Enhancing API Learning for Entry-Level Developers

A Study on API Knowledge Resource Optimization

Ashwin Savarimuthu Rajeswaran

Master of Science in
Informatics: Design, Use & Interaction
(60 ECTS study points)

Department of Informatics
Faculty of Mathematics and Natural Sciences

University of Oslo

Spring 2023

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to express my gratitude to my supervisor, Jens Johan Kaasbøll, and Magnus Li, my co-supervisor, for their support and guidance throughout this research. Their expertise and mentorship were invaluable in shaping the direction of this research by providing insightful feedback and criticism, which helped me refine my ideas and strengthen the research. Their guidance was instrumental in bringing this thesis to fruition.

I would like to thank the participants of this study, without whom this research would not have been possible. Your willingness to share your experiences and insights made a significant contribution to the field, and I am grateful for the opportunity to learn from you. Also, a special thanks to those from UNIMA and my research group who welcomed me with open arms during the field trip to Malawi.

Finally, I owe a great debt of gratitude to my family and friends for their unconditional love and encouragement, especially during the most challenging times. Their unwavering support gave me the strength to persevere and push through the obstacles that I encountered and motivated me through the lowest lows.

Thank you all for your support and encouragement throughout this journey.

Ashwin Rajeswaran S.
University of Oslo
May 2023

Abstract

Application Programming Interfaces (APIs) play a pivotal role as critical infrastructural connectors in the development of platform applications. Insufficient API knowledge resources may hinder novice developers from comprehending fundamental platform application functionalities. The acquisition of knowledge pertaining to the design of efficient resources and the resolution of API-related issues through specialized courses can facilitate the creation of educational materials that enhance the learning strategies and problem-solving capabilities of API users. The study of API documentations and other knowledge resources has predominantly focused on experienced developers but lacks sufficient information from a novice's standpoint.

This research aims to offer insights into the reasons behind the perceived inadequacy of API knowledge boundary resources among novice platform application developers. The research addresses the question: *"How can API knowledge resources be optimized using usability principles for entry-level platform application developers?"*. In order to do so, the research will construct a constructivist grounded theory through an interpretive case study and integrative literature reviews. The empirical findings from these research methods are used to discuss and construct a grounded theory using cognitive load theory and usability principles about entry-level developers' API learning experiences.

Usability principles significantly affect learners' intrinsic, extraneous, and germane cognitive loads, according to the study. The research shows that optimizing intrinsic load, increasing the germane load and reducing extraneous load for novice developers is essential for positive learning experiences. This study offers valuable insights into the optimization of entry-level technical knowledge resources by incorporating pedagogical and psychological theories. Additionally, it provides two guidelines that learners can use as a source of inspiration in their learning journey.

Keywords: *problem-solving, API, documentation, pedagogical approaches, user friendliness, usability principles, entry-level developers, education research, platform application development, learning resources, cognitive load theory*

1 | Introduction

This research aims to contribute to the HISP project *"Developing EDucation with Input from CoordinATED research students (DEDICATED)"* in collaboration with East African institutions in Tanzania, Malawi, and Mozambique to develop courses that teach about application development for digital health platforms.

A fundamental part of platform application development is API implementation and use as they provide functionalities and data exchanges to third parties and act as important infrastructural connectors in platform application development (Snodgrass & Winnie, 2019). In large-scale platforms, the APIs provided by the platform owners may be as equally large as the platform systems. These APIs are more complex due to their scale and relation with other components of the API, which make them challenging to learn and can lead to decreased productivity (Robillard & DeLine, 2011). In order to learn and use these APIs as efficiently and effectively as possible, developers need to use API knowledge resources, such as API documentations, that inform about the API's purpose, usage and the vast amounts of functionalities it provides.

Ineffective API knowledge resources have major implications for the learnability of APIs, as these resources may hinder entry-level developers from grasping how fundamental functions work in their platform application projects (AltexSoft, 2022). With the API documentation being one of the most technical of the API learning resources, it is the resource that impacts the developers' experiences the most. In educational settings, this may have a significant impact on individuals who are learning application development practically through, for example, project development.

Also, developers seek out other application development resources in addition to the API documentation to address their use cases (Ghazawneh & Henfridsson, 2013). These knowledge boundary resources are informative materials may be provided by platform owners to educate third parties on the platform, its ecosystem and how the developers can contribute to the ecosystem (Ghazawneh & Henfridsson, 2013; Wingkvist et al., 2010). Knowledge resources therefore become an integral part of the developers' journey in API learning, and they may be left in the dark if the platform community is unable to provide assistance for the developers' use cases when these resources provide inadequate support.

API learning in application development projects is also essential from an educational

standpoint, as it provides developers with a venue to learn how to effectively utilize API knowledge resources to create successful platform applications. API documentations are the most significant type of resource available to the developers, and other learning resources associated with the API supplement the knowledge needed to utilize the API efficiently. In some cases, the documentation may be the only technical resource available to developers, for instance in lesser-known or small APIs. Therefore, ineffective and inadequate resources impede and frustrate inexperienced developers who are unable to adequately comprehend the API or adapt the knowledge imparted to their own application development scenarios.

These adverse experiences have been present for students at UiO who learn platform development with DHIS2 as the primary learning phenomenon in the course IN5320. The course has a project time of 6 weeks, after learning fundamental development, to develop an online application for the DHIS2 platform. As the individuals were novice platform application developers in educational contexts, their primary sources of frustration stemmed from developing sub-optimal solutions or working around development obstacles rather than finding ways to overcome them, especially when the project development time is considerably compressed.

Consequently, learning more about designing effective resources and developing courses that provide information on problem-solving with APIs may aid in the creation of content that enhances API learning strategies and application development problem-solving for the learners. Therefore, technical documentation should be examined from the perspective of user experience and usability, given that documentation is the most technical resource that application developers must manage and comprehend. The study will also examine existing API documentations using psychological- and pedagogical principles to determine platform development learning challenges and how to enhance documentations and learning resources to better support API learning by less-experienced platform application developers. The objective is to find effective API learning resources and the factors that contribute to positive learning experiences.

Exploring how documentation can be designed with psychological- and pedagogical considerations and usability in mind can contribute significantly to the creation of documentations that improve the learning experiences of less-experienced individuals and facilitate an efficient development process. While this study utilizes DHIS2 as a starting point, it is anticipated that this information can be applied to other data-complex APIs with the same limitations as DHIS2.

1.1 Knowledge gap

Previous research mainly concerns more experienced developers when API documentations and other knowledge resources are studied (Robillard & DeLine, 2011; Uddin & Robillard, 2015). Entry-level developers have been studied in regards to API usability (Piccioni et al., 2013), but there is not enough knowledge from a novice perspective on API learning resources. Additionally, the documentation still needs to adequately address both inexperienced and more-experienced developers for it to be an effective informative resource. There is therefore a need for further research into what constitutes documentation that fosters self-learning, is user-friendly for beginners, and is also efficient for professional developers. Additionally, providing one effective resource that accommodates both groups of API documentation users remains to be achieved. Hence, this research will investigate the differences in documentation usage goals to better understand how documentations may support diverse target groups with varying levels of application development competence and experience.

Insufficient courses that do not equip students with the knowledge or skills necessary to adapt simple use cases presented in the documentation to more complicated development situations also offer difficulties for students. It is challenging for teachers to provide adequate help when students' development projects involve unexpected and complex scenarios. Understanding the challenges of adjusting documentation to more complicated use cases will help learn more about how to equip beginner platform application developers with the means to efficiently learn and understand how to use APIs.

1.2 Theoretical motivation

Uddin and Robillard (2015) discovered in their study of API documentation that less-experienced developers rely more on code examples and online communities to learn APIs. They suggest that API designers should prioritize clear and useful code examples and encourage community participation to assist developers in using APIs. Robillard and DeLine (2011) discuss the evaluation of API documentation by more experienced developers. According to the study, developers prefer clear, concise documentation with examples and code snippets. Piccioni et al. (2013) studied API usability and reached the same sentiment, and they also argued that imprecise or incomplete descriptions would reduce the perceived usability by their target group. Likewise, the study by Robillard and DeLine (2011) also found that documentation quality influences developer API satisfaction and usage.

Robillard and DeLine (2011) also discuss the use of visual elements and organization

to enhance comprehension and usability, while Uddin and Robillard (2015) examine the effectiveness of API documentation examples and propose a model for creating them. These articles seek to enhance the usability and effectiveness of API documentation and assist developers in utilizing APIs. Additionally, the API usability study by Piccioni et al. (2013) complement these studies with an API usage perspective. The studies highlight the different needs between entry-level developers and experienced developers, and how those user demographics have needs that overlap with each other.

Existing research on API learning and API documentation does not place significant attention on *how* APIs are learned. Moreover, there is insufficient research on what constitutes "good" and "efficient" documentation from a pedagogical and user experience standpoint. Equally rarer are pedagogical and usability-related studies on other significant API knowledge resources that can enhance the API learning experience. Some research, such as that conducted by Robillard and DeLine (2011), has focused on determining why API documentation fails for more experienced developers. However, the impact of other educational materials designed to supplement the documentation, such as online courses, has not been studied. When resource providers overlook the fact that documentation alone is insufficient to learn about the API and fail to provide adequate resources that complement and explain the API, novice developers may experience stagnation and frustration. The novice developers are then, in turn, required to spend additional time understanding their obstacles and finding solutions.

As previously stated, it is necessary to overcome the challenges that API knowledge resources present to inexperienced developers. Given the limitations associated with the documentation's pedagogical contribution, a more pedagogical approach is necessary to overcome them and enhance the material's learnability. Additionally, it will be essential to address the user-friendliness of the documentations in order to develop materials that provide the assistance needed by less experienced groups by improving the usability of the materials. As a starting point, it is essential to examine knowledge resources such as the API documentation since it contains the most technical documentation available to developers. Moreover, this may in some cases be one of the few resources they have access to for APIs that are not widely used. It is therefore necessary to address the obstacles in the documentation in order to improve it, as well as to produce supplementary content to compliment it.

This research will contribute to the field by shedding light on why entry-level platform application developers may perceive API knowledge boundary resources as insufficiently meeting their needs during their API learning journeys and by examining their problem-solving strategies.

1.3 Research question

The study will be designed with a constructivist research paradigm since the objective is to comprehend the factors that contribute to a perceived problem and suggests solutions (Verne & Bratteteig, 2018). Consequently, the research question will request an investigation of the issue, and the data will be collected in order to examine the issue from multiple perspectives to gain an understanding of the phenomena (Verne & Bratteteig, 2018). The study will fundamentally view this subject on API knowledge resources through usability principles, which are principles that can be applied to interface design to make it more user-friendly and increase adoption rates (Lee et al., 2010; Mvungi & Tossy, 2015). More specifically, the usability of the learning material will be evaluated in order to study the learnability of the API. The research question for this study is as follows:

RQ: *How can API knowledge resources be optimized using usability principles for entry-level application developers?*

As a starting point, the research will first dive into the most technical knowledge resource associated with APIs – the API documentation – as that is the resource that provides the most insight to developers about the possibilities of the API and how to use it. The following sub-questions (SQs) have been constructed to aid the process of gaining answers to the research question:

- **SQ 1:** *What specific obstacles do entry-level application developers face when attempting to learn APIs and how can they be addressed?*
- **SQ 2:** *How do entry-level developers overcome their obstacles currently?*
- **SQ 3:** *How can an API documentation be written to support learning and efficient problem-solving?*
- **SQ 4:** *How should the knowledge resource producers communicate with and guide the development community, given the diversity of the user base?*

In order to study these questions, interpretive case study, semi-structured interviews, participant observations and integrative literature reviews will be conducted. The main contribution from the research will be characterization of factors that impact the learnability and usability of API knowledge boundary resources so that the learning resources can be designed for entry-level developers, and how they can approach problem solving.

1.4 Thesis structure

The remainder of the thesis is organized as follows:

Chapter 2 - Background: The background chapter provides a thorough overview of the research topic as well as descriptions of key theories and concepts. It examines relevant concepts such as knowledge resources, API, DHIS2 and its resources, and relevant studies related to API documentations.

Chapter 3 - Literature Review: The literature review chapter synthesizes research topic literature of psychological and pedagogical theories such as Learner-Centered Design (LCD), usability principles, cognitive load theory, worked examples, expertise reversal effect, layer-cake scanning method and inquiry-based learning. It critically evaluates the theories, concepts, findings, and debates concerning these topics in relation to API knowledge resources.

Chapter 4 - Research Approach: This chapter outlines the research's philosophical assumptions, involvement style, target group characteristics, methodology, and research methods used in data collection and analysis. The qualitative interviews in the interpretive case study were done in collaboration with Johannes Skøien to understand sample groups' API learning obstacles and strategies, and the integrative literature review represents the theoretical sampling of the study.

Chapter 5 - Findings: The chapter on findings provides a comprehensive review and analysis of the data collected during the preliminary study and interpretive case study. The preliminary study provides an overview of the research's starting theoretical perspective, whereas the interpretive case study is divided into two iterations of the qualitative interview processes with differing sample groups.

Chapter 6 - Discussion: This chapter provides an in-depth analysis of the interpretive case study and the integrative literature review. The findings were discussed using psychological-, pedagogical-, and usability-related knowledge. Two API learning guidelines, one theoretical and one practical, were presented as final contributions of the study.

Chapter 7 - Conclusion: The conclusion chapter restates the research questions and provides a summary of the study's key findings, contributions, and implications. In addition, limitations, avenues for future research and research validity and reliability are provided.

2 | Background

The background chapter provides an overview of the research topic and descriptions of key theories and concepts such as knowledge resources, API, DHIS2 and its resources, and relevant studies related to API documentations.

2.1 Application Programming Interfaces (API)

APIs are defined as the interface to a reusable software entity utilized by various clients outside the creating organization, and that can be provided separately from environment code (Maalej & Robillard, 2013; Robillard et al., 2013, p.1). They enable connection to other systems' infrastructure to enable the use of functionality from the connected infrastructure. Most large-scale systems include libraries, frameworks and functionality made available through APIs, hence API usage is a regular element of the programming process (Maalej & Robillard, 2013; Uddin & Robillard, 2015). As APIs are extensions of software, manufacturers have to innovate through means like adding new features, fixing security problems, making development easier, and decreasing technical debt by eliminating no longer utilized functionality (Lamothe et al., 2021).

The communication between a software and a server of an API service follows a HTTP protocol where the client sends a request to the server and the server responds (freeCodeCamp, 2022). The protocol provides the means necessary for the developers to build and connect their applications to the vendor's servers (Tray.io, 2023), and the most commonly used API type is REST (Hygraph Team, 2022). A REST API request consists of an URL (also called endpoint), a method (GET, PATCH, POST, PUT, and DELETE), a header and a body (the requested data) (Hygraph Team, 2022). The data response that is provided by the API when a request is sent to the server is referred to as a 'payload'.

API learning can be understood as the process of acquiring and assimilating knowledge about underlying API structures that are essential for API operation. According to Gao et al. (2020, p.2), API learnability research often concerns the developers' API learning experience, API documentation, and designing APIs for learnability. The goal of API learning enhancement is to provide efficient and effective API knowledge resources that give the learners the ability to quickly learn new APIs (Gao et al., 2020). These learning materials provide knowledge through documents, articles, tutorials and code samples (Yin et al., 2021).

2.2 Knowledge resources

APIs are one of many boundary resources that platform owners can distribute in order to cultivate the platform ecosystem (Ghazawneh & Henfridsson, 2013). The main purpose of knowledge resources in the context of platform development is to convey knowledge to third party application developers about the boundary resources of the platform and its ecosystem. These resources facilitate how the application developers can use core functionalities to build applications (Ghazawneh & Henfridsson, 2013) and consequently contribute to the platform ecosystem. Therefore, it is important for the platform owners to cultivate growth of knowledge through developing these resources and updating them as the platform core grows (Ghazawneh & Henfridsson, 2013). In the context of this study, the terms 'knowledge resources', 'pedagogical resources' and 'learning material' will be used interchangeably.

