in their projects, making the provided examples feel inadequate for the students.

The examples were also perceived as difficult to understand, as they seem to expect an underlying understanding of how the API works. Thus, the examples do not describe relations between endpoints, functionalities etc. sufficiently so that novice developers understand what happens when interacting with the different requests and responses. For example, the relation between the DHIS2 datasets dataSets (which contains information like item-ID, name, time registered etc.) and dataValueSets (which contains values registered for objects within dataSets) is not perceived as sufficiently described for students in IN5320, for them to easily understand that these datasets must be combined to fetch complete and comprehensive information about dataElements (items registered in DHIS2).

Another content-obstacle with the current documentation, is what was mentioned by several participants as insufficient information. This perception was caused by what was described as unclear explanations, meaning explanations that were either too broad (not specific enough) or too narrow (too specific to

cURL

```
curl -d @datavalueset.json "https://play.dhis2.org/demo/api/dataValueSets"
  -H "Content-Type:application/json" -u admin:district
```

React

```
import { useDataQuery } from "@dhis2/app-runtime";

const request = {
 request: {
  resource: "/demo/api/dataValueSets",
 }
}

const sendRequest = () => {
  const { loading, error, data } = useDataQuery(request)

  if (error) {
    return <span>ERROR: {error.message}</span>
  }
  if (loading) {
    return <span>Loading...</span>
  }
  if (data) {
    //Use data
  }
}
```

Figure 27: Curl vs. React
Request towards the dataset "dataValueSets" presented in cURL-format and React

understand). One participant mentioned the struggle of understanding possibilities by describing how *"... the API does not tell you what you can do... [the documentation] tells you how to do things, but not what can be done with the API."*. By this, the participants underlined the mentioned struggle of parts of the documentation being too specific, as it provides information about how to do one thing, but not describing the opportunities available. Another participant described the opposite struggle, too broad information: *"[The documentation] gives you enough info to understand that it is possible to do things, but not enough to understand how to do it."*. By this, the participant expresses the struggle of insufficient thoroughness in some parts of the documentation. They understand that something can be done, but not how to actually do said thing, thus leaving the documentation perceived as less useful than expected. This describes two issues regarding information: insufficient information about what can be done, as well as insufficient information about how to do things. Together, these issues present an obstacle for new users understanding the possibilities of the API. Finding a balance between said issues, where users are both able to understand what can be done with the API, as well as how to do said thing can be crucial for the users motivation to interact with the API. It is how-

ever difficult to conclude with what is the sufficient amount of information and what is too much, as users may subjectively perceive information differently, making what is perceived by one user as just enough possibly perceived as way too little, or too much by another.

At last, a struggle that was mentioned, is what was described as missing information about issues, meaning that core-issues and updates are not communicated properly from the platform owner. This was primarily an issue described by the DHIS2-developers, but during the span of the students' project, they experienced several components and parts of the documentation that were not working. Some of these had indications that they were deprecated, but some had lacking information about issues. Figure 28 presents an example of a deprecated section As seen in figure 29, small changes in the URL-path could separate working documentation and seemingly empty documentation. The not-working filepath was found by students when they attempted to google the UI-documentation during their project, with no indicator that the URL caused the issue. In order for the documentation to serve its purpose, it was suggested that platform owners and -administrators inform about deprecations in a clear manner, with provided replacements (if existing) to avoid this confusion.

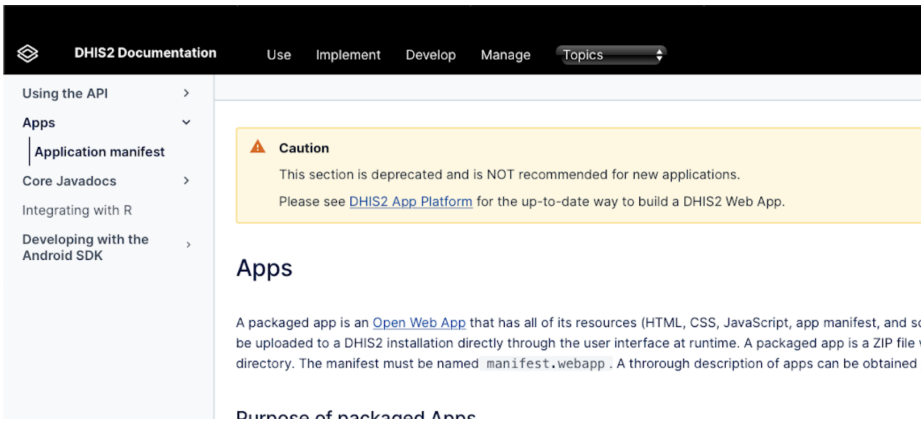


Figure 28: Deprecated section
Deprecated section in available documentation

Potential for improvement As previously mentioned, the data structure of DHIS2 can be perceived as complex, and can thus be hard to understand sufficiently. Participants highlighted visual elements as an important improvement that may strengthen their understanding of relations, thus making development and usage of the DHIS2 API more efficient. In the case of DHIS2, understanding how things are connected can be important for the ability to utilize the system. However, it was noted that the documentation should visualize not only the overall structure, but also how to apply the structure in practice in a

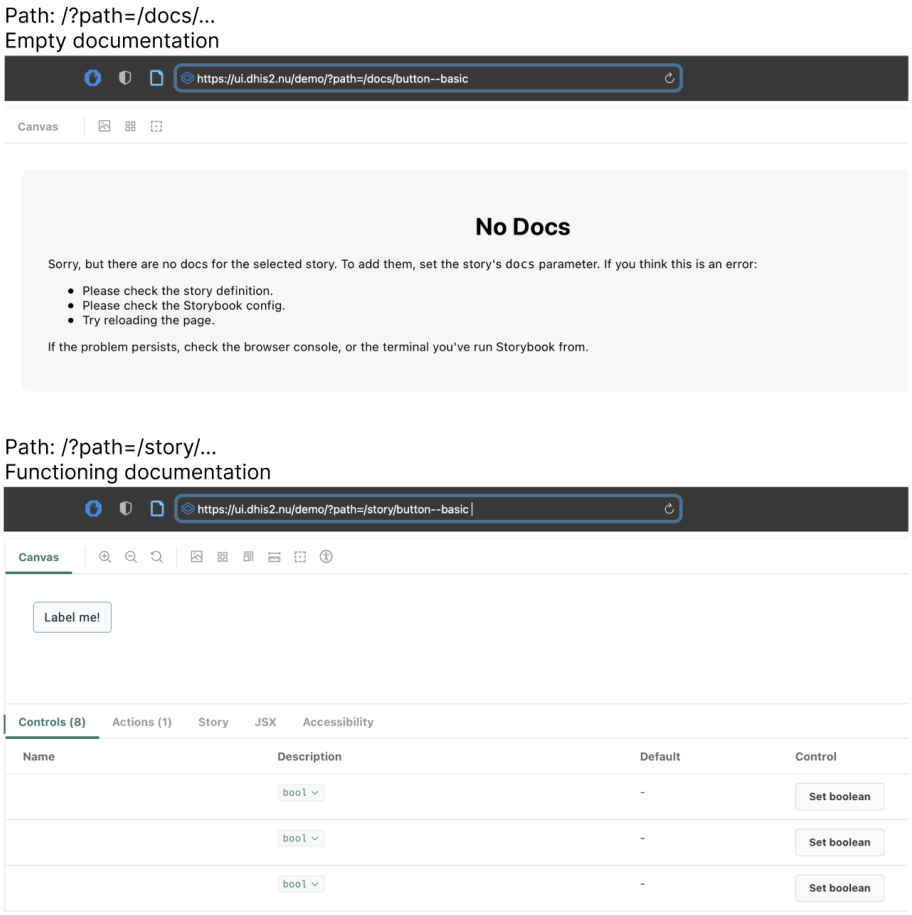


Figure 29: Path issues
Example of documentation available online, where small changes in the path decides whether the documentation contains information or not.

project. One participant mentioned the importance of *"... teaching concepts, not technologies."*. By this, the participant underlined the importance of not only getting to know how to perform actions, but also understanding how requests work in the underlying structure. Visualizing this can possibly have a large impact on the outcome for new users' understanding of the overall system.

A mention by several participants is the importance of good, understandable and practice-oriented code examples. What's considered understandable may of course vary from person to person, but the important common denominator is to adjust the examples to the user group in focus, and use cases that reflect actual intended use of the system. This way, users can be more likely to find the ex-

amples useful for gaining understanding of the use. "Understandable" can vary based on experience, thus should examples be adjusted depending on what the focus of the relevant documentation has. As previously mentioned, a suggested improvement by participants was to combine an encyclopedia-style documentation with a guided approach. Participants noted that the documentation should include a balance between short descriptions and examples, combining theory (descriptions) and practice (examples).

A common perception by the participants was the idea that the documentation should be a presentation of different sequences of actions and interactions, describing how to use the system. A significant issue, especially for the less experienced participants, was the difficulty identifying how to use the information from the different parts of the documentation in practice. Even though there were descriptions and general examples, the participants struggled to use these in practice in their projects. The impression regarding exactly how this information should be presented varies based on experience. However, all participants agreed that the documentation should present an easy to find, practical approach on how to perform different interactions in order for it to improve their problem solving. A possible way to achieve this is by integrating relevant use cases for different scenarios, to display a practical application of the different endpoints.

Another suggestion presented by participants, aimed at continuously improving the resources, is to implement a way for the community to contribute and expand the documentation with relevant experiences and best practices. This way, experienced users can share their ways of working and "shortcuts" displaying good practices to novices, hence participating in sharing of knowledge, as well as possibly simplifying debugging for all users by displaying common errors and solutions directly in the documentation. A suggestion by participants was to implement a Wikipedia-style community participation-functionality. This way, everyone has a possibility to contribute and take part in the improvement of system documentation, while improvement-suggestions are validated and verified by system-moderators in order to ensure quality. This can also improve the continuous usability of the documentation by regularly updating procedures with up-to-date techniques and tips.

5.2.2.2 Organizing

Obstacles Looking at specific issues and obstacles with the current DHIS2 API-documentation, a major obstacle mentioned both by students and developers was that the documentation lacks standardization. It was mentioned that *"... everyone does things their own way. "*, indicating that the different sections of the documentation does not necessarily follow the same design pattern, making it difficult to navigate through the documentation as a whole. An example of the described differences can be seen in figure 30, 31 and 32, displaying three different designs within the documentation of the same system. Even though the displayed pages have some similarities, e.g. a menu located on the same

side, the design differs in such degree that some users can struggle connecting them together, leaving the users with the impression that the parts of the documentation are independent of each other without connection.