2.2.1 API documentation

API documentations are knowledge resources that are used as tools by developers to gain insight in the interfaces related to domain terms and other elements of the interface in order to utilize the libraries and frameworks provided by the API correctly in programming tasks (Maalej & Robillard, 2013). In contrast to other learning material, API documentation provides information (such as classes, methods and so on) about each element present in the API in a systematic order, for instance alphabetically (Maalej & Robillard, 2013). As understanding the API usage is crucial to the development process, efficient and resourceful documentation is needed in order to provide easier access to knowledge and efficient development processes (Maalej & Robillard, 2013).

Documentation purpose: Instructional guide vs functionality compendium

There are fundamentally two distinct purposes of why a developer may seek to use API documentation as a learning resource. For developers who are newly exposed to the API, gaining operational instructions and theoretical knowledge is important in order to use the API (Snodgrass & Winnie, 2019). As they become more confident in their knowledge about data structures and other theoretical aspects of the API, the need for this information may decrease. As a result, the need for a comprehensive listing of functionalities provided by callback functions may increase in importance. For that reason, it can be argued that developers need different types of documentation at different stages of their learning process (Piccioni et al., 2013). The design and assisting purpose of these two API documentation designs result in significantly different information presentations.

An example of API documentation in the instructional guide format can be the Twitter

API documentation (Twitter, 2023). The documentation style provides a linear learning path with tools to get started with the API, fundamental theories, tutorials and the likes (see figure 2.1).

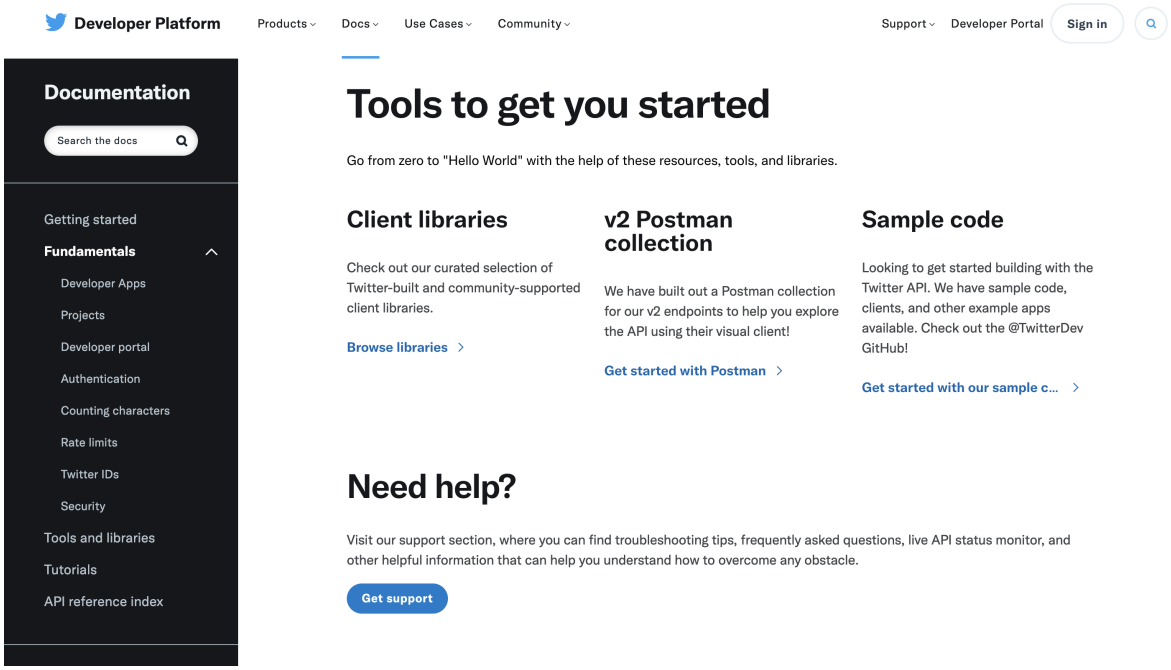

Figure 2.1: Instructional guide format of the Twitter API documentation

Additionally, GitHub provides the REST API documentation in the functionality compendium format (GitHub, 2023). This sidebar provides a comprehensive overview of the callback functions available, and provides more detailed information about the different functions once they are accessed (see figure 2.2).

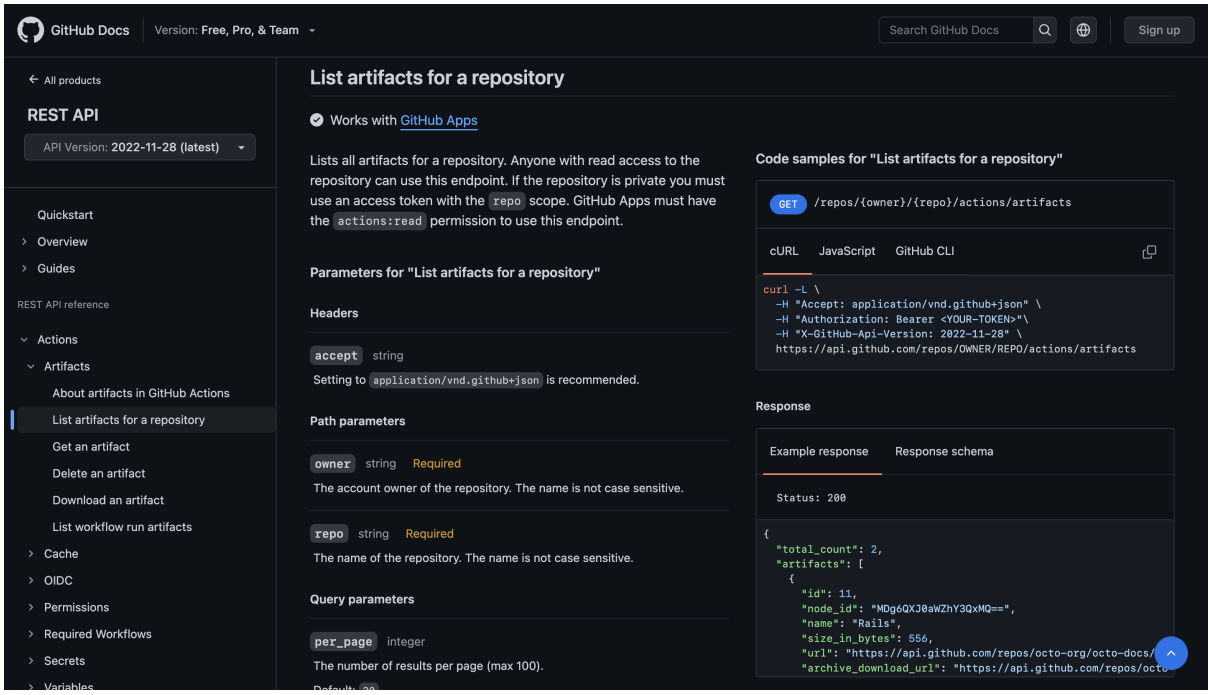

Figure 2.2: GitHub's REST API documentation that shows a comprehensive overview of functions

2.3 Studies of API documentation and learning obstacles

The following information is concatenated findings of API documentation suggestions and observed learning obstacles from existing literature.

Important sample group considerations

Robillard and DeLine (2011) conducted their study with participants who are developers with an average of 9.8 years of development experience, indicating that the sample group had extensive professional expertise. Even though the study highlights criteria that may be useful also for less experienced developers and students, this is an important aspect to be aware of. Additionally, the study by Uddin and Robillard (2015) had a sample group with an average of 1-40 years of experience, and a mean of 13 years. The level of experience between their sample group and the learning population of entry-level DHIS2 developers is likely to differ. Even though the conclusions from the study are likely still relevant to a large extent, this is still important to keep the level of experience in mind when comparing with these API documentation usage obstacles.

2.3.1 The dichotomy between simplicity and depth

Information may possess simplicity and depth as attributes to describe the different ways content is presented in knowledge resources. Their relations are elaborated in order to provide a clearer picture of what content simplicity and depth entails in this research.

Determining 'depth' through penetrability and simplicity

Robillard and DeLine (2011) present the concept of **penetrability** which concerns how API "facilitates exploration, analysis, and understanding a fine line between an over-exposure of the APIs internal elements (which violates the principle of hiding), and a design that makes the behavior of the API impenetrable". Designing a penetrable API entails making a distinction between what must be transparent and encapsulated to the readers and what must remain opaque to understand the performance of the API function to predict the outcome of said functions so that so that developers are able to see and understand the structure and behavior of the API (Robillard & DeLine, 2011).

The article suggests " (1) explicit documentation on the performance consequences of API elements, (2) better descriptions of error handling, and, (3) for methods triggering significant chunks of behavior, explanations of the abstract operations of the API" as possible ways to increase the penetrability of the API documentation content (Robillard & DeLine, 2011), which may assist in increasing the level of depth in the provided content. Overall, the objective would be to make the API as clear as possible, so that developers can quickly learn how to use it.

Important considerations to make the API design and the documentation more penetrable could be to:

1. Design the API with well-defined structures and behaviors.

2. Provide sufficient information and examples to help developers understand how to interact with the API.

3. Provide feedback or error messages that assist developers in understanding why their code is not functioning as expected.

The characteristics of penetrable APIs can serve as a model for API documentation creators. For instance, in the code examples provided in API documentation, having examples that are easily understood can facilitate developers' learning. By adhering to the principles of penetrability, such as designing a well-defined structure and behavior, providing sufficient information and feedback, and differentiating between what must be

transparent and what must remain opaque, the API documentation can become more penetrable and easier to understand for developers. Ultimately, this could result in a more efficient and effective learning process.

Simplicity on the other hand is a design principle that promotes the idea of reducing design elements that do not affect the overall functionalities of the artifact (Anaç, 2022; Preece et al., 2015). In a sense, it would entail reducing extraneous complexity that may adversely affect the users' goals of use. In the context of knowledge resources, it illuminates the idea of presenting information in a clear and concise manner that is easy for learners to understand. One way of manifesting simplicity in designs is through focusing on essential features and functions of the artifact to ensure optimal performance (Lawrence, 2020).

Implications on this study

Simplicity and penetrability become two different approaches in presenting information in documentation design; the concept of penetrability places emphasis on the precision and comprehensiveness of information, while simplicity prioritizes the elimination of extraneous complexity and the delivery of information in a clear and straightforward manner. Both of these perspectives illuminate two of the different ways to understand **clarity**; the quality of being clear and straightforward to understand, and the quality of being fully detailed to facilitate easier comprehension (Merriam-Webster Inc., 2023).

The clarity attribute is, by these definitions, attunable to both simplicity and penetrability. Simple and concise descriptions exhibit high simplicity and clarity, and are not necessarily comprehensive. On the other hand, detailed descriptions exhibit high penetrability and clarity, which provides higher levels of depth to the content. In this study, simplicity and depth are used to describe these two forms of content presentation with the assumption that the written content in the knowledge resources are formulated with high levels of clarity. Higher levels of simplicity entails precise, short descriptions, with only the most necessary details included to understand and operate the API functionality. Higher levels of depth would entail substantially precise and comprehensive descriptions. Examples of how these attributes can manifest in API documentations are shown below (see figure 2.3).

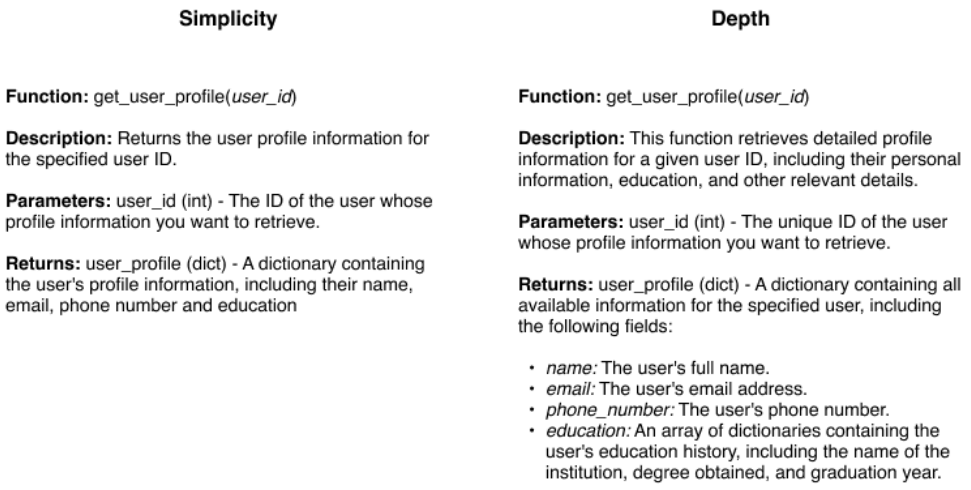

Figure 2.3: Simplicity attribute manifesting as clear and concise descriptions of an API function, while depth manifesting as detailed descriptions.

2.3.2 Intent documentation

It may be necessary to develop a well-organized and logical structure for learning resources such as API documentation so that the target group may access the knowledge they require quickly. The necessity of segregating content and formatting the documentation with a goal-oriented plan is discussed in Aitman's video conference on authoring successful documentation (Aitman, 2019). They propose creating content that responds to user demands by addressing the end user's overall objectives. A similar concept is addressed as 'intent documentation' in a study of API learning obstacles (Robillard & DeLine, 2011) where the documentation should give information about the API's intended usage.

These studies do, however, present the goals of intent documentation in slightly different ways. Aitman's reasoning for intent documentation is for it to meet the specific goals that a reader might have when visiting the documentation, while Robillard and DeLine's research mention intent documentation when it comes to API design decisions and intended usage. Additionally, Robillard and DeLine (2011) argue that intent documentation "provides information about why certain API design decisions were made, and how the API is intended to be used" to ensure that developers do not become uncertain of the provider's intended purpose of the API. Though simultaneously, the study also indicates that this additional information may bloat the documentation and should therefore be provided on a 'as-needed' basis (Robillard & DeLine, 2011). They also claim that correct usage of the API is not self-evident, and that providing the intention of use can make performance aspects clearer for the readers (Robillard & DeLine, 2011).

2.3.3 Textual narratives and content segmentation

The slight difference between intent documentation goals formulated above leads to implications about what is considered to be a well organized documentation with a logical structure. With Aitman (2019)'s suggestion one may write documentation segments where multiple interconnected API methods may be mentioned under the same goal, while in an API learning obstacles research (Robillard & DeLine, 2011) it was implied that "A Continuous Narrative Provides a Single Place to Look and an Obvious Place to Begin". In other words, the first formatting suggestion is to differentiate content and segment the information through outlines, while the latter suggests making segments in a single page. An example of a continuous textual narrative style formatting is the DHIS2 Developer Manual, while a segmented formatting suggestion can be for example the DHIS2 web API documentation and Google's Android API reference documentation where each segment opens a new page (Google, 2022). Examples of both styles, the Twitter and Dropbox API documentation, were illustrated previously (see chapter 2.2.1).

The downside of the aforementioned level of segmentation is the lack of opportunities for the reader to search through the document with the browser's searching functionalities. Robillard and DeLine (2011) also argue that too much fragmentation decreases the likelihood of discovering the information that is needed. Therefore, there needs to be a balance of fragmentation that needs to be considered, especially if the fragments open new pages or that redirects the users attention. Aitman also alludes to this balance, mentioning that including more information about a topic "just because it's connected" becomes counter-productive for the readers as there is more to process and the additional information *may* be irrelevant to *their* goals (Aitman, 2019). They also note the importance of using hyperlink structures to create a well organized and logical structure for this level of fragmentation, though Robillard and DeLine (2011) discovered that said structure can be disconcerting for developers, especially if it impacts the ability to quick-search a coherent, linear document with browser functionalities significantly.

Robillard and DeLine (2011) also reported that boiler-plate (generic) description of methods does not yield developers any valuable insight and may be a waste of their time to read through, as they are standardized text elements used frequently with minimal or no alteration across various contexts or circumstances. The setback is the lack of specificity regarding the endpoint's functionality and optimal usage, thereby constraining its utility for developers who require comprehensive guidance to proficiently utilize the API. The article also suggests that fragmentation makes information less discoverable and developers most often look for coherent and linear presentations of the documentation to find the information they seek (Robillard & DeLine, 2011).

2.3.4 Information overflow and front-loading sentences

An overflow of information should be avoided so that the reader is able to understand information at a glance (Aitman, 2019). A technique to do so is writing front-loading sentences; a writing style in which the reader's plausible aim is stated at the beginning of a fragment, allows the reader to determine whether or not they need to read the part. For example, the statement "*You need to do X if Y is the case*" may be rewritten as "*If Y is the case, do X*" (Aitman, 2019). Front-loading sentences will accommodate the readers' reasons for seeking the documentations in the first place; either emphasizing scenarios they are experiencing or emphasizing the functionalities they may be looking for.

2.3.5 Code examples

Providing code examples may also provide a certain amount of context about the specific aspects of the API (Robillard & DeLine, 2011). The study highlights two implications for working around the learning obstacles regarding the complexity and recommendations of the examples. The first implication is to provide small examples, for example code snippets, that demonstrate usage patterns with related method-call examples. The other implication is that these examples demonstrate "best practices" for using an API (Robillard & DeLine, 2011).

Forms of code examples

There are three different forms of code examples that are prevalent in API and application development learning material; code snippets, tutorials and applications. The formats are distinct in the following ways:

Code snippet: Code snippets are small lines of code that demonstrate how to access basic API functionality (Robillard, 2009). They usually do not address relationships with other functionalities that need to be put together in order to achieve full operation.

```python
import requests

# Define the endpoint and parameters
url = 'https://api.example.com/users'
params = {'name': 'John', 'age': 30, 'country': 'US'}

# Make a GET request to the API
response = requests.get(url, params=params)

# Check the response status code
if response.status_code == 200:
    # Extract and print the response data
    data = response.json()
    print(data)
else:
    print('Error:', response.status_code)
```

Figure 2.4: Snippet of code that makes a GET request with parameters to an API endpoint using the popular requests library.

Tutorials: Tutorials are multiple code segments of a somewhat complete application that are intended to teach a specific aspect of the API (Robillard, 2009). Programming and development skills are often taught using tutorials, which can be written, video, or interactive online. More specifically, they provide step-by-step descriptions of how to achieve an end result.

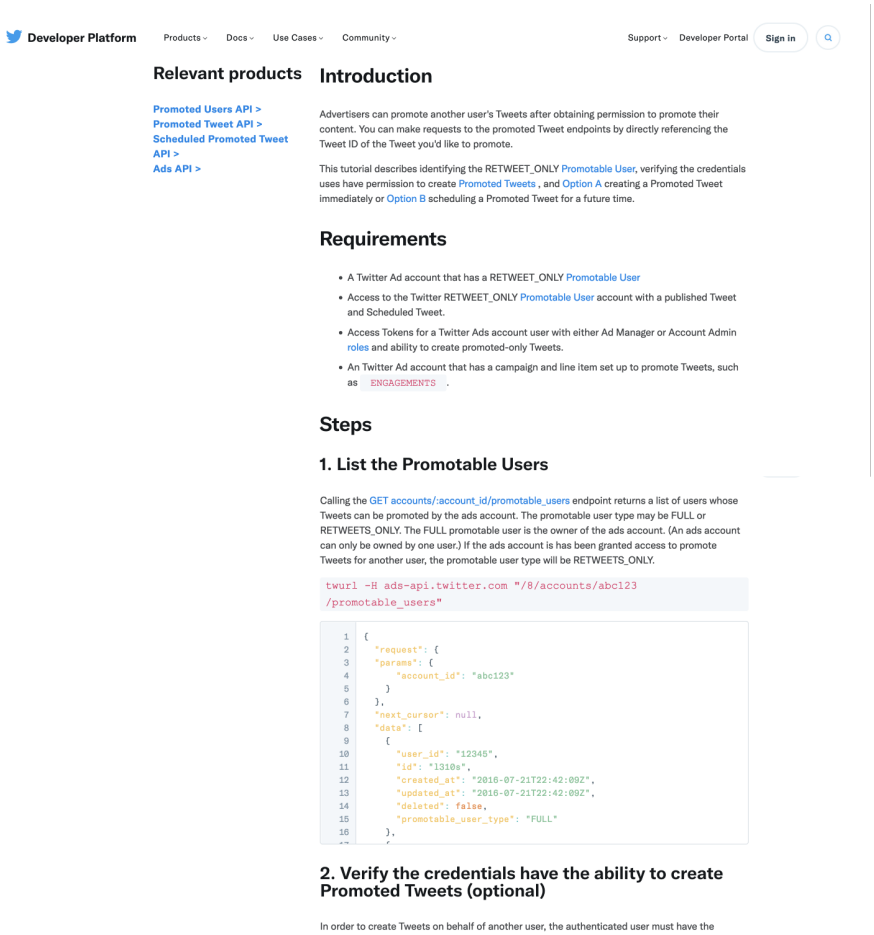

Figure 2.5: Example of a tutorial from the Twitter API documentation

Applications: Applications can also be considered code examples as they can demonstrate how the API works in a complete context, or be open source projects downloadable for developers to inspect (Robillard, 2009). For example, when documenting a social media API and demonstrating how to retrieve a user's profile you can showcase a real-world application that uses the API to retrieve and display the user's profile information rather than providing a code snippet with generic placeholder values.

Sandbox environments: Sandbox examples, by definition, are environments constructed for developers in order to test and demonstrate different aspects of the infrastructure, interface and other aspects of the system (Arntzen et al., 2019). In other words, these tools are designed to give developers a way of learning how the system (in this case, the API) works before coding and implementation to grasp what is necessary to be implemented, changed or how to alter the provided functionalities to receive desired results. The tool aids in answering the developers' questions about the different components and their respective reactions (Arntzen et al., 2019).

2.3.6 Matching APIs with scenarios

Robillard and DeLine (2011) also provides insight into developers becoming surprised and confused when they were unable to match their usage scenarios and adapt functionalities from previously used APIs onto other kinds of APIs. The implication is to match API elements with scenarios to compensate for hard-to-understand structures (Robillard & DeLine, 2011). An API designer can mitigate learning obstacles by providing clear and concise documentation that describes the scenarios in which the API is intended to be used and how to match API elements with those scenarios. For instance, the designer can provide specific use cases and examples for API functions to aid users in comprehending how to use them. In addition, the designer can provide detailed information regarding the parameters required for each API function and how they should be formatted to ensure that the correct results are returned. The designer can also provide practical examples and case studies that demonstrate the API's use in real-world scenarios, which can assist users in applying their knowledge and better understanding the API's functionality.

2.3.7 Reported API documentation problems

A study by Uddin and Robillard (2015) discusses the findings of two surveys on API documentation issues. The first polled IBM employees in Ottawa and Toronto, and 69 responses (9.9% of those who responded) were utilized to identify 10 flaws with current documentation. The authors then conducted a follow-up survey of IBM employees in Canada and the United Kingdom, inquiring about the frequency and severity of the prior study's difficulties. These results of these surveys provide an overview and summary of existing API documentation issues, as well as how developers perceive them. This research looks into a number of topics, including some common annoyances and difficulties with reading and using API documentation in general. It provides a clear picture of areas for improvement and can thus be quite valuable for future research processes. The reported documentation usage obstacles are listed below (see figure 2.6).

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 2.6: API documentation problem reported in a survey (Uddin & Robillard, 2015, p. 70). E* = number of examples that mentioned a problem, D* = number of developers who reported a problem

2.3.8 Teaching non-technical students to use an API

Wyner and Lubin (2011) studied MBA students who were instructed in the coding and use of a simple, custom API during a one-week intense lab lecture. Some of the participating students have experience with technology development, while others have none. They studied programming fundamentals for three days before implementing a simple restaurant API.The study does not focus on API documentation for a realistic and widely used API; rather, it employs a simplified API adapted to the students' level of expertise to examine how students comprehend how APIs function and to teach them fundamental programming. A particular observation in this study was the construction of an API documentation that contained precise, practice-oriented descriptions and output examples of the expected behavior (see figure 2.7 and 2.8). Their pedagogical strategy was "to carefully design our own custom API to precisely fit the needs of our students, rather than forcing them to confront an overly complex real-world system." (Wyner & Lubin, 2011, p. 6)

```
Tax exempt? (y or n):  n
Item: 3.95
Item: 6.95
Item: 7.95
Item:
Subtotal: 18.85
Tax: 0.9425
Total: 19.7925
```

Figure 2.7: Sample of expected algorithm output used in lab lectures (Wyner & Lubin, 2011)

Function	Description
`start_meal(restaurant_name, greeting)`	Initiates the system for a new group of diners. Does not return anything. Displays restaurant name followed by greeting on the line below.
`end_meal(restaurant_name, message)`	Closes out the session for the current group of diners. Does not return anything. Displays message followed by restaurant name.
`show_logo(restaurant_name)`	Displays restaurant name surrounded by a box. Used mainly by start_meal and end_meal but you can use it any time you want to display a logo.

Figure 2.8: Snippet of table used in the final lab session. Provides API functionalities with brief descriptions of their use (Wyner & Lubin, 2011).

The observation is interesting as the documentation structure and presentation may have made the learning process easier for the less experienced students. Additionally, as illustrated in figure 2.8, their API documentation also highlights expected output in addition to its use. It also underlines the importance of APIs in software development and their usability. Given the positive response to the course, it's clear how important and useful API knowledge is right from the start of programming.

2.4 Examples of good API documentation

Thomas Bush (2019) conducted a comprehensive examination of API documentations and their design, including potential examples and their defining characteristics. The blog post provides a careful examination of five exemplary API documentations from renowned companies, namely Stripe, Twilio, Dropbox, GitHub, and Twitter. The author describes the distinguishing characteristics that contribute to the efficacy of these API documentations in providing developers with a smooth and enjoyable experience. Notably, Dropbox, GitHub, and Twitter are chosen as particularly interesting examples for the purpose of this study, highlighting the importance of providing developers with a pleasant and streamlined API documentation experience. The blog post highlights key components of effective API documentation, including clear explanations, code snippets, example responses, authentication and quick-start guides, and the ability to accommodate a variety of programming languages and developer backgrounds.

2.4.1 Dropbox API documentation

The API documentation stands out for its simplicity and approach that allows developers to select their preferred programming language prior to receiving language-specific documentation. This strategy is commended for accommodating the diverse backgrounds of developers. It also eliminates unnecessary information overload and focuses on language-specific implementation advice, making it easier for developers to get started quickly.

2.4.2 GitHub API documentation

The top-right menu of GitHub's REST API documentation is commended for pointing developers in the right direction with links to reference material, guides, and libraries. The use of well-ordered, blog-style articles to provide guidance to developers makes the documentation more engaging and straightforward to follow, thereby enhancing the user's learning experience. The article also highlights the the inclusion of a small status widget on every page that provides information about the status of the API allows developers to quickly determine whether or not any problems they are experiencing with their implementation are server-related, saving them time and effort.

2.4.3 Twitter API documentation

The API documentation for Twitter is commended for its visually appealing and interactive design. The single container layout focuses the user's attention on a single location, creating a seamless experience. The documentation is flexible and user-friendly because it includes quick notes on estimated reading time, the ability to switch between different

methods within a page, and a clear "Next Steps" heading. Additionally, the presence of a Frequently Asked Questions (FAQ) section on nearly every page improves the user experience. This separate FAQ section enables users to quickly locate answers to frequently asked questions without having to search through the main content, thereby facilitating the answer of their inquiries. In general, Twitter's API documentation stands out for its visual appeal, interactivity, and adaptability, making it an excellent option for developers.

2.5 HISP and DHIS2

District Health Information System 2 (DHIS2) is a global, open-source platform for health data collection, analysis, and management. The HISP Center at the University of Oslo (UiO) is in charge of developing, coordinating, and operating the platform. The platform is used in 73 low- and middle-income nations by 2022. The platform provides functionalities such as data warehousing, visualization features, and the possibility for data users and policy makers to generate analysis from live data in real-time (HISP, 2023).

2.5.1 Metadata design

The metadata design in the DHIS2 API is a hierarchical structure of metadata elements that defines how data is collected, stored, and analyzed within the system (DHIS2 Team, 2023a). The metadata elements in DHIS2 consists of:

- *Data elements:* The fundamental components of the DHIS2 data model. They represent a single piece of collected data, such as the age or diagnosis of a patient.
- *Data sets:* Collections of related data elements used for a specific purpose, such as monitoring a specific health program or tracking a disease outbreak.
- *Categories:* Used to group data elements with similar characteristics, such as clinical data, demographic data, and program indicators.
- *Category combos:* are sets of categories used to identify data elements. Combining categories allows for a variety of reports and visualizations that help identify data patterns and trends.
- *Option sets:* defines a list of possible values for a particular data element, such as a list of possible diagnoses.
- *Data element groups:* Collections of data elements that can be combined in various ways, such as a collection of demographic data elements.

Understanding these metadata elements is necessary for efficient use of the DHIS2 API, as the API interacts with them to create, update, and retrieve system data. For instance, when using the API to create a new data set, you must specify the data elements and categories that comprise the data set. Similarly, you would need to know the data element IDs and data set IDs to retrieve data from the system using the API.

Visualization of the metadata design

Following is a simplified visual representation of the metadata design described above, illustrating the relationship between the various metadata elements.

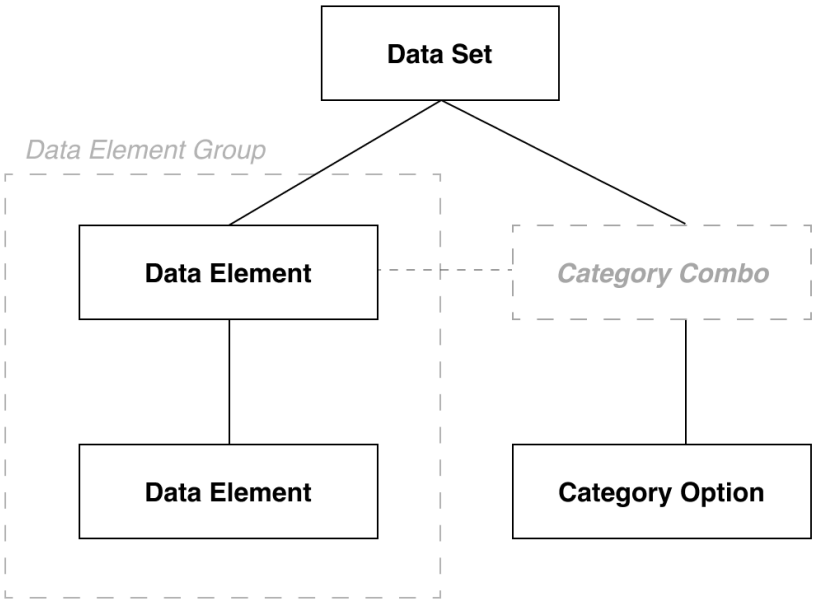

Figure 2.9: The structure of the DHIS2 API metadata elements.