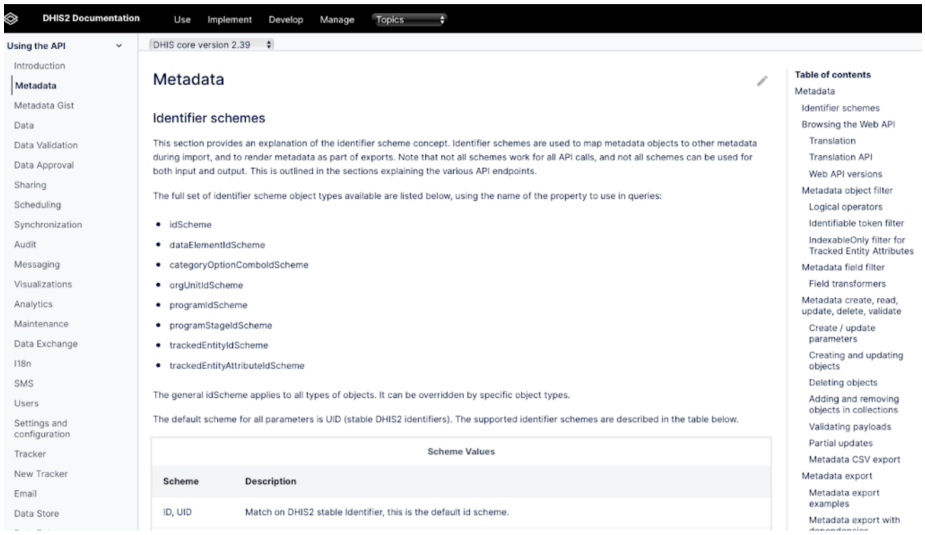


Figure 30: DHIS2 developer documentation

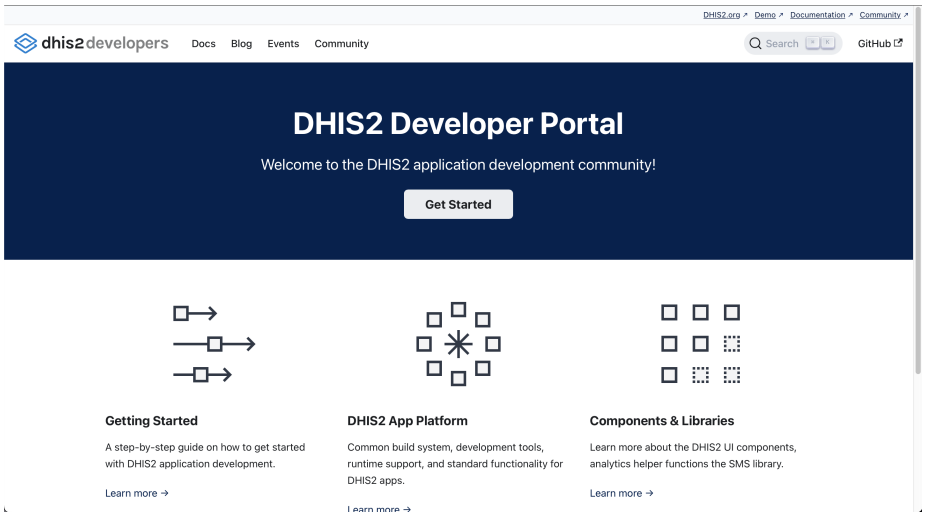
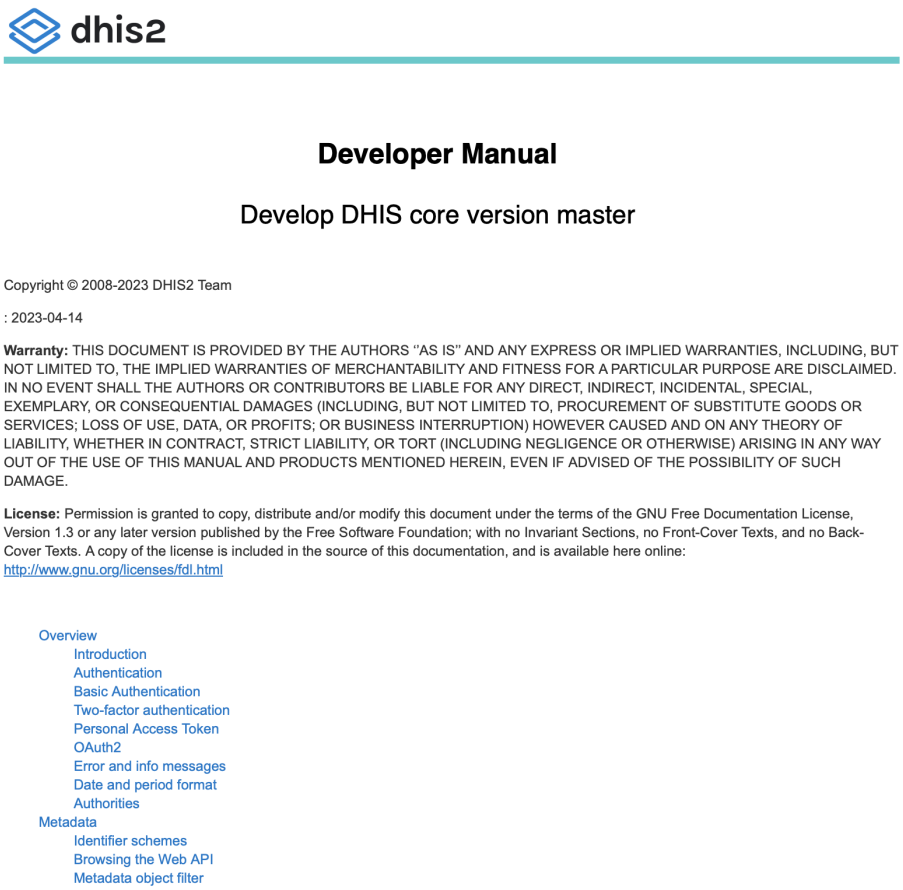


Figure 31: DHIS2 developer portal
DHIS2 Developer portal, displaying information in another design than the developer documentation


dhis2

Developer Manual

Develop DHIS core version master

: 2023-04-14

Figure 32: DHIS2 developer manual
DHIS2 Developer Manual, displaying a third design within the same documentation.

Lack of standardization does not necessarily always result in poor quality, but it was indicated by participants, especially students, that the lack of standardized pages makes navigating the documentation and locating correct resources difficult. As seen in the examples above, the different pages vary in structure, colors and layout, allowing for confusion and misunderstanding for users. Even though they have some of the same design-choices, they are different enough that users may struggle to see the direct connection between them. In the students' case, they tended to rather choose course-specific learning resources like the self-paced online course (UiO, n.d.) in an attempt to solve their problems rather than using the documentation to debug.

Participants also described a lack of clear categorization as a significant obstacle when using the API documentation, as structure in the current developer-

documentation is mainly collected in one page. This strengthens a feeling of information overflow for the lesser experienced participants, making the documentation difficult to navigate. Combined with another mentioned obstacle, that the documentation consists of too much text, the documentation can quickly become hard to comprehend for a new user. Figure 33 presents an example from the DHIS2 documentation where a large amount of text is presented in one page.

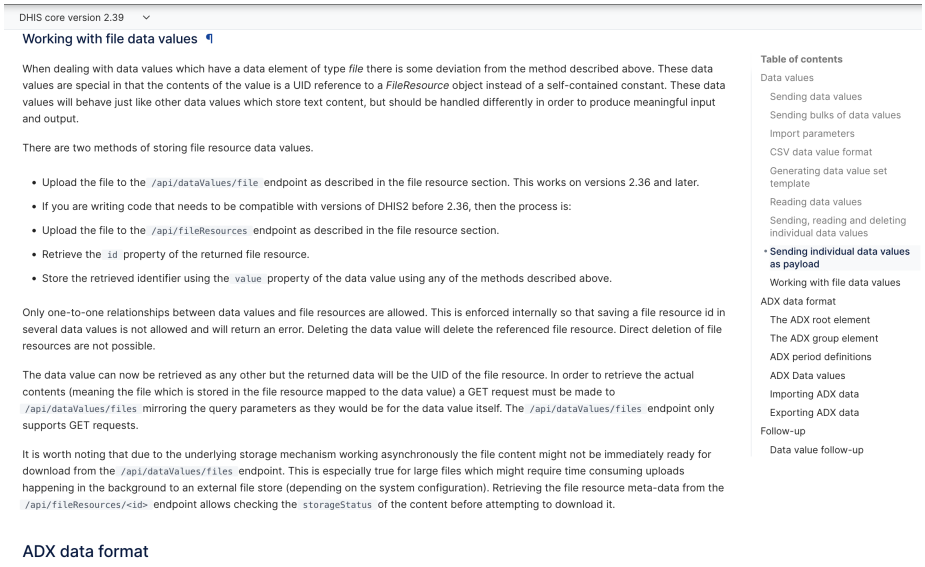


Figure 33: Large amounts of information
A part of the DHIS2 documentation containing a large amount of text

Participants also mentioned internal search-functionality as an obstacle when using the DHIS2 API Documentation. Even though the current documentation has a search-option, the students especially struggled to identify the existence of this, as it was perceived as a part of the overlying page rather than connected to the documentation. Observations of said group also uncovered that using the search-functionality once located turned out not satisfying, as results varied a lot and gave few answers and solutions in use. The developers working with DHIS2 however, did not mention the search-functionality as an issue, possibly indicating that this may be a group-specific obstacle most relevant for students - the less experienced part of the sample group.

Potential for improvement Seeing how the sample group of this iteration consisted of both students, fresh graduates and more experienced developers, an important consideration was identified: how the level of so-called "hand-holding" in the documentation must be adjusted to the target group. Experienced developers and API-users may have different needs than less-experienced,

hence requiring different resources than the less experienced group. While some may require a simple way to quickly search for actions, others need a step-by-step guide describing how to get started and approach different actions and obstacles. As a result, a suggestion from the participants was to structure resources in a way that makes it function both as an encyclopedia, and a more detailed guide, allowing the user to decide what level of "hand-holding" is required.

It was mentioned, especially by the novice users, that large pages that must be searched through should be avoided, in order to ensure understanding. However, as other participants mentioned, more experienced users may appreciate this way of navigating, making a combination of encyclopedia and guide possibly a good solution. A participant mentioned appreciation for the first version of the documentation (v. 1) more than the current documentation, as it resembled a book with its structure. Thus, it was described as having a structure that was easy to follow. Following known structures like this may be a way of adapting documentation to the different users.

When discussing learning of DHIS2, the developers highlighted mastery-focused learning as a purposeful approach. All participants had a somewhat similar experience when starting to work with DHIS2, as they experienced the onboarding as an overwhelming process with a lot of information at a time. Not unlike this, the students' also experienced a large amount of information without a structure allowing them to feel mastery as they proceeded. Participants suggested a structure where theory and practice is taught and connected simultaneously, allowing for personal mastery over time, could help them improve their problem solving at an early stage. They also underlined the value of having the possibility to learn from experience from previous projects. Inspecting previous code served great benefits for both the students and the developers working with DHIS2 in their onboarding-phase.

5.2.3 Iteration summary

This iteration has covered interviews and observations of developers and students working directly with DHIS2. The sample group consisted of five full-time developers, and three students working with DHIS2 in an educational project. Seeing how this iteration was aimed more directly towards DHIS2 by exploring participants working first-hand with the platform, the findings are potentially more accurate for this particular API than findings from the previous iteration. Findings from iteration 2 are summed up in table 8.

Category	**Issue**	**Description**
Content	Inadequate examples	Examples are perceived as inadequate for the user, and are difficult to read due to unknown syntax.
	Unclear information	Information is perceived as inadequate, too theoretical, lacking relevant details and difficult to put into practice.
	No provided information about platform-/core-issues	The documentation lacks clear information about issues and updates.
Organizing	Unclear categorization	The content is perceived as uncategorized, making it difficult to navigate without experience.
	Design not standardized through all parts of documentation	The documentation is made up of several parts, made with different designs, possibly making it difficult to understand relations and navigate.

Table 8: Findings from iteration 2

6 Discussion

Findings presented a large number of categories, covering the different pros and cons with current API documentation. However, to avoid the discussion being too large and wide covering, the findings were compressed through a new round of analysis. From this, the following categories were identified: covered under "Content" are the categories "From documentation to practice", "User-group experience level" and "Continuous documentation". Covered under "Organizing" are the categories "Layout and design of the documentation" and "Categorization and organizing of the content". The categories from findings covered by each of these new categories are presented in table 6 on page 43. Through this chapter, the findings in said categories are discussed against existing literature, to identify similarities and differences between them.

6.1 Content

6.1.1 From documentation to practice

As described in the previous chapter, the interviews clearly revealed the value and importance of having examples available in order to utilize the possibilities of the API. As mentioned by Robillard and DeLine (2010), code examples are essential tools for learning how to use systems in software development, for *"... understanding the purposes of the library, its usage protocols, and its usage contexts"* (McLellan et al., 1998, p. 83). Thus, the sample group of our study emphasizing the importance of having good examples available came as no big surprise. The participants emphasized how being presented how to use the different functionalities in practice can greatly improve the process of learning and understanding how to use the API. However, the usefulness of the examples strongly depend on their quality and adaptability into actual use.