The Category Combo is not contained within the Data Set in the same way as a Data Element. It is independent from the Data Set despite being associated with it, hence the reason behind why it is grayed in the visualization. The gray line between Data Element and Category Combo indicates an association between the Data Set and Category Combo metadata elements. A Data Set is specifically associated with a Category Combo, which determines the available category options for the Data Set's data elements. The Category Combo specifies the categories that can be used to disaggregate the data in a Data Set and the available options for each category. For instance, a Data Set may collect information on the health services provided by various facilities. The Category Combo associated with this Data Set may contain categories for facility type, facility location, and health service type, with multiple options for each category.

Two Data Elements are depicted as being connected by a line in this visualization. This indicates that the Data Elements belong to a particular type of relationship known as a "data element group." A data element group is a method for organizing Data Elements that are related for easier management and analysis. Data Element groups can be utilized to organize related Data Elements based on criteria such as location, program, or data type. It becomes simpler to manage and analyze the data collected by Data Elements when they are grouped together.

2.5.2 Developer manual

The DHIS2 Developer Manual is a comprehensive guide for developers who wish to work with DHIS2, detailing the architecture, APIs, and development tools utilized by DHIS2. The manual covers topics such as establishing a development environment, working with metadata, and building custom applications with DHIS2. While it is a guide to DHIS2 development which covers a variety of topics beyond the web API, it is not specifically focused for web API developers. This resource is frequently referenced during the second iteration of data collection, when it represents the textual narrative style of presentation in contrast to the DHIS2 web API documentation (see figure 2.10).

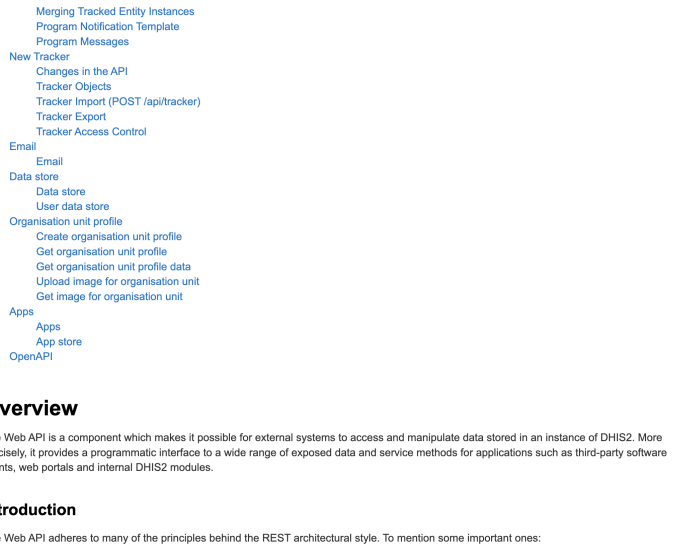

Figure 2.10: A snapshot of parts of the table of contents and the narrative start with introduction.

2.5.3 Web API documentation

The DHIS2 web API documentation is a detailed guide that provides information about the API utilized by DHIS2. The web API enables programmatic interaction between developers and DHIS2, allowing them to retrieve, create, update, and delete data in DHIS2 (DHIS2 Team, 2023e).

Documentation for the DHIS2 web API typically includes information on API endpoints, which are URLs used to access specific DHIS2 functionality. In addition, it contains information regarding the query parameters, request and response formats, authentication methods, and error handling. It also includes data elements, data values, metadata, indicators, data sets, users, and permissions, among other facets of DHIS2. The DHIS2 web API documentation assists developers in understanding how to programmatically interact with DHIS2 and use the web API to interact with DHIS2 data and metadata.

The API documentation is extensive and technical, intended to provide developers with in-depth information on web APIs and programming concepts. The complexity of the DHIS2 web API documentation is due to the nature of the DHIS2 platform, which is a robust health information system with numerous configurable options and modules. As a result, the documentation may cover a broad range of topics and employ difficult-to-understand technical jargon. Naturally, as a consequence of the vast amount of information and features that must be provided to developers, the documentation becomes densely populated with a large number of interactive areas and multiple needs for content navigation.

Some observations that can be made on the first glance of the documentation is the use of three navigational elements (see figure 2.11); two sidebars and one top navigation bar. The navigation bar to the left provides macro-navigation between fundamentally different implementation and development topics, while the navigation bar to the right provides micro-navigation on the topic that is selected from the navigation bar on the left. The top navigation bar enables navigation between the different intentions a developer might use the documentation for; information on the API in use, implementation of the API and development of the DHIS2 software (DHIS2 Team, 2023e).

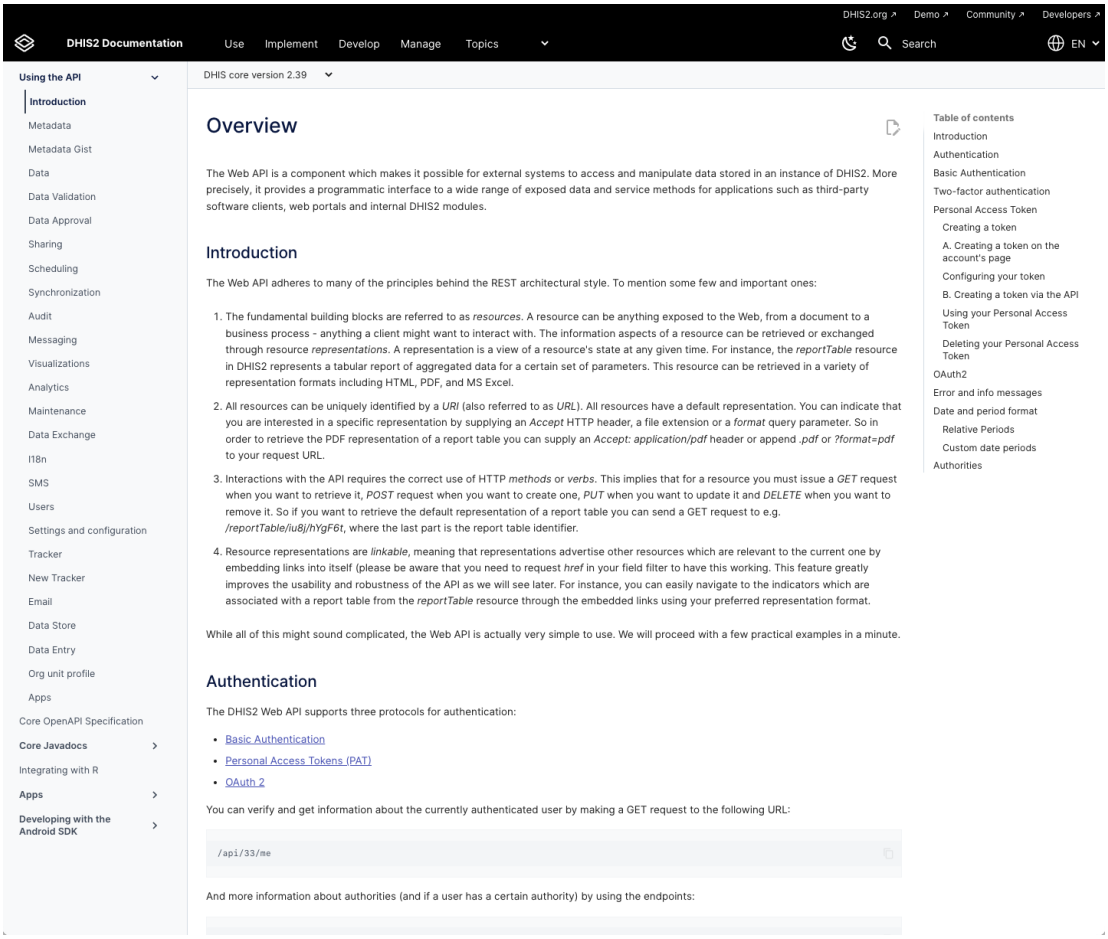

Figure 2.11: Snapshot of the introduction page for the DHIS2 web API documentation (develop section).

The home page of the documentation provides more information on metadata, tutorials, and the option to view the documentation as a single page (see figure 2.12). The metadata section provides in-depth, mostly textual, information of specific metadata libraries like nutrition, HIV, immunization and so on. However, abstractions of the metadata design of the API core are not provided.

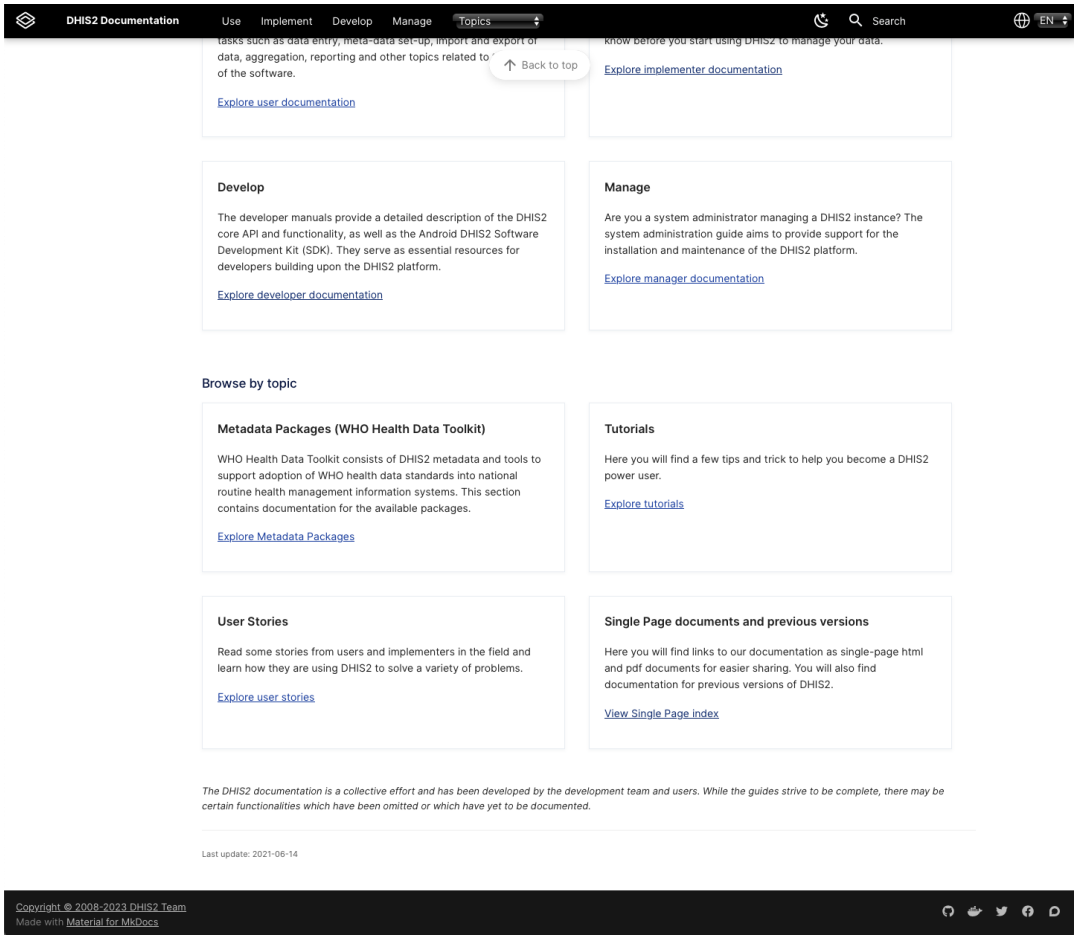

Figure 2.12: Home page of the DHIS2 documentation website which provides topics on metadata, tutorials and single page viewing.

2.5.4 DHIS2 Academy

The DHIS2 Academy is an online learning platform that provides training and educational resources for organizations and individuals using DHIS2 (DHIS2 Team, 2023f). The DHIS2 Academy's mission is to support the global DHIS2 community by providing high-quality learning materials and resources that assist users in better comprehending and utilizing the DHIS2 platform. The DHIS2 Academy supports the global DHIS2 community by providing educational resources and encouraging collaboration and knowledge-sharing among DHIS2 users and developers in their web forum (DHIS2 Team, 2023f).

The DHIS2 Academy offers a variety of courses, tutorials, and webinars on system administration, data analysis, and program management, among other DHIS2-related topics related to application development and the API (DHIS2 Team, 2023d). The DHIS2 Academy's materials are designed to be accessible to users of all experience levels, from beginners to advanced users. In addition to providing learning resources, the DHIS2 Academy serves as a hub for the DHIS2 community by providing a forum for users to

connect, share best practices, and collaborate on new projects and initiatives. The platform also recommends learning paths for building a pathway to build DHIS2 skills (see figure 2.13). The purpose of this resource is health information related capacity building and data analysis, and in addition to providing basic introduction to platform application development training and API implementation.

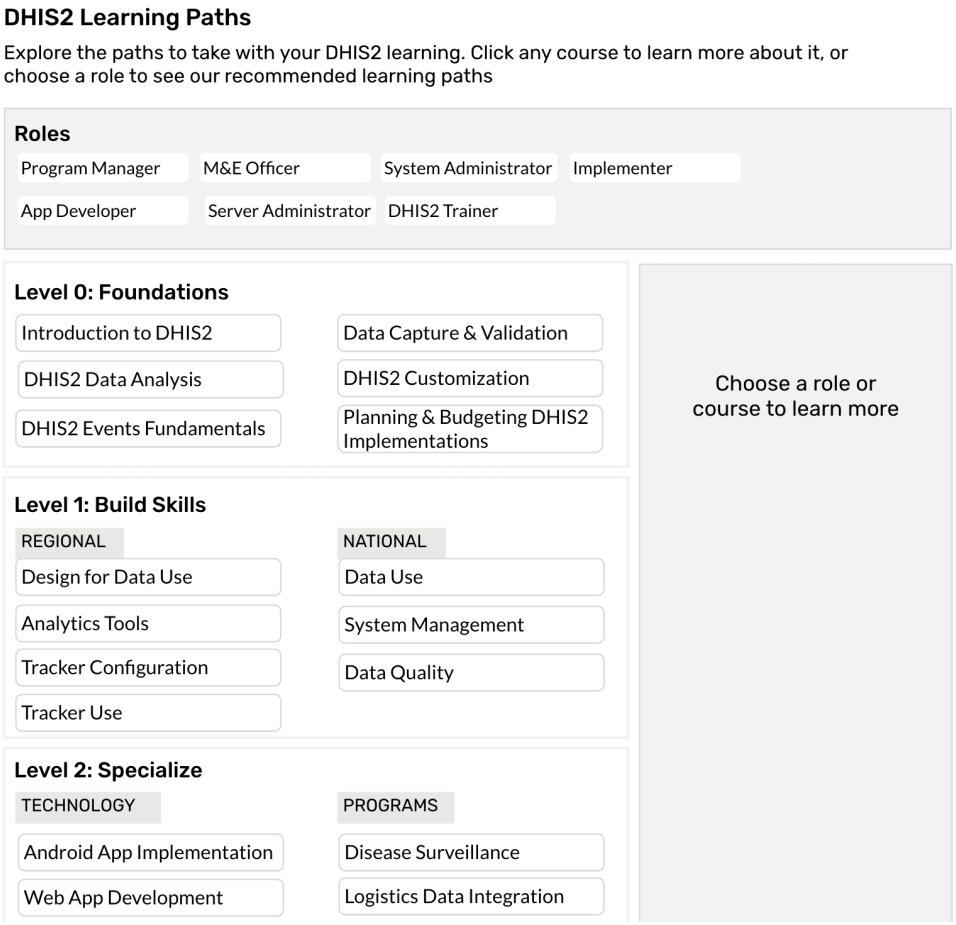

Figure 2.13: Snapshot of the interactive diagram used to identify the learning paths.

3 | Literature review

This chapter is a summary of a more in-depth analysis of the relevant literature. Key concepts relevant for the analysis will be explained in this chapter.

3.1 Learner-Centered Design (LCD)

This design philosophy puts 'learners' at the core of design inventions, addressing their needs in order to effectively provide support in learning (Soloway et al., 1994). It concerns how educational learning material is designed to improve knowledge sharing and growth in order increase the probability for the students to retain and apply this knowledge (Abdelmalak & Trespalacios, 2013; Ardito et al., 2006). This theory applies additional focus on user tasks and goals of using a tool to consider the ways in which it may improve the learning circumstances. This can for example be trying to understand a target groups' goals, motivations, and diversity (Soloway et al., 1994).

The main aspect of this approach is to address the power-balance between instructors and students and sharing the responsibility of learning between the two (Abdelmalak & Trespalacios, 2013). It is a participatory approach that involves giving the students a voice in decision-making that involves their learning and promotes their educational needs and interests. By using this approach, their motivations to learn may increase, which in return may facilitate knowledge growth.

This theory provides a perspective that concentrates on understanding learning material through factors such as motivation, performance and memory workload. For the purpose of this study, these metrics will be associated with the usability principles learnability, efficiency and memorability. The philosophy will be applied overarchingly to this research to examine how these metrics affect the API learning experience.

3.2 Usability principles

Usability principles are criterias that are used to assess the quality of an artifact in order to determine its success or failure on market level, with the goal of making it more accessible for more users in order to increase adoption rates (Mvungi & Tossy, 2015). The usability principles can, in the context of design, be understood as practical guidelines for designing interfaces (Lee et al., 2010) and the attributes it conveys are different depending

on the perspective used to examine the artifact (Jeng, 2005). There are several attributes that determine the usability of an artifact, and the ones of importance for this study are learnability, efficiency and memorability. Each of these attributes have an impact on the perception of learning material and need to be balanced in order to increase the success rate of knowledge transfer for learners.

3.2.1 Learnability

As previously stated, one of the characteristics of the usability principles is the artifact's learnability. The learnability of a system is measured by the time required for a novice user to acquire the knowledge necessary to become proficient in a desired task or topic (Joyce, 2019; Kapenieks, 2013). Learnability aids with quicker onboarding processes and the learner gaining greater confidence in their abilities as they quickly master new systems and achieve optimal task performance (Joyce, 2019).

In addition to the difficulty of achieving a balance between learnability and efficiency due to the contrasting needs for depth and simplicity, addressing the learnability attribute is complicated by the divergent needs of novices and experts. Although novices desire rapid comprehension to achieve satisfactory levels of performance through frequent hand-holding, it is not necessary for more advanced students (Joyce, 2019). For experts, on the other hand, quick task completion times are desired, necessitating process shortcuts to avoid repeated inconvenience (Joyce, 2019). Therefore, it is important to strike a careful balance between the attribute and the artifact's efficiency, as the two have a close relationship and may sometimes necessitate opposing requirements for the attributes to be optimal (Joyce, 2019).

3.2.2 Efficiency

This attribute describes how quickly and easily the user can complete their tasks (Kapenieks, 2013). An efficient artifact would enable the user to complete a task in a short amount of time. It is essential to comprehend efficiency in order to grasp the time and effort required to master core academic competencies (Hoffman & Schraw, 2010). In the context of education, efficient learning processes are those that allow students to acquire knowledge rapidly and with minimal effort. For effective learning materials, concise and accurate information delivery to the user can be the common denominator. Hence, there is a connection between learning material that is easy to understand and quick to use; in other words effective learning resources and processes.

Hoffman and Schraw (2010) delved deeper into the elaboration of what 'efficiency' would entail, as comprehension of efficiency would aid in boosting educational productivity.

They define two core concepts that influence how efficiency is perceived: performance competency and the ability to perform a task with minimal effort or time expenditure (Hoffman & Schraw, 2010, p. 2). Additionally, Hoffman and Schraw (2010) also touch upon the concept of "instructional efficiency," which is the ability to achieve educational goals with minimal time investment. These theories imply that, in order to improve the learnability of a learning resource, it will be necessary to develop materials that efficiently convey knowledge and require minimal time to use.

3.2.3 Memorability

The characteristics of the memorability attribute is reestablishing proficiency after longer time periods of not addressing the topic (Nielsen, 2012). In the case of learning material, memorability would refer to knowledge retrieval from the long-term memory after longer time periods of inactivity and engagement with the knowledge. Sufficient memorability ensures that knowledge transfer was successful from the learning material and that the users have high retention rates of how to use the material. While learnability concerns first time encounters with the material, memorability concerns returning users.

3.3 Cognitive load theory

Cognitive load theory associated with instructional design principles emphasizes the limited capacity of working memory which affects learning effectiveness and performance (Blayney et al., 2015; Chen & Lin, 2016), and can assist in designing learning material that optimizes intellectual performance (Chen & Lin, 2016). Cognitive load has an impact on the working memory and the processing of information. This can be controlled by presenting information in different styles and in different quantities (Bannert, 2002, p. 140). There are three general categories of cognitive load; intrinsic load, extraneous load, and germane load (Chen & Lin, 2016):

- **Intrinsic load** is essentially the degree of difficulty of the subject at hand which cannot be changed through instructional treatment (de Jong, 2010). It is necessary to optimize the load by dividing the subject into smaller pieces as the intrinsic difficulty of the content cannot be altered.

- **Extraneous load** is workload that does not directly affect learning efficiency and performance, but exists from the information design (de Jong, 2010). Examples of this load can be conventional practice problems, coordination of different resources, finding relevant information or the likes. This workload affects schema construction (de Jong, 2010), which is a useful tool for quicker learning.

- **Germane load** is processes that involve actions such as interpretation, exemplification, classification, differentiation and organization or the likes (de Jong, 2010). In other words, the germane load is the workload on processing of information gathering and comprehension through schemata. Steep changes in germane load can have the adverse effect of overwhelming the learners.

The overall objective of using cognitive load theory is to design instructional material that optimizes the intrinsic load (van Gog et al., 2011), increase the germane load, and decrease the extraneous load in order to promote effective learning experiences (Costley & Lange, 2018) for novice learners. Since extraneous load is necessary for the formation of initial problem-solving schemata, novice learners cannot eliminate it entirely. If the learner has little prior knowledge and the material is intrinsically difficult, it may have a negative effect on the learner (Costley & Lange, 2018, p. 70). Therefore, it is essential to customize instructional procedures and formats to the learners' prior knowledge (Blayney et al., 2015). Optimal learning occurs when the learner has optimized intrinsic and germane load. Due to their limited prior knowledge, novice learners may need to prioritize minimizing extraneous load.

However, experienced learners may be able to handle greater levels of extraneous load while optimizing intrinsic and germane loads to accommodate their existing knowledge and skills (Costley & Lange, 2018). This may involve increasing the intrinsic cognitive load to present more complex material, the germane cognitive load to encourage reflection and application of existing knowledge, and the extraneous cognitive load by reducing redundant or irrelevant information. Therefore, the learning material needs to consider the target audiences and optimize the cognitive loads to benefit both parties.

3.4 Worked examples

Worked examples are instructional materials that present a problem and its detailed solution. They are frequently used to instruct students because they provide a clear and structured model of how to solve a given problem (van Gog et al., 2011). The worked examples are frequently employed as part of a broader instructional strategy that involves providing students with both worked examples and problems to solve independently. They are particularly useful for novice learners, as they prevent the use of ineffective problem-solving strategies, allow learners to focus on the relationship between problem states and operators, and facilitate the development of a cognitive schema for solving such problems (van Gog et al., 2011).

These examples are intended to reduce the cognitive load experienced by learners while attempting to acquire a new skill or solve a problem (van Gog et al., 2011). By providing learners with a worked-out solution procedure to study, it is envisioned that they can devote more of their cognitive capacity to constructing a cognitive schema for solving similar problems and to decrease the extraneous cognitive load (van Gog et al., 2011). By reducing extraneous cognitive load, students can process and retain relevant information more efficiently, leading to improved learning outcomes (Costley & Lange, 2018). The worked examples may therefore be a way to to ensure that cognitive load is not increased through, for example, providing redundant information.

3.5 The expertise reversal effect

The expertise reversal effect occurs when instructional methods that are effective for novices become ineffective or even detrimental for experts, resulting in a decline in learning outcomes as novices become experts (Kalyuga et al., 2003). Blayney et al. (2015) explains that the expertise reversal effect is tied to the cognitive load theory and that it highlights the effects differences in prior knowledge has on the effectiveness of instructional strategies. They note that even though novice learners gain better learning output from significant instructional guidance, the more experienced individuals should be provided with less instructional support in order to use their own schemata for their problem solving activities (Blayney et al., 2015). To optimize cognitive load as learners advance in a task domain, instructional guidance should be provided at the appropriate time and unnecessary support should be removed (Kalyuga, 2007).

3.6 Layer-cake scanning method

The layer-cake scanning method is a technique for quickly locating relevant information on a website (Pernice, 2019a). The method consists of a series of layers or stages:

1. Scan the headlines and bolded text to gain a general understanding of the content.

2. Scan subheadings or bullet points to identify vital information.

3. Search highlighted or emphasized text for pertinent information or keywords.

4. Search the entire text for any remaining details or context.

The method is intended to assist users in efficiently scanning content and locating pertinent information, while allowing them to bypass less important or irrelevant information.

3.7 Inquiry-based learning

Inquiry-based learning is an instructional method in which students acquire knowledge through an active process of inquiry, comparable to the practices of professional scientists (Pedaste et al., 2015). The student develops hypotheses, tests them through experimentation or observation, and identifies new causal relationships. This strategy emphasizes the learner's active participation, self-directed learning, and responsibility for the development of new knowledge.

The learning method emphasizes the importance of active learning and material engagement. By constructing their own knowledge through inquiry, students are better able to manage their cognitive load than if they were passive recipients of information (Pedaste et al., 2015). Inquiry-based learning entails the application of problem-solving skills (Pedaste et al., 2015), which can reduce cognitive load by focusing attention on relevant information and strategies. In addition, breaking down complex scientific processes into smaller, logically connected units can help students manage cognitive load by reducing extraneous cognitive load and focusing attention on crucial aspects of scientific thought in this inquiry cycle (Pedaste et al., 2015).

3.8 Studies of pedagogical approaches

Falkner and Sheard (2019) presented several psychological and pedagogical viewpoints on learning and growing one's knowledge in their study. These perspectives have been applied in computer science knowledge domains in order to evaluate teaching methods. The most significant takeaway is that pedagogic techniques must be reviewed not just in terms of the real-world setting in which they will be used, but also in terms of the dynamic student population and emerging technologies that may affect teaching methods. Individual evaluation is used in some pedagogical models, whereas community learning is used in others. Students' involvement and enthusiasm in blended learning strategies may rise, while reluctance to participate in flipped situations may impair student attendance. In computer science disciplines, active learning methodologies have been used the most since they stimulate more reflection, whether individually or in groups. Below are some interesting topics that are relevant to this study.

3.8.1 Collaborative and cooperative engagement

- Techniques like pair programming and studio-based learning, for example, can help students acquire programmatic logic faster by allowing them to reflect on and correct their logical understanding of the challenges at hand (*Collaborative*).

- Students may have an easier time comprehending theory-based curriculum by group discussions or having the opportunity to reflect on each significant step in the process. In discussions with others, students are encouraged to critically assess the knowledge they have received and correct any misconceptions they may have (*Cooperative*).

- Gamifications, media computation, and physical programming devices could all be used to boost student motivation and engagement.

3.8.2 Abstraction / "blackboxing"

It is critical to be aware of and understand the needs of students in order to make the learning process more engaging and fruitful. The overall goal for Computer Science students is to learn how to write programs that produce a specific result as well as to comprehend what an existing program will do. Mental models and representations are tools for better comprehending and interpreting complex logics and computer systems. Abstraction to a degree ("blackboxing" - understanding the process without necessarily needing to comprehend the entire rationale) may be a factor in improving learning success.

3.8.3 Interdisciplinary importance

This study also sheds light on the overall mechanisms that underpin learning processes. This kind of knowledge could be extremely useful when working with educational materials for students and new API users. Knowing the fundamental elements of educational pedagogic theory can help raise the chances of success and make the process of achieving the end goal easier.

4 | Research approach

This chapter explains how the data was collected and analyzed in detail and its relationship with theory and literature in order to address the research question. The data collection and analysis in the first and second iterations were conducted as a joint effort with Johannes Skøien.

4.1 Philosophical assumptions

To decide an appropriate methodology, it is necessary to highlight the ontological and epistemological stances adopted in this study. These perspectives influence the approach taken to answer the research question, as well as the research methods, data collection strategies, and analysis strategies. This study assumes a constructivist ontological stance and interpretivist epistemology. A constructivist ontological perspective is essentially a relativist position that assumes realities are social constructions and implies there are multiple subjective definitions of reality (Mills et al., 2006); The perspective involves co-constructing meaning between the researcher and the participants (Mills et al., 2006, p. 26). An interpretivist epistemology is a subjective position that involves attempting to understand phenomena through the meaning that people attribute to them and focuses on the complexity of human sense making as the situation emerges (Myers, 1997).

4.2 Methodology: Constructivist Grounded Theory

The study assumes a constructivist research paradigm since it seeks to comprehend the factors that contribute to a perceived problem and suggests solutions (Verne & Bratteteig, 2018). Consequently, the research question requests an investigation of an issue, and the data will be collected in order to examine the issue from multiple perspectives to gain an understanding of the phenomena (Verne & Bratteteig, 2018). A constructivist grounded theory entails that the researcher "constructs theory as an outcome of their interpretation of the participants' stories" (Mills et al., 2006, p. 32). As the research positioning suggests, the methodology places emphasis on that reality is subjectively constructed and that knowledge is constructed through interpretation of experience and context. Hence, a grounded theory study with a constructivist paradigm investigates how participants construct their own understanding of the phenomenon being studied and how their experiences and interactions with the world shape this understanding. The interpretive epistemology assists with comprehending how participants interpret their

experiences and how social and cultural factors influence them. The methodology implies a favoring of qualitative approaches rather than quantitative in order to gain subjective perspectives.

4.3 Style of involvement

The researcher's role during the study will severely impact the way the methods are adopted and the findings of interpretive studies (Crang & Cook, 2007; Walsham, 2002, 2006). Crang and Cook (2007) advise adopting a role that meshes with the participants' values and behavior without compromising their own in ethnographic research. Walsham (2002) also implies that close investment is beneficial for in-depth access to people, issues and data in the field. Also, adopting a role as a 'neutral observer' may make participants not perceive the researcher as aligned with particular individuals or groups within the organization (Walsham, 2006); the participants may be more earnest and open to expressing their opinions and feelings if that is the case. It is also important to strike a balance between passivity and over-direction when collecting field data (Walsham, 2006). On the basis of a closely invested "neutral observer" role, one would want a position in the field studies that is neither dominant nor passive. The design of the study requires somewhat active participation to address the research question.

4.4 Target group characteristics and sample size

The purpose of this study is to gain an understanding of the factors that contribute to the design of learning materials that are self-learning, beginner-friendly, and effective for professional developers. Therefore, the target audience for this study must also reflect the various levels of expertise that the documents must address. As suggested by the literature review, the vast majority of API documentation-related research is conducted with the participation of experienced developers. In the context of DHIS2, the experienced developers could provide valuable insight into overcoming learning obstacles from their own journeys. In addition, as the research focuses primarily on entry-level developers, it is essential to include them in the study in order to understand the obstacles they face and the strategies they employ to determine what learning strategies they deem effective.