Aghajani et al. (2020) found that faulty and incomplete examples serve as an important issue for usefulness in development. If an example is made in a way that is difficult to understand and adapt, e.g. by presenting examples without clear connections to actual use or by providing incomplete descriptions, the usefulness is significantly lower than an example created to illustrate how to approach a realistic use case. An example displaying the technical functionality of an API-endpoint, but not connecting it to actual intended use, e.g. by providing an example of using the endpoint to achieve a described goal, can make it perceived as little practice oriented, and consequently reduce its usability. On the other hand, formulating use cases reflecting actual issues that users may face, and using the examples to approach these may result in more practice-oriented, thus potentially more usable examples. For an example to be perceived as practice-oriented, it should present and be connected to use cases that represent situations it is likely that developers will face during development with a system.

As Uddin and Robillard (2015) found, poor quality in descriptions and exam-

ples serve as a major problem for efficiency when using documentation. Like our findings, they found that insufficient explanations can lead to the documentation being of less use than optimally, thus not serving its intended purpose. In a complex structure such as DHIS2, sufficient understanding of the use cases, as well as how things are connected can be important for understanding and usability of the information. Wyner and Lubin (2011) illustrates how a documentation containing precise, practice oriented descriptions and examples can make an API usable for developers regardless of previous experience. Findings from our study indicate that the lack of said practice-oriented focus in the documentation can have a negative impact on the learning and understanding for novice users, possibly reducing the usefulness for users without much experience.

Presenting how to use the API with actual use cases is an important finding from the data collections for our study. If the user is not able to understand how to utilize the API in their project, it is likely that they will either 1) struggle finding out how to use the API, strongly affecting both productivity and satisfaction negatively, and/or 2) avoid using the API at all. Especially the participating students in our study expressed lacking use cases and the documentation appearing not practice-oriented as a major struggle for their understanding and progression using DHIS2 in projects. Robillard and DeLine (2010) discuss the issue of matching APIs with scenarios, or use cases, where their findings indicate that documentation without a clear connection between use case-scenarios and the documentation can cause confusion when trying to learn and utilize the API. Like the findings from our study, they found that users often want a clear connection between use cases and the API displayed. The lack of relevant use cases can cause the user to be confused regarding the possibilities, as well as how to perform the correct actions to successfully achieve different goals. As a result, the documentation can be perceived as difficult to use, negatively affecting the development process for users regarding both time and effort.

Another important mention was the possible benefits having access to interactive examples can have, e.g. through a "sandbox" where you can play around with a technology and try different combinations of features (Arntzen et al., 2019). This can improve the adaptability and usability of the endpoint documentation, by allowing developers to test endpoints directly in the sandbox-environment, instead of having to implement the API call in a program first to be able to test it. Zinovieva et al. (2021) describes how what they mention as "online coding platforms" can be used to improve remote learning for educational institutions. They state that interactive learning contributes to cognitive activity, positively affecting the educational quality. Like the case of their study, API learning is generally a remote process, where one is often required to approach the documentation to gain insight and understanding. It is therefore likely that the findings of Zinovieva et al. can be relevant also for novice developers approaching new APIs, as the process of approaching new learning material remotely can be seen as similar to using documentation to understand new APIs. However, interactivity in examples has not been highlighted as an area of importance

in any of the studies exploring API documentation for experienced professionals (Aghajani et al., 2020; Robillard and DeLine, 2010; Uddin and Robillard, 2015), indicating that this may be most relevant for the less experienced user group.

As an extension of the previous section, a possible solution can be to provide step-by-step descriptions and examples, presenting a scenario in a way that's easy to replicate in order to achieve the same goal. As the findings from our study show, trial and error is a common way to approach a problem, by testing different parameters and combinations of requests in order to reach a goal. Jones et al. (2010) discusses how trial and error can be an effective way of approaching problems with limited preliminary knowledge about the given scenario. Thus can trial and error be a natural way of approaching problem solving when learning to use an API. However, this approach is not necessarily the most effective, and can possibly lead to a less efficient development process as a whole. With relevant use cases, designers of documentation can present different actions and functionalities in practice for cases developers are likely to face, displaying the different steps that need to be performed in order to reach different goals efficiently. Trial and error will still be a practical approach to adapt the examples into the developers own project, but describing the steps one by one may simplify the process initially and reduce the need for comprehensive trial and error-exploration for basic actions (e.g. getting access, examples of fetching different data etc). A possible way to present step-by-step how to fetch data from a dataset can look something like presented in figure 34.

6.1.2 User-group experience level

An important thing to consider when designing documentation, is what the purpose of the documentation is supposed to be, and what experience level the intended target group possesses. Designers must decide whether the purpose of the documentation is for it to serve as a guide aimed at describing use step-by-step or as an encyclopedia where developers can quickly access information needed to solve a task. Garousi et al. (2013) suggest that the needs of the less experienced users should be considered when designing documentation. If less experienced users are able to understand and make use of the documentation, it is likely that it also functions for developers with more experience. However, as found by Uddin and Robillard (2015), information aimed at and designed for novice users may reduce the efficiency of more experienced users. Thus, what's considered an improvement for some may be seen as a deterioration for others. Participants proposed splitting the documentation as a possible solution. Seeing how our findings, as well as previous research indicate that experienced users may value searching through documentation e.g. by using "CTRL-F", a design partially focused on supporting this may be beneficial. Meanwhile, another part of the documentation can be aimed at novice users, containing more of a step-by-step approach and page-separated categories to ensure learning also for those with little preliminary knowledge. This way,

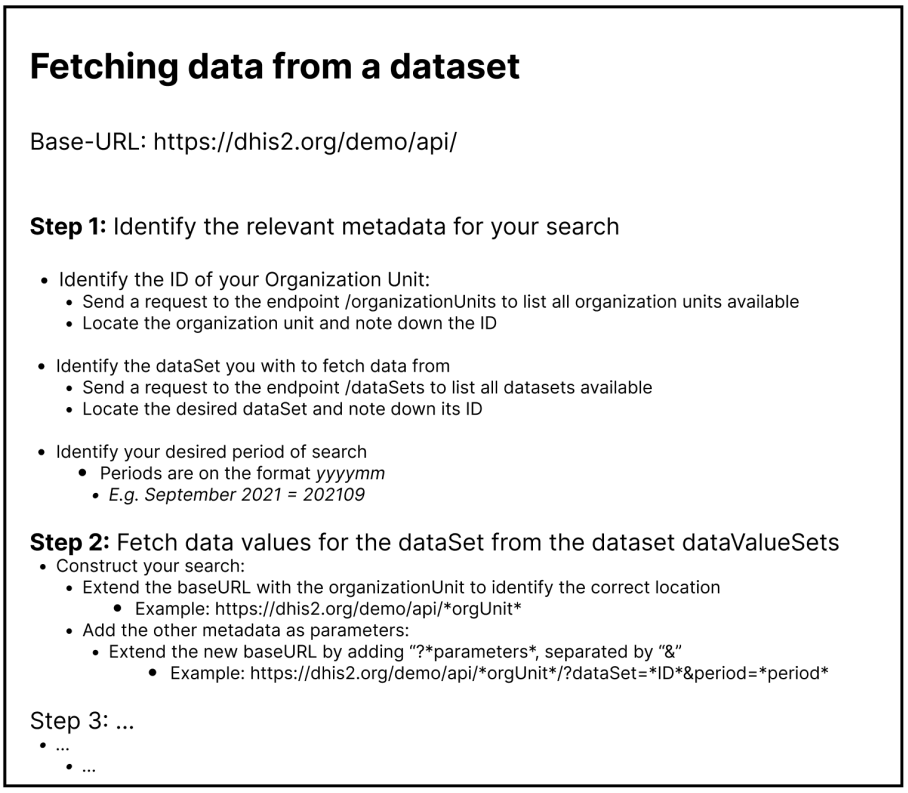


Figure 34: Step-by-step descriptions
Example of how a step-by-step-guide for an interaction with the DHIS2 Web API can be presented. Note that the information presented is purely for illustrative purposes, and not necessarily accurate.

it is possible to increase usability for both ends of the "experience-specter", possibly maintaining what's considered good for some users while improving what's considered lacking by others. Providing a detailed introduction to use for some endpoints and functionalities can lead to an easier understanding of how the API works, thus making use of the rest of the documentation easier.

Findings from this study shows that all participants that have or have had direct experience with the DHIS2 Web API, acknowledge that the system has quite a steep learning curve, and is not perceived as fitted to developers with less experience, but rather more experienced developers. Thus, knowledge boundary resources like the documentation etc. are perceived as difficult to use for novice developers. Also the participants without direct knowledge of DHIS2 agree that level of experience is important to consider when producing API documentation, as experience can strongly affect the understanding and abil-

ity to comprehend technical information. The findings of Wyner and Lubin (2011), regarding how an API with thought-out, targeted documentation can be successfully used by users with very limited technical experience, indicates that considering the experience-level of the target group can be of great importance when designing documentation for it to be accepted by the user group. This idea is strengthened by Aghajani et al. (2020), who states that creators of documentation should *"... always keep in mind the actual documentation users and their needs..."*(p.8). Seeing how DHIS2 is an open-source platform aimed at improving public health monitoring, it may be beneficial to consider all experience-levels, allowing more than only developers with long experience to contribute to application development.

A major issue regarding documentation, also in the case of DHIS2, is that information provided is perceived as unclear, thus not helping users sufficiently to understand. As found by several previous studies (Aghajani et al., 2020; Uddin and Robillard, 2015), unclear and incomplete information serve as a significant problem when using documentation to understand and use APIs efficiently. Participants noted that information often can be perceived as inadequate for their understanding, as it lacks relevant information in order for them to effectively understand the descriptions provided. Thus, they perceive the documentation as lacking sufficient usability. As described by Shackel (1981), usability can be understood as *"... the capability to be used by humans easily and effectively"* (p. 24). Seeing this in the context of the user-oriented view (Bevan et al., 1991), one can argue that documentation with descriptions perceived as unclear does not have sufficient usability.

Large amounts of information at a time, exceeding cognitive processing limits, may lead to poor decision making and worse performance (Malhotra, 1982). Large amounts of text can also make it difficult to distinguish between what's relevant and what's not. (Moy et al., 2018). Especially for newcomers and novice users, large amounts of information can make content more difficult to comprehend, thus making usage of the API more difficult. To improve usability of the documentation, enhancing use of the API for novice users, documentation designers should strive to keep descriptions short and precise. Short, however, is a subjective opinion, making it difficult to define what's too short or too long. It is therefore important for the documentation-designer to consider what information is relevant to include and what is excess for each specific case, in order to avoid making descriptions perceived as too long, while still including enough information for the user to be able to make use of it.

Making content aimed at being accessible and perceived as clear, however, is not necessarily as easy as it may sound. Furnas et al. (1987) discusses what they refer to as "the vocabulary problem". They state that *"many, many alternative access words are needed for users to get what they want from large and complex systems."* (p. 971), referring to how choice of words can affect the availability of information. Even though this study primarily focuses on words for accessing functionality, it can also be seen in the context of understanding descriptions as

a result of choice of words. Even though technical documentation such as API-documentation must be expected to contain theoretical phrases, the vocabulary problem displays how the choice of words can make information difficult to locate and use without preliminary knowledge of system-specific concepts and terms. One participant in our study mentioned that it could be beneficial to provide an "appendix", or separate resource providing descriptions of important keywords, as well as a short description of the context(s) in which these are used. By doing this, the designers of documentation can eliminate some confusion and failure due to the vocabulary problem, possibly increasing the understanding, accessibility and usability of the documentation.