As the study began with the IN5320 course at UiO, it will be relevant to include students who participated in the platform application development project. In addition, other students with limited experience developing API-based applications are also relevant. From the more experienced side, it will be appropriate to include instructors and seasoned developers who are familiar with application development that includes API implementation and utilization. Together, they portray the demographic of this study, and they

are contacted personally or via e-mail. As the DEDICATED project aims to coordinate multiple geographical contexts, it will also be necessary to contact application developers with experience in other countries. Specifically, students from the University of Malawi (UNIMA) or DHIS2 developers from that region may be eligible to participate in the study. More detailed descriptions of the sample groups are provided for each iteration of the study.

The following table briefly illustrates the two iterations, the research methods and sample groups of the case study:

Iteration	**Data collection method**	**Sample group characteristics**	**Number of participants**	**Experience and DHIS2 affiliation**
1	Qualitative Interview	Academic professors	4	Varying industry experience, not affiliated with DHIS2
1	Qualitative Interview	DHIS2 Core Developers (IT consultants)	2	Moderate industry experience
1	Qualitative Interview	UiO Informatics Student	1	Limited development experience, limited affiliation with DHIS2
2	Participant Observation	DHIS2 Developers (UNIMA)	12	Moderate to expert level industry experience
2	Qualitative Interview	DHIS2 Developers (HISP Norway + UNIMA)	5	Moderate to expert level industry experience
2	Qualitative Interview	IN5320 Students	3	Limited development experience, limited affiliation with DHIS2

Table 4.1: The different sample groups for the case study (iteration 1 and 2).

As shown in the table, the participants' backgrounds, expertise, and affiliation with DHIS2 varied. For the interpretive case study, there were 18 participants in total from the interviews. According to reports, the optimal number of qualitative interviews for single case studies is between 15 and 30, whereas grounded theory qualitative studies should aim for between 20 and 30 interviews (Marshall et al., 2013). In addition, the reports assert that the number of interviews conducted is substantial in grounded theory studies when additional interviews fail to generate new insights (Marshall et al., 2013); when the research has reached theoretical saturation. Therefore, it can be argued that the

sample size is satisfactory when the number of qualitative interviews for this particular study is between 15 and 30, which is the case for the sample size of this study.

4.5 Research methods

Each iteration of data collection and analysis methods used in this research will now be presented. The research utilized qualitative research methods as they are designed to help researchers understand people and the social- and cultural contexts within which they live (Myers, 1997). The style of involvement in the study implies somewhat active participation in the data collection process. The qualitative methods and techniques which suit the style of involvement are semi-structured interviews, participant observations and literature reviews. These methods and techniques will contribute to the theoretical sampling for the grounded theory process. This section highlights the grounding in the choice of research methods, while in-depth clarifications are provided based on the different iterations of the research as these methods were used in conjunction with each other.

Research method choice considerations

The prerequisites for the selection of appropriate research methods are moderate levels of participation and the interpretive nature based on the epistemological positioning. As the study also assumes a perspective of subjective realities, qualitative, in-depth research methods are the most appropriate. Additionally, consideration of the fact that the theory is formulated based on an initial case description is also necessary. As a consequence, the qualitative research methods that are appropriate to conduct is interpretive case study, with qualitative interview and participant observation as research techniques. Additionally, integrative literature reviews will be a necessary part of the theoretical sampling process to evolve the theory in constructivist grounded theories.

4.5.1 Interpretive case study

This study's primary research method is an exploratory multiple-case study. As a result of the interpretive paradigm of the study, the subjective realities of the various sample group characteristics are also interpreted. A qualitative case study facilitates exploration of a phenomenon within its context by utilizing multiple data sources to reveal multiple facets of the phenomenon (Baxter & Jack, 2008). The exploratory multiple-case study design is especially useful for achieving the exploration of multiple perspectives of a phenomenon with no clear single set of results from a demographically diverse sample group (Baxter & Jack, 2008). In the first and second iterations of the data collection and analysis process, the cases in this study are collected, documented, and analyzed. The case description is provided in the preliminary study section (see chapter 5).

In Information Systems (IS) research, interpretive case studies are employed as part of an iterative data collection and analysis procedure, with qualitative interviews serving as the primary data gathering technique (Walsham, 2002). This results in 'thick descriptions' of the studied phenomenon, at the expense of the generalizability of the studies (Walsham, 2002). Consequently, the process of theory generation is obstructed in situations where generalization is crucial. Since the methodology is constructivist grounded theory, subjectivity and contextual factors are more important than generalizability for theory generation.

Data gathering techniques

Qualitative interviews and participant observation were conducted as part of the data gathering process for the multiple-case study. As the structure and content of the interviews were altered after each iteration of the research, specific elaboration will be contributed under the explanations of the data collection and analysis process.

Qualitative interviews

Interviews are data collection techniques for gathering intel from conversation with varying degrees of involvement and structure (Crang & Cook, 2007), and can be utilized to gain subjective perspectives on the conversation topics. In order to ascertain thick descriptions, it is wise to use an interview structure that enables explorative conversations. The most suited variant is therefore semi-structured interviews which provide broad parameters for discussion (Crang & Cook, 2007). Alternatively, the study could have benefited from a focus group technique which enables multiple people to elaborate and discuss their experiences and thoughts about the research topic (Crang & Cook, 2007). However, interviews were preferred over focus groups in order to reduce group dynamic setbacks such as individuals overpowering conversations and avoiding uncomfortable environments for individuals who do not know each other beforehand. Additionally, interviews aid in giving the participants an arena to fully express their own thoughts by themselves.

Participant observation

A brief participant observation was conducted during the second iteration of the study to gain a sense of the collaborative working culture in a different geographical context. The participant observation aided in comprehending the ways of the community through both verbal and non-verbal communication (Crang & Cook, 2007); in this instance, the developers from UNIMA. The observation lasted for a span of four consecutive days and demonstrated the collaborative learning practices through group discussions that are mentioned more detailed in the qualitative interviews of the second iteration.

4.5.2 Integrative literature review

Literature reviews generate new knowledge about a subject through the revision, critical reflection, and synthesis of representative literature on the subject (Torraco, 2016). It aids in conceptualizing literature findings and enhances the validity of grounded theory research. After each method of data collection, integrative literature reviews were conducted iteratively to critique and reflect on the findings. This helps to generate new frameworks and perspectives on the topic by revealing patterns and casual relationships spanning the research (Torraco, 2016). It also aids in comparing and contrasting the insights gained from the data collection process with emerging research topics. Hence, performing literature reviews iteratively contributes significantly to the theory generation process that comprises the constructivist grounded theory research approach. For this study, two rounds of literature reviews were conducted, the first following the preliminary interview and the second following the interpretive case study as the third and final iteration of the research process. The emerging topics, codes, and code-categories from the process of data analysis encourage the exploration of new themes, and the literature reviews were conducted to address these newly emerged themes.

4.5.3 Qualitative data analysis methods

The aim of grounded theory is to derive a theory grounded in data by moving from basic descriptions, conceptual ordering and theorization (Walker & Myrick, 2006). To provide a starting point and identify relevant themes for the upcoming interviews, a loosely approached thematic analysis of the preliminary studies was performed. The results of this analysis were then used as a basis for understanding and analyzing the multiple-case study, which employed a grounded theory analysis strategy. The preliminary investigation revealed themes associated with API application development, API documentation, and application development courses. These themes were used to generate interview questions, breadboards, and concepts aimed at explaining API documentation obstacles and possible solutions.

Corbin and Strauss (1990) define three procedural analysis concepts that are relevant for grounded theory approaches: open coding, axial coding and selective coding (Preece et al., 2015). Following the thematic analysis in the preliminary study, these analysis concepts were used to facilitate the theorization process in subsequent iterations of the research process, with iterative modifications. In these coding processes, affinity diagramming in Miro was used to visualize the identified patterns and themes. The analysis methods will be elaborated further in their respective iterations because they either introduced new thematics or improved existing ones.

4.6 Data collection and analysis procedure

This section offers comprehensive clarifications of the methods and techniques used in the study for the various research iterations.

4.6.1 Preliminary interview and literature review

Preliminary interview: case description

The preliminary interview presents a case narrative that sheds light on common obstacles encountered by students of IN5320: Development for Platform Ecosystems during their application development project. Findings indicate that developers with limited prior knowledge of API application development face greater challenges when interacting with complex APIs and limited API-related learning resources. The views expressed in this interview influenced the data collection for this study.

Participant background

The participant is a student at the University of Oslo with moderate experience gained through project work, student courses, and degree curriculum. They acquired knowledge and experience with both front-end and back-end programming languages through these courses and independent study. Their exposure to DHIS2 was gained through the student course IN5320, which is a combination of theoretical lectures and a student project involving the creation of a simple health platform application. In addition to YouTube and consulting with teaching assistants, the participant utilized the DHIS2 self-paced learning course and API documentation for his project work (which they claim to have played a big part of their learning experience). The participant claims that without these additional resources, learning application development would have been impossible. Prior to the course, the participant had begun to study front-end development via the online learning platform Skillshare and by reading theoretical books. YouTube, according to the participant, has been the most useful resource for learning system development in terms of understanding why things occur and how to apply the various concepts.

Design of the data gathering procedure

As part of the preliminary research, a thorough literature review was conducted to establish a knowledge base on API learning, good examples of API documentation, factors affecting user experience, and pedagogical approaches to learning difficult subjects. The review of relevant literature provided a summary of the current state of knowledge in the field. It enabled the identification of key concepts, theories, and findings and the contextualization of the research within the larger academic discourse of the field. In

addition, the literature review allowed for the identification of research gaps and areas requiring further study. It revealed areas that have not been thoroughly investigated or where contradictory findings or viewpoints exist, highlighting the need for this research to add to the existing body of knowledge in the field.

Integrative literature review for theoretical sampling

As part of the preliminary research, a thorough literature review was conducted to establish a knowledge base on API learning, good examples of API documentation, factors affecting user experience, and pedagogical approaches to learning difficult subjects. The review of relevant literature provided a summary of the current state of knowledge in the field. It enabled the identification of key concepts, theories, and findings and the contextualization of the research within the larger academic discourse of the field. In addition, the literature review allowed for the identification of research gaps and areas requiring further study. It revealed areas that have not been thoroughly investigated or where contradictory findings or viewpoints exist, highlighting the need for this research to add to the existing body of knowledge in the field.

Breadboarding and conceptualization for discussion

Using the results of the thematic analysis and literature review, these breadboards and conceptual sketches were utilized to illustrate ideas surrounding informational structures for documentations that would enhance learnability. The artifacts were intended to be visual references of user-friendly API documentation designs that reflected the knowledge gained from the literature review for the scheduled interviews. These concepts were used to discuss variables that may influence the document engagement of newcomers.

The breadboard was constructed based on the existing flows and the layout structure on the DHIS2 web API documentation (see figure 4.1), and was used to convey to the interview participants of the complexities of finding information in order for them to propose different ways of presenting information.

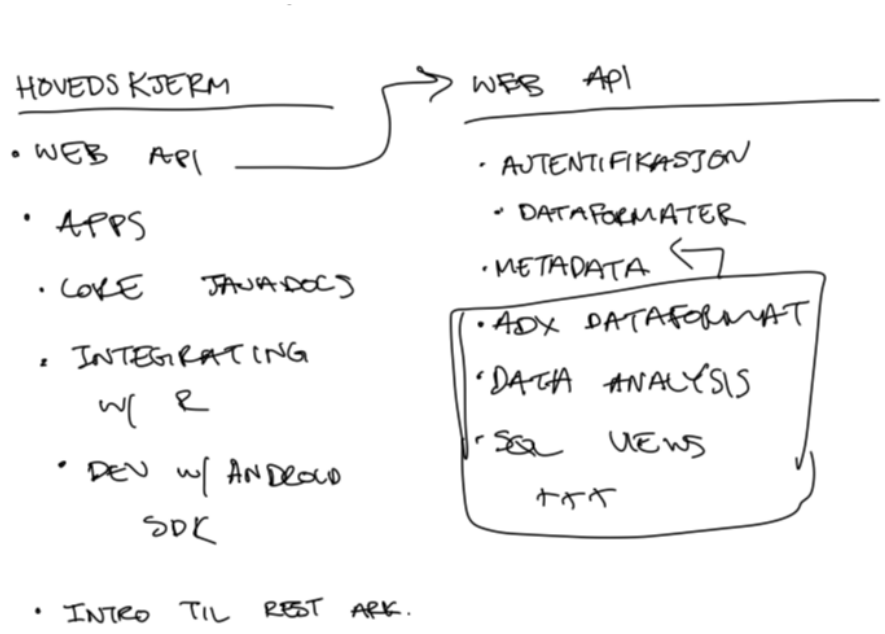

Figure 4.1: Breadboard of the imagined layout and flow of the digital interface during interviews.

Additionally, concepts based on the examples and principles in the literature review were also constructed with inspiration from preliminary studies and discussed with the interview participants. Figure 4.2 is an example concept of displaying callback functions and actions with required parameters, code examples with responses, and accepted response codes, and how brief descriptions can be paired with visualizations and abstractions to convey metadata element relationships. Figure 4.3 is the conceptualization of action descriptions with different example programming languages and architectural structure visualization.

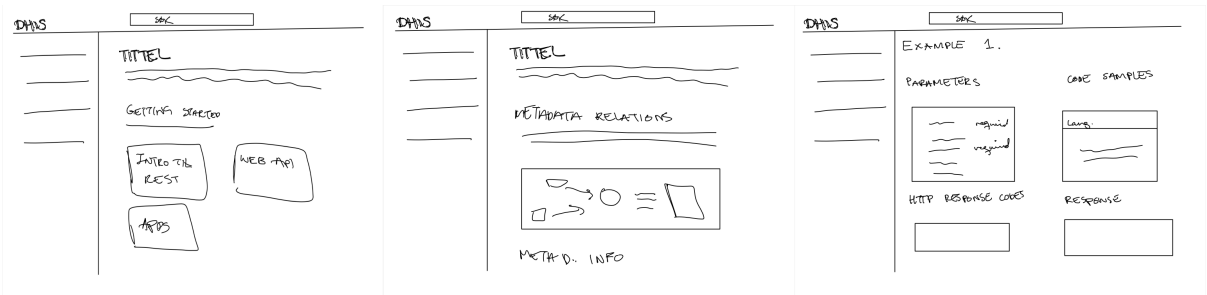

Figure 4.2: Conceptualization 1 of content presentation for iteration 1.

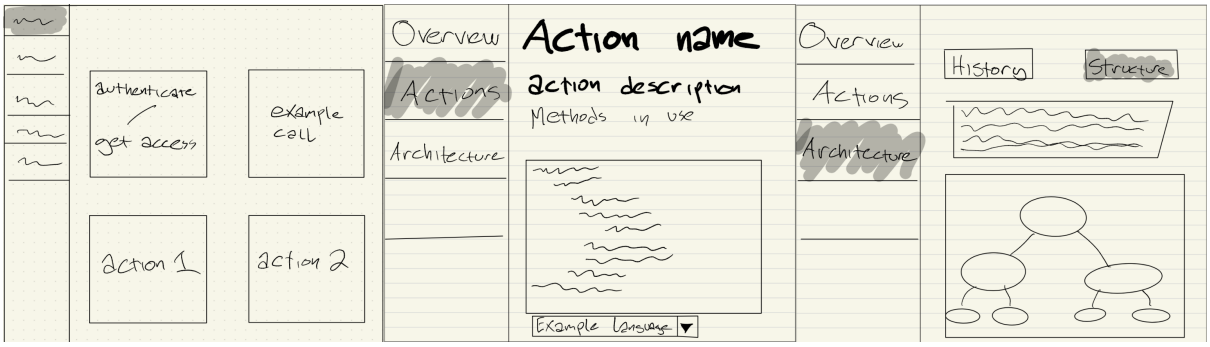

Figure 4.3: Conceptualization 2 of content presentation for iteration 1.