As mentioned by several participants, the data structure of DHIS2 is perceived as complex and can be difficult to comprehend. They suggested that use of visual elements like diagrams and models can increase the learnability of the system. Students can remember information better when visual elements are utilized (Raiyn, 2016). Thus, visually presenting the data structure and relations within DHIS2 can improve the understanding for novice users. As several participants mentioned through the interviews, efficient usage of the DHIS2 API was first possible when they understood the underlying structure. Therefore, ensuring efficient learning of this as quickly as possible can greatly improve the development process for the target group. Using models like the example presented in figure 35, understanding relations can be perceived as easier than only textual descriptions, thus possibly making development using DHIS2 easier and more efficient. However, in a complex structure like DHIS2, visualizations of data structures and relations can quickly become as complex as the structure they are supposed to describe. An example of this can be seen in figure 36, presenting an overall visualization of the DHIS2 data model on a higher, more abstract level. It is therefore important to consider the aspect of simplicity when designing models and diagrams to ensure that they actually improve the understanding of what they are presenting.

However, both our participants and the respondents of Uddin and Robillard's (2015) study also indicated that too much information can create difficulties locating the relevant and important information. Participants noted that large chunks of text can create issues understanding the content, as well as being able to perceive what's relevant. Thus can too detailed descriptions of context and relations make the documentation be perceived as bloat, working against its purpose. It is therefore important to find a trade-off between having thorough descriptions, and keeping them short to avoid cognitive issues.

Participants also mentioned that a possibility for the community to contribute may be beneficial. Especially in the case of DHIS2, the community is an important resource, serving as a forum for QAs when developers face obstacles. It was noted that the community might have established best practices that could be beneficial to include in the documentation, supplementing the underlying technical information. The community may have different motivations for contributing, including content contribution, motivated by the need for self-

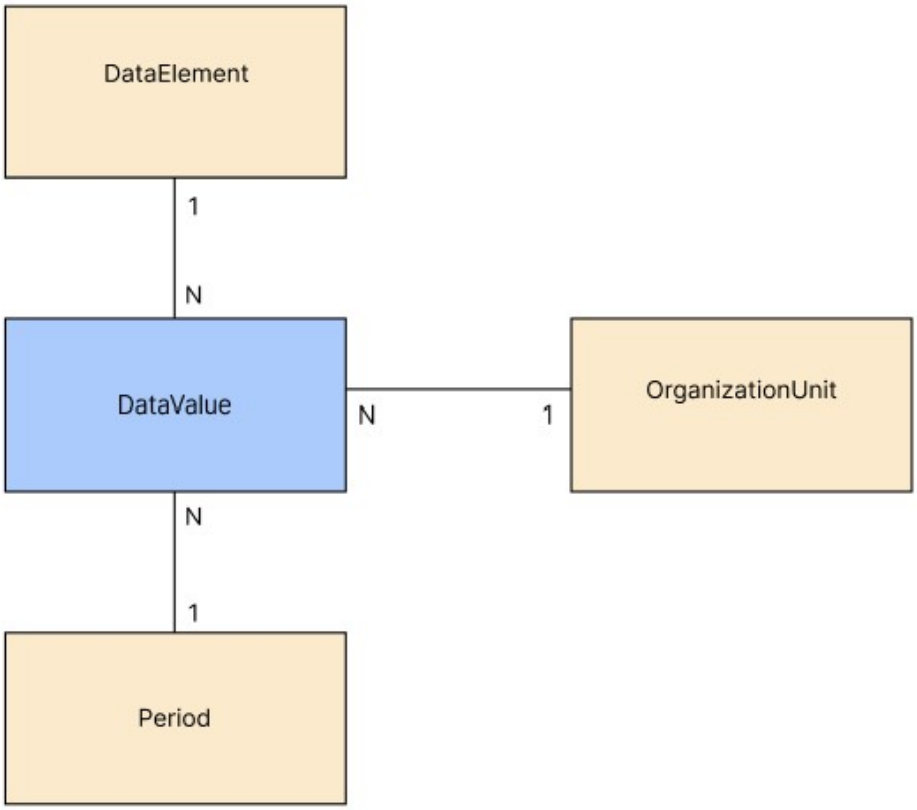


Figure 35: Simple visualization of DHIS2 data model
Note: "The data model" from Self-paced online course (UiO, n.d.)

development and reciprocity, and community participation, motivated by altruism and a sense of belonging (Xu and Li, 2015). Through the interviews for our study, the DHIS2 community was mentioned on several occasions, underlining its importance in the development process of DHIS2 applications. Thus, it is possible that an opportunity for community contribution in the DHIS2 API documentation could simplify the process of debugging and problem solving by allowing developers to add their experiences and best-practices directly into the documentation. This way, developers can benefit from others' experience if, or before encountering similar problems. The existing community FAQ-forums for DHIS2 displays that there is indeed motivation present for community contribution, and an integrated possibility for this directly in the API documentation may be beneficial for sharing of knowledge. The GitHub-API provides a good-practices-page, where developers can see what the API owners perceive as good practices when developing. Figure 37 presents this page from the GitHub API documentation. By having a section like this, with the opportunity for the com-

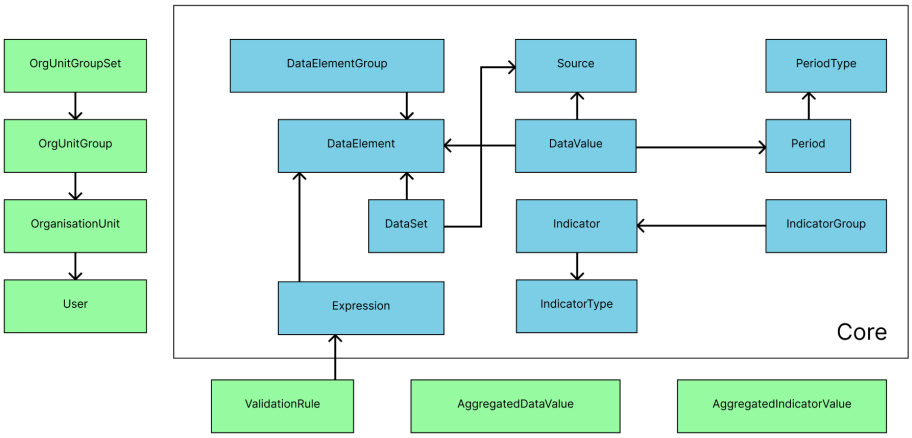


Figure 36: Visualization of DHIS2 data model on a higher level
Note: "Core Diagram" from the DHIS2 documentation. "4.5 The Data Model", n.d.)

munity to contribute as they discover new best-practices, the user group as a whole can benefit from others' experiences. This can be especially useful for novice developers, as learning best practices and procedures from experiences of more experienced developers can increase their understanding of how things are done, thus making it easier for them to develop themselves in future projects.

A possible approach to this is to integrate what was mentioned in interviews as a "Wikipedia" approach, where users can suggest improvements and additions to the different sections of the documentation. This way, the user community can propose changes and improvements they see fit, while the original designers of the documentation can still control what changes are accepted and which are rejected. GitHub has integrated this as a part of their documentation, allowing users to create "pull requests" with suggestions to the documentation who is open source. Figure 38 displays how this possibility is presented in their documentation.

6.1.3 Continuous documentation

Regardless of the system, insufficient handling of version updates causes a major problem for developers. Previous studies have concluded that poor documentation is the primary reason for quick quality degradation and aging of software (Kajko-Mattsson, 2005, p. 31). Participating students from our sample group mentioned how they on several occasions while working with DHIS2 found outdated, no longer functioning parts of the documentation together with still working resources. Aghajani et al. (2020) found in their study that 69% of their respondents considered up-to-dateness of documentation an important

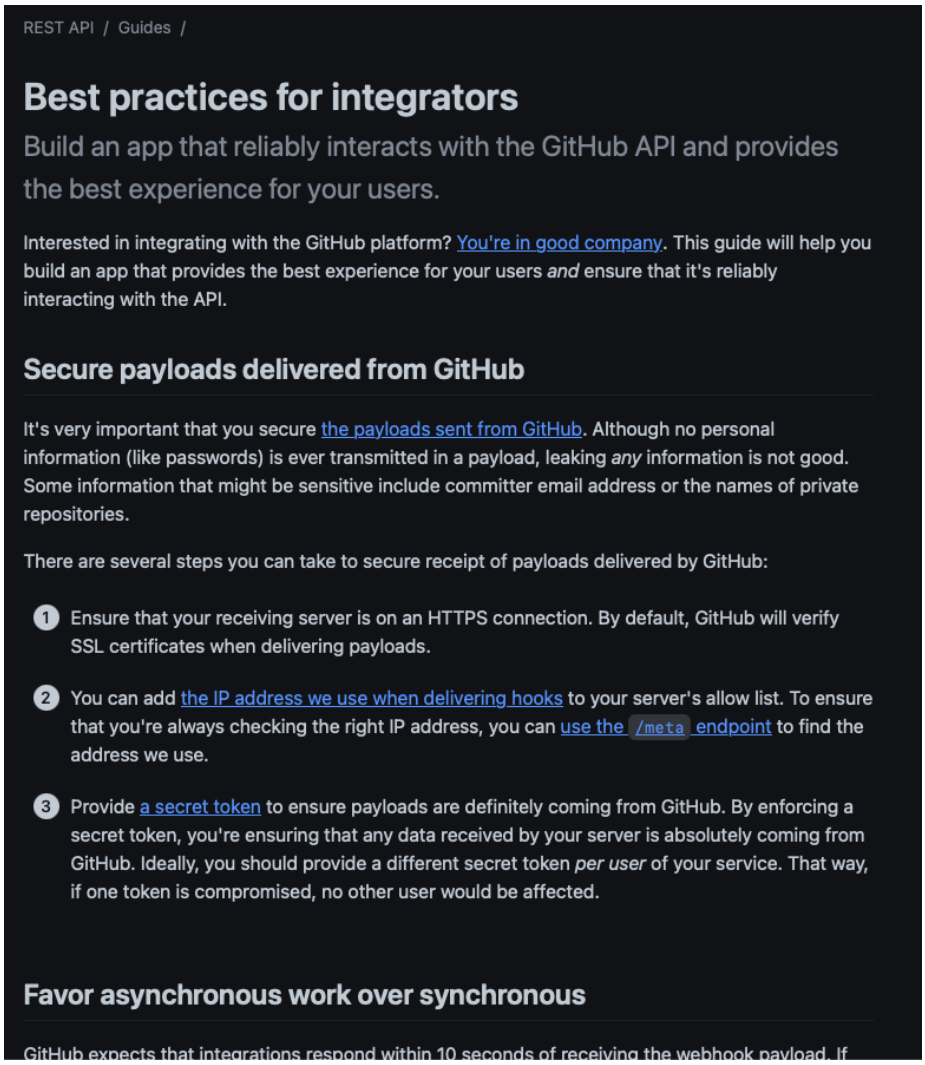


Figure 37: GitHub API documentation - best practices-page

issue. One participant in our study highlighted the Android SDK as an extreme case of this issue, with several updates a year suddenly leaving functionality outdated without proper notice or information. Figure 39 displays the evolution of Android versions and API levels since 2018, displaying how the API evolves continuously. This can make it difficult to know what version is currently supported when developing, and maintain existing applications as these need to be adjusted according to the changes in the API they communicate towards. Keeping the documentation up to date with system-version updates is therefore crucial in order to ensure usability for users.

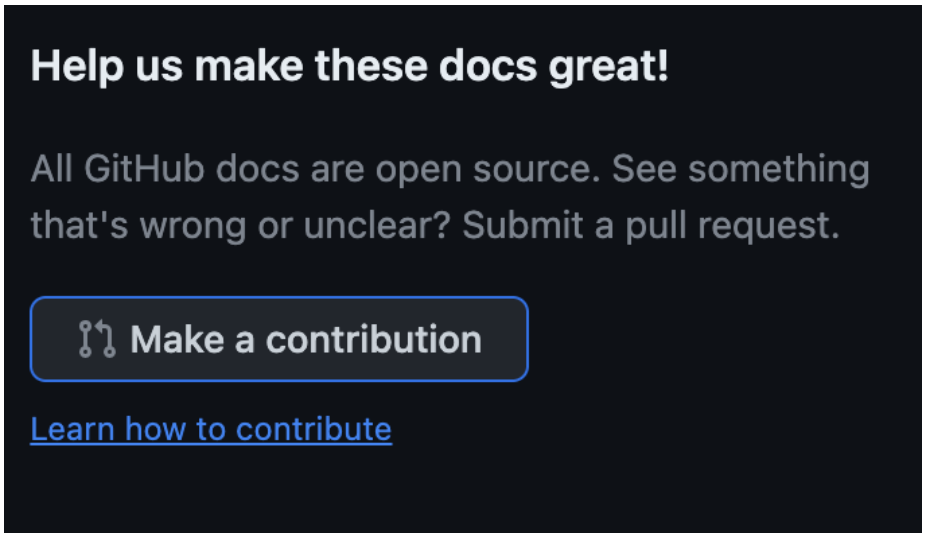


Figure 38: GitHub API documentation - possibility of community contribution

Version	SDK / API level	Version code	Codename	Cumulative usage [1]	Year
Android 14 DEV	Level 34			—	*TBD*
Android 13	Level 33	TIRAMISU	Tiramisu [2]	11.8%	2022
Android 12	Level 32 Android 12L	S_V2	Snow Cone [2]	36.9%	
	Level 31 Android 12	S			2021
	▪ `targetSdk` must be 31+ for new apps and app updates. ▪ `targetSdk` will need to be 31+ for app updates by Nov 2022 and *all existing apps* by Nov 2023. [3]				
Android 11	Level 30	R	Red Velvet Cake [2]	60.0%	2020
	▪ `targetSdk` must be 30+ for app updates, and new WearOS apps. ▪ `targetSdk` will need to be 30+ for *all existing apps* by November 2022. [3]				
Android 10	Level 29	Q	Quince Tart [2]	77.7%	2019
Android 9	Level 28	P	Pie	86.3%	2018
	▪ `targetSdk` must be 28+ for Wear OS app updates.				

Figure 39: Android API evolution since 2018
Android versions and API-levels (compatibility) since 2018, illustrating how APIs rapidly evolve. Note that each API level also includes updates within itself, increasing the amount of updates beyond what's illustrated by the amount of levels.

The participants also mentioned that lacking information about system-issues can serve as a significant obstacle when developing. Server-problems, system-maintenance and similar issues should be clearly communicated from the API-administrator, so that developers are aware of issues, changes and problems with the platform. Participants especially noted that some kind of notice of major server problems would be beneficial, as server-downtime was never noti-

fied clearly from the DHIS2 platform core. The GitHub API-documentation is described as an example of how this can be solved (Bush, 2019), as they provide small widgets indicating the current status of the API on every page of the documentation. This way, developers can always quickly check whether problems they face are due to server-problems or are programming-related. Other ways of integrating this can e.g. be by adding sections in the page-header or top of the page informing about status (OK/Downtime + why).

6.2 Organizing

6.2.1 Layout and design of the documentation

A significant struggle for participants trying to explore the DHIS2 documentation, was the lack of standardization through the different sections of the documentation. Seeing how the documentation is structured with a separate developer guide, UI-documentation, developer documentation etc., users may struggle to navigate without proper consistency in layout and design of the documentation. Consistency is an important design principle for usability, allowing users to use a familiar interface throughout the whole experience of a product, enhancing learning and understanding, and reducing the insecurity for users whether the information means the same thing (Lidwell et al., 2003; Preece et al., 2019; Schlatter and Levinson, 2013). Without said principle present, users may find it more difficult using resources provided, as they do not necessarily resemble one another. E.g., the participating students expressed that the difference in design of the developer documentation and the UI documentation made it appear as documentation for two separate systems. They suggested that a consistent design through all parts of the documentation for DHIS2 would both help them navigate easier, and make it appear more as one system, rather than several separate. The issue and obstacle that lack of standardization creates has been observed across levels of experience, hence indicating that it is not necessarily a matter of experience (Aghajani et al., 2019). However, consistency is not always easy to achieve: when trying to design an interface to be consistent with something, one faces the risk of making it inconsistent with something else (Preece et al., 2019). It is therefore important to consider the design as a whole to make it familiar through all parts.

Participants suggested that documentation-designers should look to large, established, public APIs when designing documentation (examples mentioned in data collections included Twitter and Dropbox). Figures 40 and 41 displays the mentioned APIs' layout of presenting relevant information for using an endpoint. These are exposed to large amounts of users on a daily basis, generating feedback quickly regarding issues and weaknesses in the documentation. By getting inspiration from these organizations' API-documentation, designers of documentation can accomplish a design accepted by the users with less effort than if they were to create it from scratch. Thus, it is likely that users will be satisfied, possibly perceiving the documentation as usable with less effort for the designers. Seeing how documentation usually is written as the last step of

a development process, often suffering from reduced motivation (Parnas, 2010), creating the documentation with inspiration from established organizations can lead to good results with reduced effort.

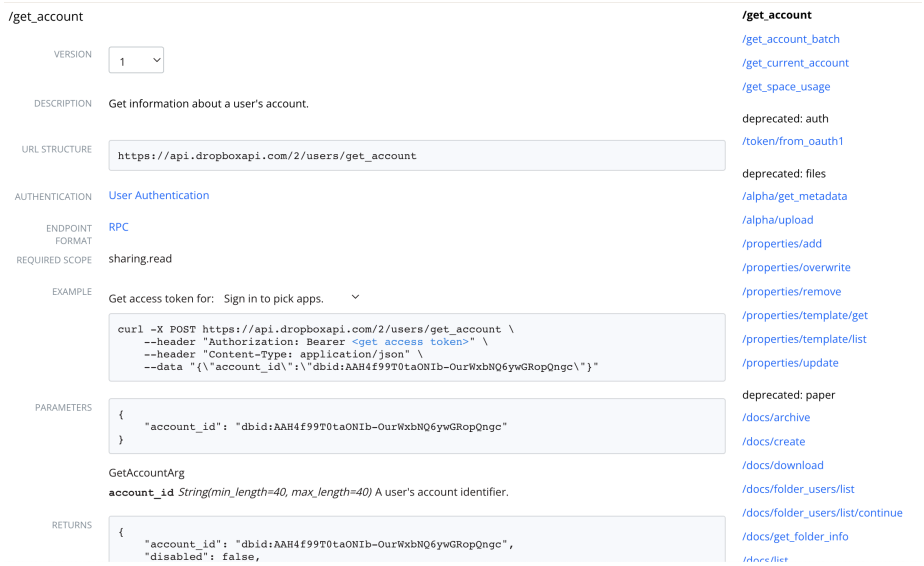


Figure 40: Dropbox API documentation

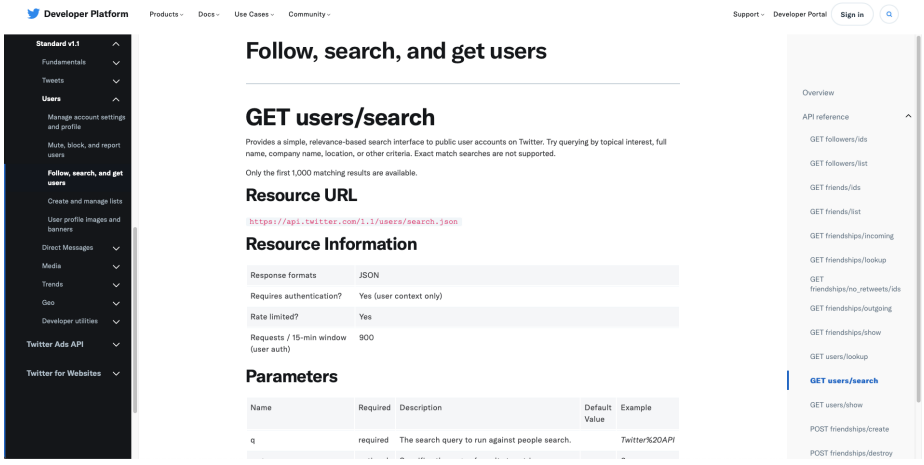


Figure 41: Twitter API documentation

While the target group of this study is focused on novice developers, Uddin and Robillard (2015) studied how a target group with significantly more expe-

rience perceived API documentation. While some findings are similar, they are also different in some areas, e.g. view on modularization (which is appreciated by novices but seen as an obstacle and irritation by experienced professionals), which indicates that different expertise-levels have different needs. This was further strengthened by the more experienced participants of our study, as they described their own way of working as fitting to a one-page encyclopedia-style documentation design with less comprehensive descriptions, while still recognizing that novice users may require more thorough and "hand-holding", guide-style kind of documentation. Garousi et al. (2013) found that there is a significant difference in documentation usage between experience levels. They found that experienced practitioners refer less to documentation, suggesting that needs of the less experienced should be considered when creating documentation. The perception by the participants of this study regarding the current DHIS2-documentation indicates that they experience it as aimed more towards professionals with long experience than covering the spectrum from beginner to professional.

6.2.2 Categorization and organization of content

An important note that was mentioned by the participating students, as well as some participating developers, was the lack of clear categorization throughout the DHIS2 API documentation. Without categorization, they perceived the information as one large chunk of text, and expressed difficulties separating the different things from each other, and gathering the related information. They suggested that a clear categorization of things that are related would help them see what's similar and not, thus making it easier to gather relevant information for the use case. Aghajani et al. (2019, 2020) found that findability and information organization were considered important for a large number of the participants in their studies. Like the participants in our study, they found that a logical structure with related information gathered together can help understanding the documentation easier, and make more use of the documentation. What's considered a "logical structure" is a highly subjective opinion, and can vary depending on experience level and personal preferences. However, clear connections between related functionalities and categorization grouping functionalities used together for different use cases is likely to give a feeling of logic in the structure. Categorizing after common areas of use, e.g. endpoints used for fetching different data about users etc. being collected can possibly improve the usability of the documentation for novice users, simplifying the ease of use.

However, previous studies show that not all users agree on this. Uddin and Robillard (2015) explored common irritations and obstacles for developers with long experience (mean $\approx$ 13 years experience), finding that the second most mentioned irritation regarding presentation of information was fragmentation: that information is separated across several different pages or sections, indicating that they might rather want a structure with everything collected at one

place. Thus, it can seem like there is a difference between what the users with different experience perceive as useful and efficient when developing. This difference has also been identified in other literature (Garousi et al., 2013). This underlines the importance of considering the user group when designing documentation, and not necessarily trying to make a one-size-fits-all design in an attempt to find a quick way of dealing with all users. Whether this issue is solved by separating content into separate pages, compressing related information into smaller portions, reorganizing the content or a combination of these depends on the primary user group in target for the documentation.

Rather than to read thoroughly, people tend to scan pages when they read online (Moran, 2020). Even though some follow the commitment pattern, observations and interviews from our study describe developers as more likely to follow the layer-cake scanning pattern, as seen in figure 42 (Pernice, 2019). It is therefore important to prioritize and organize the documentation after relevance, so that the most important information is presented first. This way, developers may find it easier to locate the information relevant for understanding the use of the API and solving their problems, possibly improving the usability of the documentation.