4.6.2 Iteration 1 & 2: Interpretive case study

Iteration 1: Qualitative interviews with academics and developers

This iteration entailed gathering data from the relevant user groups. The selected user groups consist of (1) academic professors who interact with beginning students and (2) developers who utilize the DHIS web API documentation frequently.

Sample group background

It was decided to engage with academics from a variety of institutions in order to gain insight into the design of system and/or app development courses, as well as their strategies for addressing common challenges related to student API usage. Two participants had moderate industry experience in development, one participant had substantial industry experience, and one participant had no industry experience. Even so, all participants had prior experience with various types of development documentation, allowing for a comparison between the DHIS web API documentation and other educational documentation.

Furthermore, discussions were held with three developers to learn more about their onboarding procedures and their perspectives on the challenges they faced when learning how to develop apps for the DHIS platform. Two of the participants are contracted consultants who work with the DHIS2 core's front-end or internal systems. The final participant is a master's student who is using the documentation to create a tool for DHIS API education. These participants had varying degrees of prior involvement with the DHIS web API documentation and varied circumstances in their onboarding processes, providing a means to learn more about existing onboarding obstacles.

Design of the data gathering procedure

Following the review of the relevant literature, it was decided to conduct informal interviews with other Norwegian institutions and developers who are already involved with the documentation in order to gather their diverse perspectives. Due to the academics' lack of familiarity with the DHIS2 web API documentation prior to the conversation, the study was structured as an informal discussion about API documentation, using the DHIS2 web API documentation as an example. The initial structure of the discussion was for academics and interviewers to engage in a "workshop" to generate various informational structures for documentations that would enhance learnability. However, because these discussions were held either physically or digitally based on the location of the participants, any other discussion tactics would have to be adaptive to both spaces. As a result, the workshop framework was reconfigured so that the interviewers generated concepts based on previous literature reviews prior to the discussions to use these concepts as conversation points. The concepts were used to address aspects that may affect newcomers' experiences with engaging with documents on their own.

Mode of analysis

As previously described, the case study was analyzed using open coding, axial coding, and selective coding. Codes were identified through words and sentences in the data set and then organized according to their properties for each interview. The results of the interviews and the open coding procedure were visualized using affinity diagrams. The process of selective coding revealed 'API learning experiences' as a central category for the research cycle. In the findings chapter, the results are presented in a table where the codes from the open coding process are loosely associated with the categories, as it was intended that these results would be reproduced and refined in the next iteration.

Open coding

Content, *structure*, *onboarding suggestions*, *educational structure*, *learning process*, and *API documentation feedback* were derived to be the most significant themes from the open coding process. Figure 4.4 depicts the organization and presentation of these interview notes, while Figure 4.5 depicts the open coding structure for one of the interviews.

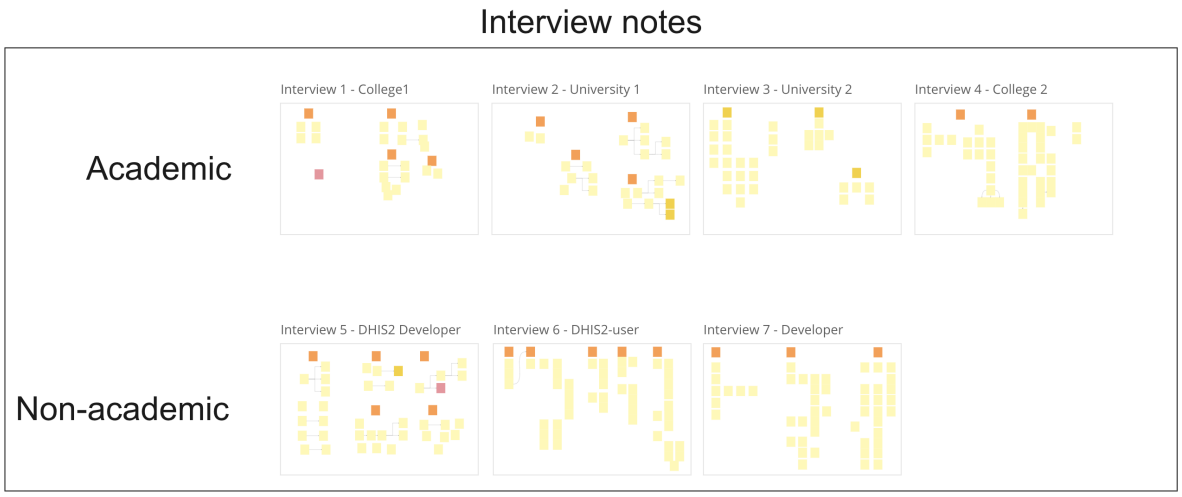

Figure 4.4: Interview notes from iteration 1 organized using affinity diagrams in Miro.

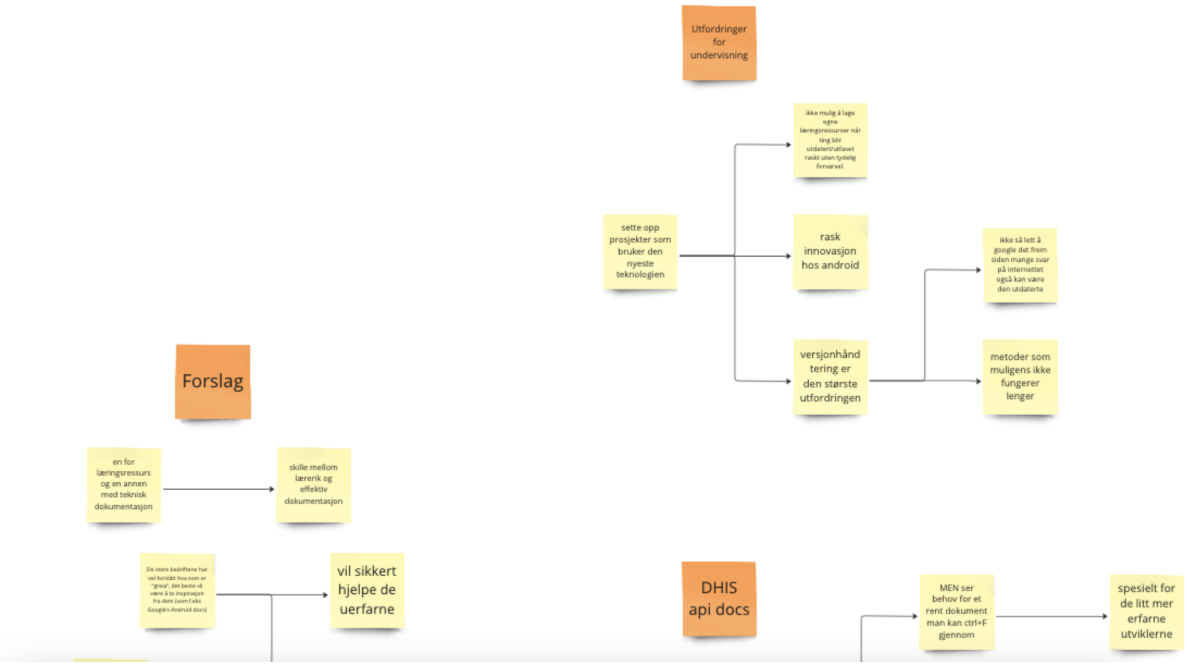

Figure 4.5: Open coding category examples from an interview, illustrating relations between statements

Axial coding

The subcategories generated by open coding corresponded to the overarching, recurring themes and observations from all the interviews. As shown in figure 4.6, these were *improvement potential*, *pedagogical aspects*, *content*, and *structure*. In the 'other tips' category were statements that did not fit into any of the other categories.

Figure 4.6: Categories constructed from open coding themes for iteration 1.

The subcategories were structured for each of these categories so as to comprehend the various perspectives of each major theme (see figure 4.7). These subcategories were refined so that they would fit within these categories.

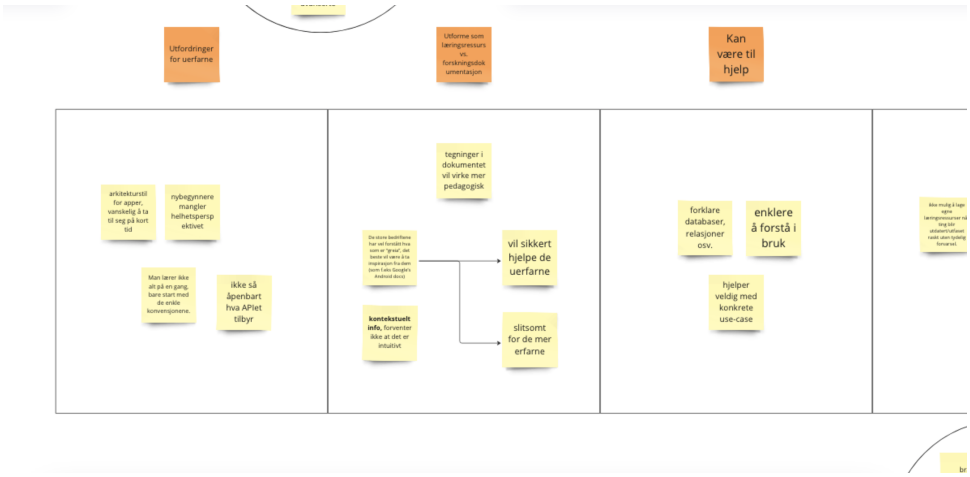

Figure 4.7: Subcategories sorted through all of the themes identified in the open coding process.

Iteration 2: Qualitative interviews with students and DHIS2 developers

To further the investigation, the study aimed to gather additional information from individuals with varying levels of experience in application development. The interview process was divided into two groups: Norwegian developers with ties to DHIS2 development and API core, and Malawian developers who create DHIS2 applications. The distinction between contexts was intended to provide a more diverse cultural perspective on learning strategies and environments.

Sample group background

For this iteration, it was determined to involve individuals with developer experience and a level of familiarity with DHIS platform application development. Five of the individuals were developers with experience in DHIS application development ranging from recent graduates to ten years. One of these individuals is from Malta, whereas the other four were from Malawi. The developer from Malta is presently working on an internal project to improve the existing DHIS API, meaning they have a direct connection to the platform's core. The Malawian developers do not interact with the core because they create applications using the API. Certain individuals also have experience with coaching other developers either in DHIS or in previous workplaces. The remaining three participants were UiO master's students enrolled in the IN5320 course where they were expanding their knowledge of platform application development through theoretical and practical engagement with the DHIS API. Two of the students were pursuing software development, while the third was a UX/UI designer. The students' levels of development experience varied, and varying degrees of professional experience.

Design of the data gathering procedure

In addition to interviews with Norwegian students, a field study was conducted with DHIS platform application developers in Malawi as part of the research. The objective of the field research was to gain insight into the learning and onboarding experience of developers and to generate findings that could be applied to a single group as opposed to two distinct target populations. The study included questions about the experiences of the participants, comparisons with other API onboarding processes they may have encountered, and problem-solving advice they wished to impart to new developers. Participants were questioned about their learning tools in order to determine the various modes of learning and the most prevalent problem-solving strategies in an unfamiliar environment.

Mode of analysis

In order to maintain uniformity, this iteration followed the same coding procedure as the previous iteration. As this iteration's interviews were conducted with a greater emphasis on pedagogical approaches than in the previous iteration, the appropriate categories and subcategories were also refined for use in this coding process. The process of selective coding revealed that "API learning and problem-solving approaches" is the primary category for this research cycle. As in iteration 1, the results are presented in a table in which the codes from the open coding process are loosely associated with the categories, with the expectation that these results will be replicated and refined in iteration 2.

Open coding

As with the previous iteration, this iteration's codes were categorized based on difficulties encountered while using the DHIS2 API documentation, suggestions for improving the existing documentation, and factors that facilitate successful documentation utilization (see figure 4.8). Newly emerged codes included problem-solving techniques, additional learning resources, project development process experiences, and suggestions for learning new solutions or APIs (see figure 4.9).

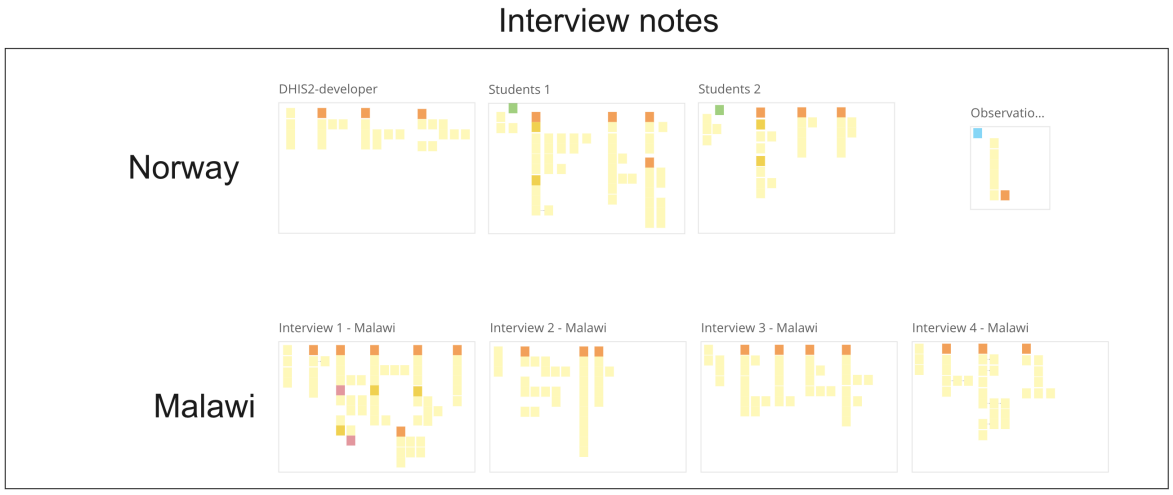

Figure 4.8: Interview notes from iteration 2 organized using affinity diagrams in Miro.

Figure 4.9: Newly emerged codes and subcategories from open coding

Axial coding

The subcategories generated through the process contributed in the generation of the categories *approaches to problem solving*, *improvements for API problem solving approaches*, and *obstacles for problem solving*. Like in iteration 1, the subcategories were refined so that they would fit within these categories (see figure 4.10.

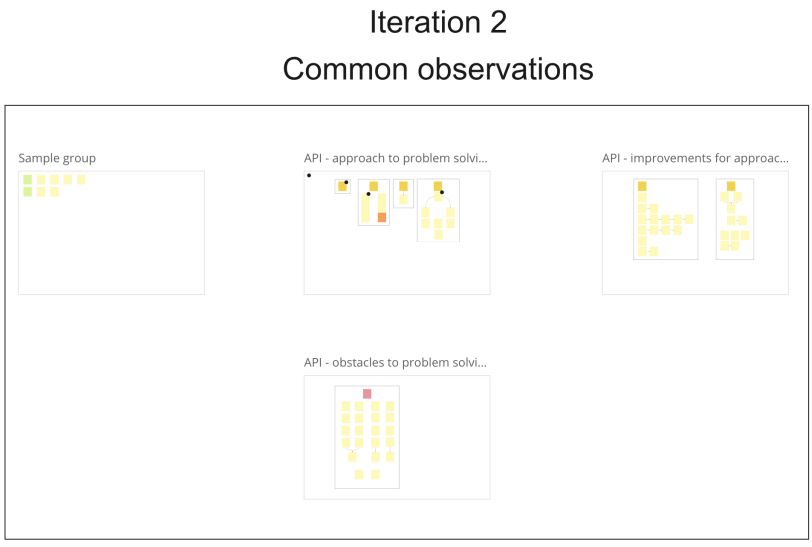

Figure 4.10: Categories constructed from open coding themes for iteration 2.