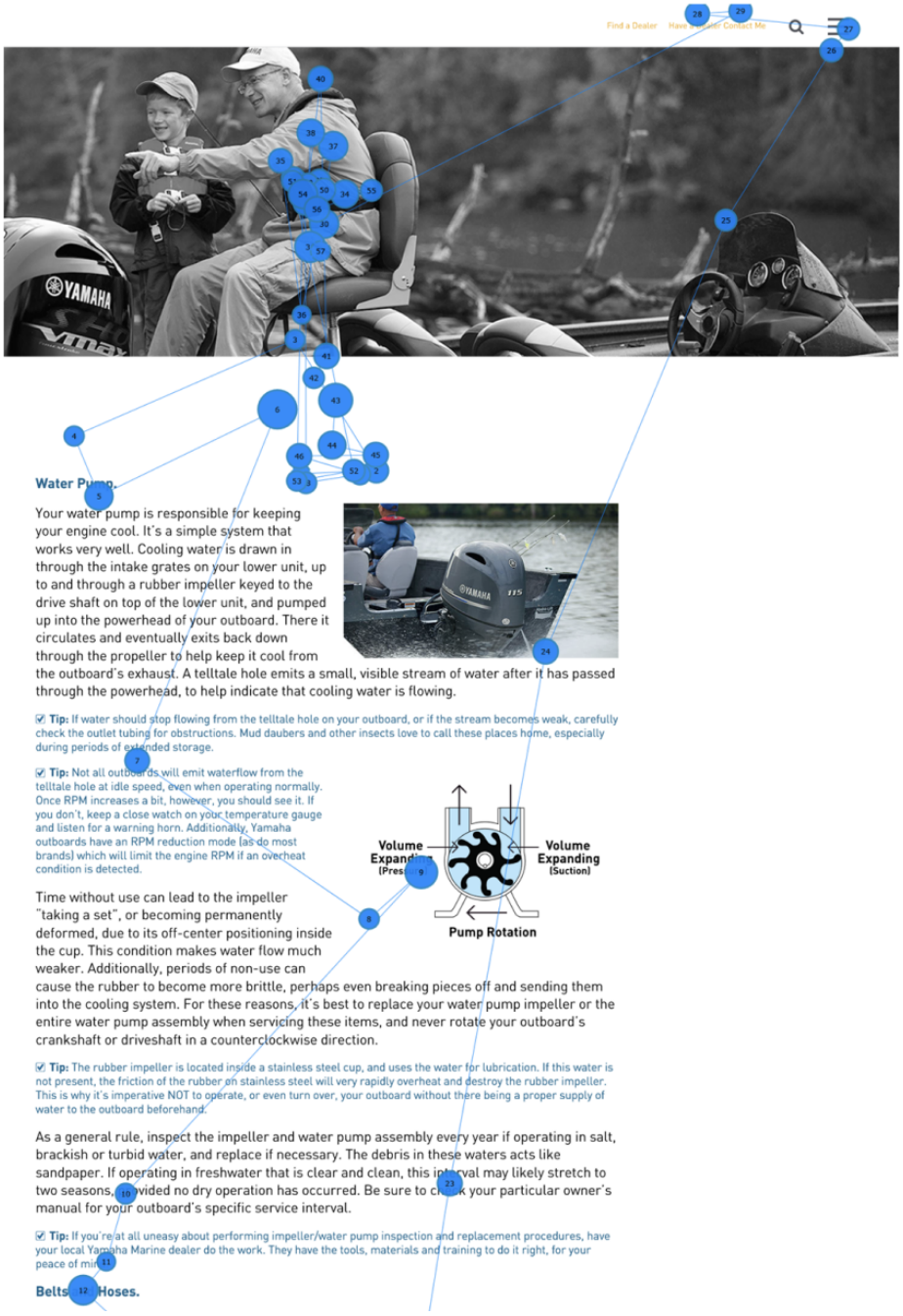


Figure 42: The layer-cake reading pattern

7 Conclusion

In this chapter, the discussion is summed up, and connected to the research questions. First, the sub-questions formulated in the introduction-chapter are addressed (RQ1.1-1.5), and connected to relevant results from the discussion. After this, the main research question is addressed (RQ1), with basis in the answers to the sub-questions, and recommendations to practice are presented. Then, the contribution to practice and research is presented, before a brief reflection on further research and limitations is conducted.

7.1 Research questions

RQ 1.1: What are common factors that make existing documentation difficult to understand?
The findings from this study shows that lack of use cases and practice-orientation in examples and descriptions serves as a major issue for understanding information. Presenting theoretical information without connecting it to possibilities of actual use, can result in the information being difficult to adapt and put into use in projects, reducing the value of the documentation for the users without much experience. Issues with layout was also found to be a significant issue, especially in the case of the documentation for the DHIS2 Web API, as participants struggled to locate relevant information and parts of the documentation.

RQ 1.2: What are common factors that make existing documentation easy to understand?
Users value simplicity in the documentation. Simplicity can be difficult to describe accurately, as it is indeed quite a subjective opinion. What is meant in this case is that descriptions contain relevant, short and to-the-point information regarding relevant details, without information not directly relevant to what's being described. Additional information can be included, but should be put somewhere else than the main description, e.g. further down the page as additional information to support if needed.

RQ 1.3 What content is necessary in the documentation?
Information about what is required to communicate with the API-endpoints (e.g. authentication keys etc), as well as information about what is possible to provide to the different endpoints, is important information to provide in the documentation in order for users to be able to use the API. Information about what the user can expect to get in return is also important for the user's understanding of the endpoints. This information is essential for using and understanding the different endpoints. Without information regarding what is required and possible to provide to the endpoint, and what it is supposed to return, its core functionality remains unknown.

For initial use of an API, information regarding how to get authentication and getting access to the API is important. Even though processes like authentication often are similar between APIs, this can require some degree of experience

to be perceived as intuitive. Clear descriptions of how this is done is therefore important for users to gain access to an API and start using it.

Relations and connections between components in the internal structure of the API may be important to understand in order to utilize the API to its full intent. Especially in the case of complex data structures like DHIS2, understanding the underlying relations can be important. This can however be difficult to perceive by reading text-only descriptions. Visualizations like diagrams and figures can help the user understand the information easier, thus serving as important tools for the understanding of the overall system.

Lastly, information about version updates are important to provide, in order for users to follow the continuous evolution of the API, and stay updated on what changes are being made to the endpoints.

RQ 1.4: What content is perceived as excess in documentation?
Findings identify information about external resources (e.g. frameworks (React etc.), REST APIs etc.) included in the documentation as excess. Whether it is needed information or if it can be assumed that the user already possesses knowledge about the resources strongly depends on the target user group experience level. Thus, to encompass all users, this information should not necessarily be removed fully, but rather be linked to instead of using time and effort describing the concepts in detail in the documentation. This way, the users that require additional knowledge regarding relevant external resources can easily find and navigate to these, while it does not fill the documentation with excess information for those not needing it. Too detailed descriptions of concepts can be perceived as excess in documentation. While it is important to provide enough information for the user to understand the concepts, too much information can work against its purpose. It is therefore important to decide what is important and less important information for descriptions.

RQ 1.5: How should the information be organized?
An important aspect for easy usage of the documentation is regarding organization and presentation of data. Visible information, information that is presented to users upon opening a page, should be reduced so that only the important information regarding the endpoint is displayed initially, and what's more in the category of "nice to know" or less important information can be hidden, e.g. in collapsible sections. This way, it is possible to reduce the amount of information visible to the user, thus allowing for easier location of the most relevant information. If more in-depth, additional information is needed, this can still be available, but not necessarily visible by default to reduce the feeling of information overflow for the user.

Exactly how information should be organized, depends on the purpose of the documentation, whether it is to function as a guide or an encyclopedia for the user. If the purpose is for the documentation to serve as a guide to using the API, a more modularized organization of the information can be beneficial, separating the different actions being described into separate sections. However, if the

purpose of the documentation is for it to function only as an encyclopedia, a supporting resource for developers while using the API in development projects, modularizing and separating the content into separate pages can cause more problems than it solves. A possible solution to this is to combine the two approaches, by having both a guide-section with certain actions, e.g. getting access and getting started using the API, presented step-by-step isolated from other content, as well as a section functioning more as an encyclopedia where users are able to quickly locate what they're looking for. This eliminates the need to browse through content that is arranged for a different experience level and need from your own, yet still allows both first time- and novice users, as well as more experienced users, to find resources that suit them. Table 9 presents some possible circumstances where the two approaches may be suitable isolated.

Guide-approach	Encyclopedia-approach
The guide-approach can be suitable for situations where it is likely that users will need or appreciate a thorough, "hand-holding"-approach introduction to a system. If a system includes complicated logic, intricate procedures or difficult combinations of functions to reach goals, a guide-approach can help ensure that users are guided through the necessary steps to succeed initially. The guide-approach can also be suitable for situations where users with little experience are likely to be exposed to the system, as presenting actions in the relevant steps does not require preliminary knowledge regarding the technology.	The encyclopedia-approach can be suitable for situations where users are likely to use the documentation for quick searches for how to use certain functionalities, or to find information regarding related properties and characteristics. As an encyclopedia-design can quickly grow large depending on the system complexity and available endpoints, thus possibly difficult to navigate, the approach may be more suitable for users with some degree of preliminary knowledge, or experience with the system from beforehand.

Table 9: Suggested suitable uses for the guide- and encyclopedia-approach

Regardless of purpose, an intuitive and logical structure will require a categorization of content that connects related and relevant resources together. Depending on the system, related and relevant resources can be resources that are often used together, resources that lay the foundation for use of others, resources that strengthen understanding etc. Without this, users may experience confusion regarding what endpoints are relevant for use together, thus making the overall usage more difficult to comprehend.

7.2 Recommendations for practice

RQ 1: How can API-documentation be designed to facilitate learning and development for novice developers?

Summing up the discussion, to address the main research question of the thesis, six areas of importance that can be beneficial to consider when creating and designing API-documentation for novice API users are identified and presented below. These can be taken into consideration by platform owners and -maintainers, as well as other documentation designers to improve the usability of API documentation for novice developers, while still maintaining the usability for experienced users.

1. *Consider combining a guide- and encyclopedia-approach*
   To embrace all users, a possible solution may be to create a combination of a guide and an encyclopedia, so that the users can decide what they need to make progress from the stage of the development process that they are located in at the time of accessing the documentation. This way, novice users who need a thorough description of each step to reach a goal are able to find and make use of that. Meanwhile, more experienced users can find what they are looking for to progress without necessarily having to go through a guide-structure, e.g. if they are only looking for information about a specific endpoint.

2. *Connect examples to use cases to explain how they are used in practice*
   Examples are important for the user to be able to see how the system can be used. They are therefore important to include for the understanding of usage when users utilize the documentation to include the API in their development. However, examples should be connected to realistic use cases in order for users to easily understand how to adapt the endpoints into use. Practice-oriented use cases can ensure understandability and usability, thus making learning and utilizing APIs possible during the span of a project. It can therefore be important to formulate relevant use cases based on expected scenarios of use for the API, so that the use of each endpoint is seen in the context of actual cases that the users are likely to face.

3. *Make examples interactive*
   For novice users, having a way to test the different endpoints without necessarily having to implement them into projects initially can be beneficial. Making examples interactive can therefore be beneficial to create an arena for new users to explore an API. This way, they are able to see how different parameters can affect the response, and create a general request that can be adapted into their project when it is to be implemented.

4. *Keep descriptions short and precise*
   To avoid users feeling that the documentation provides an overflow of information, it is important to keep descriptions of endpoints short and

precise, and avoid explaining endpoints in too much detail initially. To achieve this, it is important to identify what information is relevant for understanding and to start using the endpoint, and what information can be hidden until the user decides to display it, e.g. in a collapsible section. Descriptions can also be elaborated by the use of visual elements (figures, diagrams etc.) to make understanding of concepts, relations and data structures easier for the users, while still maintaining simplicity and avoiding long, overwhelming textual descriptions.

5. *Identify and prioritize presenting the most relevant information first*
   An important factor for documentation to be perceived as usable, is the ability for the user to quickly and intuitively locate relevant resources for supporting their development. It is therefore important for designers of documentation to identify the most important and relevant information for each endpoint, and present the information accordingly after importance. This way, it can be easier for new users to locate what information is most relevant and important for the current endpoint, simplifying the process of beginning to use it. Following this, additional information can be presented to elaborate, without taking the initial focus from the user when entering different sections of the documentation. Avoid presenting information and concepts regarding external resources (e.g. REST APIs etc.) in the documentation. Link to external resources explaining relevant concepts instead of including the information written out in own documentation.

6. *Gather relevant resources in categories*
   For the user to be able to easily understand relations between endpoints, it is beneficial to create categories, and collect related content in these. This way, it is easier for the user to see which endpoints are related, thus likely to be used together for the same tasks, and which are not directly related. Consequently, it can be easier for the user to understand relations, and utilize endpoints together in an effective manner.

7.3 Contribution

This thesis contributes to the research field, by expanding knowledge regarding problems and considerations that should be addressed when facilitating documentation for novice users. Seeing how previous studies primarily have targeted experienced developers (Aghajani et al., 2020; Robillard and DeLine, 2010; Uddin and Robillard, 2015), the findings from this study aims to widen the scope and fill the knowledge gap regarding areas that should be considered in documentation when novices are in focus or a significant part of the expected user group. Even though some findings are matching, e.g.:

1. **Use cases:** Both novice- and experienced users value examples connected to relevant use-cases, in order for them to present how functionalities are used in actual settings of use.

2. **Precise descriptions:** Experienced users and novices value precise and short descriptions of relevant information for different functionalities.
3. **Findability:** The ability to find what one is looking for is important for both novices and experienced users.

there are also differences in our findings compared to said articles. The findings from this study indicate that novice users highlight the value of:

1. **Interactivity:** Novice users value interactivity in examples, a point not mentioned or emphasized in previous studies as important for experienced users.
2. **Guides:** Novice users may appreciate being guided through processes when learning, compared to having the information presented in a way that requires the user to connect pieces of information and figure out procedures by themselves. More experienced users are described as appreciating the latter, thus separating the novices from the experienced users.
3. **Modularized layout:** Novice users express a positive view on a modularized design: design distributed between smaller, concretely purposed pages covering specific purposes (e.g. guide-approach presenting actions separately), rather than a one-page-design (e.g. encyclopedia-approach where all information is gathered on larger pages).

The study can act as a possible practical resource for platform owners, -maintainers and other documentation-designers when designing documentation aimed at a broader experience range, thus potentially contributing to practical improvements of future API documentation.

7.4 Further research

Further research should focus on further broadening the scope, by continuing exploring users within the target group more, also beyond the scope of DHIS2. Gaining more insight, may open the opportunity to develop prototypes that can be tested and evaluated through a design science research-approach. This way, it is possible to gain a more hands-on understanding of what works and what does not, thus extending the knowledge further by providing visual artifacts that can serve as guidance for documentation-designers. This will also allow for a more detailed process of evaluation with users, confirming findings.

7.5 Limitations

As I have attempted to gain insight into the experiences of novice developers regarding API-documentation as a supporting resource, both the data collection and the process of analysis can naturally have been affected by subjectivity and my unconscious underlying preliminary opinions and expectations. I have

however attempted through data collections and analysis to approach participants and data neutrally, in order to avoid bias as a result of my subjective opinions.

Regarding the interviews of students at UiO, I was employed as a teaching assistant, thus, there may have been some degree of bias due to my role in the subject the participants were engaged in. To reduce this, I made it clear from the start what the purpose of the study was, that this data collection was in no way connected to the course, that their answers would stay anonymous at all times and in no way affect the assessment of their project.

Another limitation may be the size of the sample group. We wanted to explore the topic broadly, thus speak to participants with different backgrounds, experience and knowledge. An issue with this, combined with limited time to conduct data collections and analysis, was that we did not have time to speak with very many participants from the different groups. Even though the total number of participants summed up to 15, we could have gotten deeper knowledge and insight with even more participants. Especially the number of participants fitting into the concrete description of the target group would have been beneficial to increase. A difficulty, however, was to identify said participants, as everybody has different experience with different technologies. Identifying lots of users with a fitting degree of experience and knowledge could therefore be time consuming, proving difficult within the time frame of this thesis. It could also have been beneficial to perform more different methods for data collection, in order to triangulate. As mentioned earlier, surveys could have been considered as an alternative method, supporting the interviews. However, as we wanted to perform a qualitative study to gain detailed insight, this was abandoned. Because of the limitations answers to surveys can provide, e.g. regarding short, little elaborated answers and reduced possibility to ask follow-up questions, combined with the previously mentioned difficulty identifying the relevant users within the time limit of the thesis, in order to separate the research from previous studies, interviews and some observations were prioritized.

A third limitation may be the DHIS2-focus of the study. Even though the results are probable to be relevant also for other technologies and APIs, there are areas of DHIS2 that separates it from most technologies, e.g. the often mentioned complex data structure that constructs the system, which may be more relevant for DHIS2 than the average API. Widening the scope of the data collections could have provided more broad insight in a variety of systems. However, participants were also asked to reflect on previous use of APIs and technologies, allowing them to supplement the conversations with experiences also outside the specific scope of DHIS2.

References

4.5 The Data Model. (n.d.). Retrieved March 29, 2023, from https://docs.dhis2.org/archive/en/2.30/developer/html/techarch_data_model.html

Aghajani, E., Nagy, C., Linares-Vásquez, M., Moreno, L., Bavota, G., Lanza, M., & Shepherd, D. C. (2020). Software documentation: The practitioners' perspective. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 590–601. https://doi.org/10.1145/3377811.3380405

Aghajani, E., Nagy, C., Vega-Marquez, O. L., Linares-Vasquez, M., Moreno, L., Bavota, G., & Lanza, M. (2019). Software Documentation Issues Unveiled. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 1199–1210. https://doi.org/10.1109/ICSE.2019.00122

Ahmed, T., & Srivastava, A. (2017). Understanding and evaluating the behavior of technical users. A study of developer interaction at StackOverflow. *Human-centric Computing and Information Sciences*, *7*(1), 8. https://doi.org/10.1186/s13673-017-0091-8

Arisholm, E., Gallis, H., Dyba, T., & Sjoberg, D. I. (2007). Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise [Conference Name: IEEE Transactions on Software Engineering]. *IEEE Transactions on Software Engineering*, *33*(2), 65–86. https://doi.org/10.1109/TSE.2007.17

Arntzen, S., Wilcox, Z., Lee, N., Hadfield, C., & Rae, J. (2019). *Testing Innovation in the Real World*. London: NESTA.

Bevan, N., Kirakowski, J., & Maissel, J. (1991). WHAT IS USABILITY? [Book Title: Contemporary Ergonomics Edition: 0]. *Contemporary Ergonomics* (1st ed.). CRC Press. https://doi.org/10.1201/9781482272574-76

Bianco, V., Myllärniemi, V., Raatikainen, M., & Komssi, M. (2014). The Role of Platform Boundary Resources in Software Ecosystems: A Case Study. https://doi.org/10.1109/WICSA.2014.41

Bigelow, S. J. (2023). What are the types of APIs and their differences? — TechTarget. Retrieved March 22, 2023, from https://www.techtarget.com/searchapparchitecture/tip/What-are-the-types-of-APIs-and-their-differences

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology [Publisher: Routledge _eprint: https://www.tandfonline.com/doi/pdf/10.1191/1478088706qp063oa]. *Qualitative Research in Psychology*, *3*(2), 77–101. https://doi.org/10.1191/1478088706qp063oa

Bush, T. (2019). 5 Examples of Excellent API Documentation (and Why We Think So) — Nordic APIs —. Retrieved February 25, 2023, from https://nordicapis.com/5-examples-of-excellent-api-documentation/

Creswell, J. W. (2006). *Qualitative Inquiry and Research Design: Choosing Among Five Approaches* (2nd edition). SAGE Publications, Inc.

DHIS2. (n.d.). DHIS2 - Home. Retrieved March 23, 2023, from https://dhis2.org/

DHIS2 - YouTube. (n.d.). Retrieved April 18, 2023, from https://www.youtube.com/@DHIS2org

Dzida, W., Herda, S., & Itzfeldt, W. S. (1978). User-perceived quality of interactive systems. *IEEE Transactions on Software Engineering 4 (4)*, 270–76.

Foerderer, J., Kude, T., Schuetz, S. W., & Heinzl, A. (2019). Knowledge boundaries in enterprise software platform development: Antecedents and consequences for platform governance [_eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/isj.12186]. *Information Systems Journal*, *29*(1), 119–144. https://doi.org/10.1111/isj.12186

Furnas, G. W., Landauer, T. K., Gomez, L. M., & Dumais, S. T. (1987). The vocabulary problem in human-system communication. *Communications of the ACM*, *30*(11), 964–971. https://doi.org/10.1145/32206.32212

Garousi, G., Garousi, V., Moussavi, M., Ruhe, G., & Smith, B. (2013). Evaluating usage and quality of technical software documentation: An empirical study, 24–35. https://doi.org/10.1145/2460999.2461003

Ghazawneh, A., & Henfridsson, O. (2013). Balancing platform control and external contribution in third-party development: The boundary resources model [_eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-2575.2012.00406.x]. *Information Systems Journal*, *23*(2), 173–192. https://doi.org/10.1111/j.1365-2575.2012.00406.x

Gluck, M. (1997). A descriptive study of the usability of geospatial metadata. *Annual Review of OCLC Research*.

Golafshani, N. (2003). Understanding Reliability and Validity in Qualitative Research. *The Qualitative Report*, *8*, 597–607. https://doi.org/10.46743/2160-3715/2003.1870

Grossman, T., Fitzmaurice, G., & Attar, R. (2009). A survey of software learnability: Metrics, methodologies and guidelines. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 649–658. https://doi.org/10.1145/1518701.1518803

Hannay, J. E., Dybå, T., Arisholm, E., & Sjøberg, D. I. (2009). The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, *51*(7), 1110–1122. https://doi.org/10.1016/j.infsof.2009.02.001

Hornbæk, K. (2006). Current practice in measuring usability: Challenges to usability studies and research. *International Journal of Human-Computer Studies*, *64*(2), 79–102. https://doi.org/10.1016/j.ijhcs.2005.06.002

Jones, R. S. P., Clare, L., MacPartlin, C., & Murphy, O. (2010). The Effectiveness of Trial-and-Error and Errorless Learning in Promoting the Transfer of Training [Publisher: Routledge _eprint: https://doi.org/10.1080/15021149.2010.11434332]. *European Journal of Behavior Analysis*, *11*(1), 29–36. https://doi.org/10.1080/15021149.2010.11434332

Kajko-Mattsson, M. (2005). A Survey of Documentation Practice within Corrective Maintenance. *Empirical Software Engineering*, *10*(1), 31–55. https://doi.org/10.1023/B:LIDA.0000048322.42751.ca

Lamothe, M., Guéhéneuc, Y.-G., & Shang, W. (2021). A Systematic Review of API Evolution Literature. *ACM Computing Surveys*, *54*(8), 171:1–171:36. https://doi.org/10.1145/3470133

Lau, W. W. F., & Yuen, A. H. K. (2009). Toward a Framework of Programming Pedagogy [ISBN: 9781605660264 Pages: 3772-3777 Publisher: IGI Global]. https://doi.org/10.4018/978-1-60566-026-4.ch601

Layne, C. (2020). A is for Application: API Basics. Retrieved April 18, 2023, from https://medium.com/programming-for-design-practices/a-is-for-application-api-basics-744661bb95a2

Lehman, M. (1980). Programs, life cycles, and laws of software evolution [Conference Name: Proceedings of the IEEE]. *Proceedings of the IEEE*, *68*(9), 1060–1076. https://doi.org/10.1109/PROC.1980.11805

Leung, L. (2015). Validity, reliability, and generalizability in qualitative research. *Journal of Family Medicine and Primary Care*, *4*(3), 324–327. https://doi.org/10.4103/2249-4863.161306

Lidwell, W., Holden, K., & Butler, J. (2003). *Universal Principles of Design*. Rockport Publishers.

Malhotra, N. (1982). Information Load and Consumer Decision Making. *Journal of Consumer Research*, *8*, 419–30. https://doi.org/10.1086/208882

McLellan, S., Roesler, A., Tempest, J., & Spinuzzi, C. (1998). Building more usable APIs [Conference Name: IEEE Software]. *IEEE Software*, *15*(3), 78–86. https://doi.org/10.1109/52.676963

Meldrum, S., Licorish, S. A., & Savarimuthu, B. T. R. (2020). Exploring Research Interest in Stack Overflow – A Systematic Mapping Study and Quality Evaluation [arXiv:2010.12282 [cs]]. Retrieved February 2, 2023, from http://arxiv.org/abs/2010.12282

Mifsud, J. (2011). The Difference (And Relationship) Between Usability And Learnability [Section: Terminology]. Retrieved March 30, 2023, from https://usabilitygeek.com/the-difference-and-relationship-between-usability-and-learnability/

Moran, K. (2020). How People Read Online: New and Old Findings. Retrieved January 27, 2023, from https://www.nngroup.com/articles/how-people-read-online/

Moy, N., Chan, H. F., & Torgler, B. (2018). How much is too much? The effects of information quantity on crowdfunding performance. *PLoS ONE*, *13*(3), e0192012. https://doi.org/10.1371/journal.pone.0192012

Nielsen, J. (1994). *Usability Engineering* [Google-Books-ID: 95As2OF67f0C]. Morgan Kaufmann.

Parnas, D. (2010). Precise Documentation: The Key to Better Software, 125–148. https://doi.org/10.1007/978-3-642-15187-3_8

Pernice, K. (2019). Text Scanning Patterns: Eyetracking Evidence. Retrieved January 27, 2023, from https://www.nngroup.com/articles/text-scanning-patterns-eyetracking/

Preece, J., Rogers, Y., & Sharp, H. (2019). *Interaction Design - Beyond Human-Computer Interaction* (5th). John Wiley & Sons, Inc.

Raiyn, J. (2016). The Role of Visual Learning in Improving Students' High-Order Thinking Skills. *Journal of Education and Practice*, *7*(24), 115–121.

Robillard, M. P., Bodden, E., Kawrykow, D., Mezini, M., & Ratchford, T. (2013). Automated API Property Inference Techniques [Conference Name: IEEE Transactions on Software Engineering]. *IEEE Transactions on Software Engineering*, *39*(5), 613–637. https://doi.org/10.1109/TSE.2012.63

Robillard, M. P., & DeLine, R. (2010). A field study of API learning obstacles. *Empirical Software Engineering*, *16*(6), 703–732. https://doi.org/10.1007/s10664-010-9150-8

Rubleske, J. (2020). The tuning of data platform boundary resources: Insights from Twitter. *Journal of Technology Research*, *9*.

Salleh, N., Mendes, E., & Grundy, J. (2011). Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review. *IEEE Trans. Software Eng.*, *37*, 509–525. https://doi.org/10.1109/TSE.2010.59

Schlatter, T., & Levinson, D. (2013). *Visual Usability: Principles and Practices for Designing Digital Applications* [Google-Books-ID: h_Ql1uIHftoC]. Newnes.

Shackel, B. (1981). *Man-computer interaction: Human factors aspects of computers & people*. Springer.

Shneiderman, B. (1997). *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (3rd). Addison-Wesley Longman Publishing Co., Inc.

Simpson, J. (2022). 20 Impressive API Economy Statistics — Nordic APIs —. Retrieved January 23, 2023, from https://nordicapis.com/20-impressive-api-economy-statistics/

Stølen, K. (2023). *Technology Research Explained: Design of Software, Architectures, Methods, and Technology in General*. Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-25817-6

Tiwana, A. (2013). *Platform Ecosystems: Aligning Architecture, Governance, and Strategy* [Google-Books-ID: IYDhAAAAQBAJ]. Newnes.

Uddin, G., & Robillard, M. P. (2015). How API Documentation Fails [Conference Name: IEEE Software]. *IEEE Software*, *32*(4), 68–75. https://doi.org/10.1109/MS.2014.80

UiO. (n.d.). Home : Development in platform ecosystems. Retrieved March 28, 2023, from https://dhis2-app-course.ifi.uio.no/

Vanhanen, J., & Lassenius, C. (2005). Effects of pair programming at the development team level: An experiment, 10 pp. https://doi.org/10.1109/ISESE.2005.1541842

Web APIs. (n.d.). Retrieved April 13, 2023, from https://www.w3schools.com/js/js_api_intro.asp

Wingkvist, A., Ericsson, M., Lincke, R., & Löwe, W. (2010). A Metrics-Based Approach to Technical Documentation Quality. *2010 Seventh Interna-*

tional Conference on the Quality of Information and Communications Technology, 476–481. https://doi.org/10.1109/QUATIC.2010.88

Wyner, G., & Lubin, B. (2011). From Hello World to Interface Design in Three Days: Teaching Non-technical Students to Use an API. *AMCIS 2011 Proceedings - All Submissions*. https://aisel.aisnet.org/amcis2011_submissions/407

Xu, B., & Li, D. (2015). An empirical study of the motivations for content contribution and community participation in Wikipedia. *Information & Management*, *52*(3), 275–286. https://doi.org/10.1016/j.im.2014.12.003

Zinovieva, I. S., Artemchuk, V. O., Iatsyshyn, A. V., Popov, O. O., Kovach, V. O., Iatsyshyn, A. V., Romanenko, Y. O., & Radchenko, O. V. (2021). The use of online coding platforms as additional distance tools in programming education. *Journal of Physics: Conference Series*, *1840*(1), 012029. https://doi.org/10.1088/1742-6596/1840/1/012029

8 Appendix

A Interview guide - Iteration 1

Iterasjon 1 - Intervjuguide

Innledning

- En kort intro av oss, oppsummering av epostene vi har sendt ut
- Kort forklare oppgaven
    - Motivasjon, mål, tidsramme osv.
- Forklare det vi er ute etter
    - Hvorfor ønsker vi å prate med akkurat *denne* personen

Hoveddel

- Spørsmål om deltakernes bakgrunn
    - Hvor mye erfaring har vedkommende med undervisning?
        - Generelt
        - I aktuelle emner
    - Hva underviser deltakeren i?
    - Har deltakeren erfaring med API-utvikling utenfor undervisning?
    - Hva slags type APIer de har vært innom? (REST, SOAP etc.)
- Spørsmål om undervisningsopplegget
    - Om passende og relevant, be deltakeren evt. vise fram sitt materiale og be dem forklare hvorfor det ble slik
    - Hvordan har fokuset i undervisningen vært? Teori vs praksis, forståelse etc.
    - Har de observert noen gjentakende utfordringer som oppstår i undervisningskontekster?
    - Har de møtt på noen utfordringer selv i prosjekter - enten planlegging eller gjennomføring?
- Nevn kort noen av våre erfaringer (hvorfor vi synes at dette er noe som bør forbedres)
    - Trekk frem konkrete eksempler fra prosjektarbeid
        - Hvordan samsvarer dette med deres erfaringer?
        - Hvordan samsvarer dette med observasjoner/tilbakemeldinger fra deres studenter?
    - Vis eksempel fra DHIS2-dokumentasjon
        - Nevn kort DHIS2 Academy, forklar hvordan dette brukes som en ressurs for å forstå hvordan man kan bruke dokumentasjonen
        - Hvorfor var dette vanskelig å lære/bruke?
            - Kompleks struktur
            - Mye metadata å forholde seg til for å forstå hvordan man kan finne endepunktene man ønsker
            - Lite ressurser som peker på hvordan man bruker APIet - eksempler vanskelige å anvende

            - Playground-ressursen tilbyr noe interaktivt, men denne er uavhengig av dokumentasjonen - om man ikke kjenner til curl forstår man ikke helt hva som foregår eller hva effekten av endringer i forespørsler gjør, slik de er presentert i dokumentasjonen
            - Strukturen gjør det vanskelig å forstå hvor i APIet man er, kan være vanskelig å navigere
            - Informasjonen er ramset opp, men det fremstår ikke som mye sammenheng mellom de ulike overskriftene. Handlingseksempler ("om du vil hente ut denne informasjonen kan du gjøre dette…") mangler for å skjønne sammenhengen mellom kallene og funksjonene som trengs for å få gjort det man ønsker
- Presenter/nevn gode eksempler på dokumentasjon, samt hvorfor
    - Twitter
        - Inneholder eksempler på hvordan man kan bygge opp requests
        - Har eksempler med forskjellig vanskelighetsgrad, viser de enkleste først for å forklare hvordan man setter det opp og oppfordrer nye brukere å forstå disse før de går over til vanskeligere kall
    - Dropbox
        - Inneholder funksjonalitet som viser hvordan man bygger requests i ulike programmeringsspråk
        - Sier noe om hva hver parameter returnerer, hva som kan skje hvis det er feil i parameteren, situasjoner som kan føre til feil ved kallet osv.
    - Forklar hvorfor disse er gode eksempler og hva som skiller dem
- Ved tid - forsøk å utforme en god struktur sammen med deltaker
    - Presenter våre innledende skisser
    - Be deltaker om å kommentere og nevne styrker og svakheter
    - Diskuter løsning

Avslutning

- Har deltakeren noe mer å legge til?
- Spør om deltaker har noen ubesvarte spørsmål
- Takk deltaker for deltakelse
- Spør om deltakeren kjenner til noen andre som kan ha kunnskap om dette, som vi burde kontakte?

B Interview guide - Iteration 2

Discussion outline

Introduction

- Introduce ourselves
- Explain the thesis/task
    - Motivation, goals, time frame, etc.
- Explain what we are looking for
    - What is the reason we want to talk to this particular person?

Practical information (consent, recording etc.)

No confidential personal information will be gathered. Your personal data will only be used for the purposes outlined in the introduction. The information will be handled with discretion and in accordance with applicable privacy legislation. In the transcription and notes, all personal data obtained in the recording will be anonymized.

Voluntary involvement

Participation in the data collection process is optional. You may terminate the interview or withdraw any information at any time. You are free to withdraw your consent at any time and without explanation. Any personal information acquired about you will be discarded if consent is withdrawn, and there will be no negative repercussions if you choose to withdraw consent.

Questions/structure

- Background
    - Experience with development?
    - Have the participants worked with app development towards APIs?
    - Have the participants had subjects/training to learn app development?
- What obstacles have they faced with learning and the documentation
- Ask participants to tell about their process:
    - How have you progressed?
    - What obstacles did you face?
    - How did you solve these obstacles?
    - What experiences have you made?
    - How was the learning process constructed for you to get started and understand more about the project/DHIS?
        - Could anything have been changed to make things easier?
        - What types of learning material have been used?
        - Balance between theory and practice?

                - Theoretical and practical learning at the same time? (e.g. project, hand-ins)
                - …or separated? (e.g. learn about programming and develop something after)
                - What do you consider to be an ideal amount of distribution of theory and practice for learning?
                    - What do you consider to be an ideal learning resource to learn app development?
                    - What do you consider to be an ideal way of learning about API capabilities and usage? (encyclopedia vs. usage guide)
        - Are resources for practicalities that surround app development (ex. Learning how to use supportive tools like Postgres) available to novice students?
    - Have there been instances where the API documentation is not clear on its capabilities?
        - If so, how have you overcome those situations?
        - Also, how do you attempt to learn more about what can be done with the API?
            - Do you read more material about it?
            - …or do you experiment through coding and seeing the results?
    - Have there been instances where you try methods from online forums/ classes that turn out to be outdated or unsupported ways to achieve your goals?
        - Example: trying to change data format received from the API to something that is mutable, but the method is unsupported in newer versions of the programming language
    - How do you approach large data sets and their relations in order to receive the data you want?
        - How do you learn about their relations?

    - What tools have been used?
        - Have these been simple to use? Why/why not?
            - How could these have been improved?
        - Have the participants used the API Testing Tool?
            - How did you use the tool?
            - What worked well using this tool? (Some particular cases where it was useful?)
            - What was difficult/could be improved?
            - How did it influence your understanding of how to use an API.
            - How did it influence your knowledge of the dataModel/Api structure?
        - Have the participants used the DHIS2-documentation?
            - How was this perceived?
            - How was locating what the participants were looking for perceived?
            - What were the biggest obstacles?
            - How could the documentation have been changed to improve the usability for the participants?

Conclusion

- Thank the participants for their participation
- Do the participants have anything more they would like to say?