Final coding results

The results of the first and second iterations were recorded in separate tables before being merged and refined to identify a universal core category, subcategories, codes, and emerging themes for each subcategory. This was done in order to generate a comprehensive and cohesive listing of codes from both iterations, as a result of selective coding. The subcategories have been reorganized and refined so that they are consistent and exhaustive. The results of this process are presented in the findings chapter.

4.6.3 Integrative literature review

This literature review supplements the results of the coding process by offering a comprehensive analysis of the emerging themes and categories. The coding process revealed two crucial factors that can impact entry-level API learning: API learning experience and problem-solving strategies. The subcategories included topics such as usability principles, learning theories, cognitive load management, and efficiency. Additionally, problem-solving activities were investigated through cognitive load studies on learning approaches. In a second review of the literature, these themes were examined in greater depth. The findings from the literature review were then used to answer the research questions regarding the API learning experiences of entry-level platform application developers. Further, the knowledge gained from the discussion was used to develop two API learning guidelines that can aid beginning developers on their API learning journey.

4.7 Alternate research designs

This study is grounded in theoretical frameworks, but it has the potential to investigate the practical aspects of API learning experiences for novice students. This could involve participatory design approaches, such as engaging participants in a workshop to experiment with and conceptualize the usability of API documentation, using a Research through Design methodology. This approach, however, would shift the emphasis to the digital interface design of a particular type of learning resource rather than the metacognitive understanding of learning processes that can be applied to numerous types of learning resources. This alternative research strategy could complement the findings of this study and aid in the evaluation of the acquired knowledge.

4.8 Ethical considerations

Informed consent and anonymisation need to be considered as this is anthropological research (Vike & L'orange Fürst, 2020). To ensure that participants make an informed decision about whether to participate in the study, they will be fully informed about the study's purpose, procedures, risks, and benefits. It will also be necessary to inform those who provide information through semi-structured interviews that the data will be stored for approximately 6-12 months for research purposes. Audio recordings of the interviews will be made to facilitate note-taking, but the information will not be traceable to individual participants to protect their anonymity. To maintain confidentiality, it will be necessary to prevent other participants from identifying participants in their environment. Only if design proposals or artifacts are constructed and need to be evaluated by the participants will video recordings be made. Participants in the study are not required to be affiliated with DHIS2, but meetings will include representatives of the organization to ensure transparency. The findings of the study will be made accessible to all participants and relevant parties.

5 | Findings

This section provides a comprehensive overview of the data collection techniques utilized for this study, including literature reviews, semi-structured interviews, and participant observations.

5.1 Case description and initial theory statement

The case description obtained from the preliminary interview served as the basis for formulating the research theory. This was supplemented by a comprehensive literature review aimed at refining the preconceived notions that would guide subsequent data collection processes.

5.1.1 Case description: DHIS2 API learning resources

The participant believes that the purpose of API documentation is to help users understand how the API works or the syntax of callback functions, seeing as there are numerous ways to perform the same actions in other APIs. According to them, the purpose of these documentations is to highlight these differences so that users can comprehend how to use the API. The participant has a limited amount of experience using API documentations to familiarize themselves with data extraction from simple APIs, but very little practical experience using these. They expressed a high level of satisfaction with the self-paced IN5320 course, citing the simplicity of the guide and the inclusion of hands-on exercises for the various learning topics. Specifically, the participant enjoyed the interactive code snippets that enhanced their comprehension of the course material.

However, they initially found the examples demonstrating data extraction from the DHIS2 API difficult to comprehend.The reason for this was that the API responses from the provided code examples did not match what was anticipated; attempts to modify the code did not produce the expected results or resulted in additional syntax errors. When they attempted to use the DHIS2 web API documentation, the examples provided were different from those in the self-paced course, which made learning the API more difficult. They explained that their experience with the self-paced course was mixed, as some segments were well-explained and easy to understand, whereas the API-related segments were more difficult to comprehend. In addition, the participant claimed there were no additional resources available to assist. The majority of the participant's difficulties were small, focused issues, such as syntax errors that were not addressed in the documentation.

The participant asserted that DHIS2's examples and textual descriptions of the various concepts are somewhat "too general," requiring more background knowledge than that of inexperienced developers. In retrospect, the participant stated that if they had another chance to redo the project, they would have sought more guidance from collaborative lab sessions and teaching assistants, as well as come prepared with questions for these sessions, rather than relying solely on reading the API documentation. The participant also suggested that the complex API should provide more examples of API callback functions as opposed to a simple list of keywords.

5.1.2 Initial theory statement

The following theory was formulated based on the case description and the results of the initial literature review:

> It is hypothesized that entry-level developers struggle to learn APIs due to difficulties in comprehending API documentation, such as information overload that makes documentation overwhelming, "too general" textual descriptions of the various concepts, lack of adequate examples that resemble their use cases, and difficult-to-understand examples of data extraction from the DHIS2 API. It is also believed that content simplicity and the provision of hands-on practice for the various learning topics can improve learning outcomes and enhance the learner's satisfaction.

This initial understanding, however, is subject to change as the research progresses and additional data is collected and analyzed, and alternative interpretations may emerge. The emergent and iterative nature of constructivist grounded theory will permit ongoing revision and refinement of the theory based on ongoing analysis and reflexive practices.

Iteration 1 & 2: Interpretive case study

The interpretive case study was conducted to collect information from academic professors who interact with newcomer students and developers who often use the DHIS2 web API documentation for the first iteration. The second iteration involved individuals with application development backgrounds, regardless of their level of experience in the field. *These iterations were conducted as a joint effort with Johannes Skøien.*

5.2 Iteration 1: Qualitative interviews with academics and developers

This iteration reports topics from the first iteration of the data collection, including content engineering and documentation enhancements for the DHIS2 web API.

5.2.1 Content engineering

The participants indicated a tension between efficiency and comprehensibility of content organization by their differing viewpoints. The majority of the sample group stated that the quantity of content disorients and confuses readers. For instance, the metadata section of the DHIS2 web API documentation contains so much information that a separate scrollbar is required for the table of contents. This implies that there is so much information available that readers feel as though they are "drowning" and have little tools to help them find what they need. Some participants have recommended the DHIS2 Team to adopt existing documentation standards as a starting reference. Building on this rationale, the majority of participants stated that dividing the documentation into multiple pages will result in better organization to aid in locating the desired content. However, there are different viewpoints regarding how fragmented the documentation should be, as 'too much' fragmentation may impair the experience of locating information if there is too much to navigate. In other words, excessive fragmentation may prove harmful.

As part of the discussion regarding the fragmentation of the DHIS2 web API documentation, additional suggestions that may enhance the experience of the documentation's readers were mentioned. One of which was that this may be a means of concealing material that is not relevant to the case at hand. Some participants proposed, for instance, that certain REST-API related information may be presented in a separate web page in order to limit the likelihood of irrelevant information. The developers also mentioned that it could be more convenient to use text descriptions and table-format information interchangeably in order to lessen the strain produced by reading through text-only content. Despite the fact that the existing format of the documentation varies between these two formats, these participants suggested that it could be done to a greater extent. However, the academic participants indicated that more illustrations and models can increase the content's learnability. They argued that visual communication of information may be simpler to comprehend and may aid in clarifying processes that are difficult to explain in words.

Both parties of the sample group also raised a concern with the ambiguity of the documentation. Academics who asserted this emphasized the need to differentiate between

learning resources and technical documentation because, from an outsider's perspective, the greatest difficulty in comprehending the documentation's content is not being able to grasp the logic of the surrounding environment. The sentiment refers to the massive quantity of metadata present in DHIS2. Therefore, they reasoned that it may be advantageous to provide more representations of these underlying data-structures in order to increase the likelihood that readers will locate the required section of the documentation. The developers on the other hand argue that it is necessary to distinguish between API documentation and metadata documentation, as their preferred technique of work is to quickly search for the resources they need and avoid reading more text than necessary. They claimed so because they believe that after a few exposures to the text, the reader will know what the API calls perform and thus be less inclined to read the content. The different perspectives from the academics and the developers imply that the intentions of the documentation needs to be clearer in order for the readers to evaluate its relevance for the scenario they are attempting to address.

Content simplicity and depth

The content can be collected and presented on a single page, but this requires some careful considerations in order for this kind of collection to be understandable. For less experienced users, the page requires some sort of structure to make it manageable for new users. A large page filled with information faces the risk of being perceived as overwhelming, hence creating a well thought-out structure is important to avoid this. Another way to avoid an overflow of information is simply to compress everything that is not critical information for the execution of the desired task so that only the relevant information for each case is shown at a time. This could for example be achieved by only showing fundamental information about a callback function and minimizing additional information. The principle is to arrange content by importance and put less emphasis on the less important information.

A suggested solution was to separate content into collapsible items, creating sub-sections or the likes. Avoiding large collections of information reduces the risk of the documentation being overwhelming, thus possibly making it more manageable. It can also be important to consider what content is bare minimum information needed for API use and what is not, and to hide information that is not necessary for that specific use. The feeling of information overload for the user can be easier to mitigate by reducing information that is not directly relevant for the majority of cases. It was also suggested that is also important to consider the target group when designing the documentation since beginners have different needs than experienced users. Thus it is important to separate "beginner-tutorials" from more advanced descriptions.

The findings also show that while experienced users often prefer large documents with the possibility to "Control+F" (the keyboard shortcut for the 'find' command) through the document, less experienced users can benefit from a more step-by-step approach. Searching through a large document requires substantial understanding of the possibilities that the system provides, as well as knowledge about what to look for in order to find what's relevant. Naturally, novice developers may not have the same level of knowledge about these topics. Less experienced users may therefore have a need for a more practical approach, providing examples and visual components rather than plain text- and table-content in the documentation.

Community-contributive documentation

One participant said that it would be helpful if users could contribute to the documentation by making suggestions. The idea is that diverse user suggestions would provide varied circumstances and settings for API use, which would make it easier for users to adapt the knowledge to their own scenarios. This would also mean that users could contribute by developing and providing alternative methods for achieving specific goals – for example suggesting multiple ways to extract data from the DHIS2 data store. The likelihood of an individuals finding what they are searching for may be increased by expanding the quantity of suggestions to the different use cases.

5.2.2 Differing levels of expertise

The diverse sample group made it apparent that the differing levels of expertise with application development that use API introduces opposing opinions on how the API knowledge resources should be designed. Some participants also asked whether the focus of documentation was to act as research documentation or if it should address the product itself (the DHIS2 API and its associated data). As of now, the participants perceive the structure of the documentation to be ambiguous, and people hold different expectations to what they think they can learn from the documentation. The developers stated that the current structure of the documentation represents more of a research documentation than of documentation designed to showcase the capabilities of the API.

Intentions of the documentation

The sample group implied that it is unclear whether the documentation is intended to be an instructional guide (higher levels of learnability) or a functional compendium that provides a quick reference sheet with the API's capabilities (higher levels of efficiency). These differing documentation styles revealed what was most significant to the different sampling groups. From the perspective of the academic participants, there was a move toward designing the documentation as a learning resource. A few participants suggested

that the addition of illustrations and visual aids could make the documentation material more comprehensible.

A few participants also mentioned the need for additional contextual information, such as additional explanations of the metadata models. The complex data structures may not be intuitive and need additional explanations to be truly understood. Doing so would improve the API documentation and may aid novices in their learning process. However, the academic sample group expresses concern that the API documentation may become more strenuous for experienced developers to navigate, as they require less fundamental information from the documentation. The developers inevitably preferred the functional compendium-style documentation. In order to address both target groups with the same documentation, it will be necessary to make the documentation as concise and dense with information as possible for novice developers, as well as incorporating search engine capabilities for developers and other navigational shortcuts for the more experienced individuals.

Degree of difficulty

The DHIS2 developers claimed that the degree of difficulty is steep. One of the participants expressed «[the content] never hits where I am currently standing». They expresses that finding content adjusted to their level of expertise about the underlying logics of the system were either 'too hard' to grasp or 'too simple' for their use cases. They have also expressed that the data model is the most challenging component of the API to understand, and suggested that the metadata model needs more attention in onboarding and coaching efforts in order to settle new developers in the system with ease.

Some participants also suggested that the content's level of difficulty should be grouped in some way to incrementally increase from novice to expert level knowledge. A few participants noted the necessity of pacing the documentation to take more "baby steps" because the material can quickly become advanced and thus less user-friendly. This was proposed as a way to differentiate between content for newcomers and content for advanced users, and to modularize the documentation content into smaller steps that lead to mastery. They also expressed that incorporating gamification techniques could boost the content's engagement and learnability by making the distinct modules more comprehensible to the target audience.

Examples in the documentation and understanding endpoint calls

A few participants mentioned that the DHIS2 web API documentation could greatly benefit from using more examples. However, some participants expressed that examples without an appropriate use context do not yield the developers anything. For instance, if a code snippet is provided in the documentation, but its use context is unclear, then the code snippet does not contribute to understanding the operation of certain functions. When the examples provided in API documentation do not reflect plausible circumstances or common use cases, it becomes difficult to comprehend the intended message. Beyond that, adapting these scenarios to the provided examples can be crucial for some developers, as it assists them in formulating a strategy to achieve their API-related objectives.

The participants also suggested that endpoint calls need to be clearer regarding their capabilities. To achieve this, they propose that the documentation should offer details about mutable variables, as well as examples illustrating how data changes in response to variable adjustments. Details about the different callback variables are provided, for example under the metadata identifier schemes section (see figure 5.1), but they do not provide example callbacks that illustrate how the endpoint result may look and how to read the values. A major challenge for students enrolled in IN5320 is their inability to comprehend the endpoint results because the callback responses needs multiple linked requests. This requires deciphering and understanding the various request parameters in order to comprehend the meaning of the JSON object values. The interconnectedness of metadata in the DHIS database presents novice developers with an additional challenge, making comprehension even more difficult.

The interconnectedness can be understood through for example the process of retrieving information about a specific disease in DHIS2. The first step involves identifying the organization unit (orgUnit) ID, which represents the geographic location or administrative unit corresponding to the data. data set ID and desired search period must then be specified. With this information, the dataValueSet endpoint of the DHIS2 API can be used to retrieve the data values for the desired data set. To retrieve the number of Malaria cases reported in the "Hargeisa" region of Somaliland during the year 2021, the orgUnit ID would be "O5Ov9cWET1c", the data set ID for Malaria cases would be "dDwZmeSPyZa", and the period would be "2021". The corresponding data values can be obtained using the dataValueSet endpoint. It is important to note that the process of comprehension necessitates four endpoint calls (the underlined data entities above).

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 5.1: Metadata identifier scheme for fetching data from an example template dataValueSet.

Examples can be important for the user's understanding of different aspects. A documentation with only textual descriptions alone can be overwhelming, but difficult concepts can become easier to understand for the user when combined with relevant and descriptive examples. Though, the examples need to be anchored in relevant use cases to ensure that the examples are usable and provide relevant information; an abundance of less relevant examples can make them work against their intended purpose. It was also mentioned by participants that examples describing relevant use cases with the possibility to adjust query parameters can improve learning efficiency significantly. This way, the user is not only able to see what an example-solution is, but also play around with the example to discover what happens when different parameters are changed.

The examples should not replace the technical documentation (the textual descriptions) but rather complement it to make the technical details more understandable. To avoid the feeling of information overflow, it can likewise be beneficial to combine textual descriptions with different visual and interactive elements like graphs, sandbox examples and figures. Some participants claimed textual descriptions should be short and precise (i.e. high simplicity and clarity), and a consideration of relevance is important to avoid unnecessary content. For supplementary external documentation that may be beneficial to include for learning, it can be wise to link these rather than describing the these tools and resources again in the documentation.

Visualizations and illustrations

Both the academic participants and the developers expressed that illustrations and figures as examples may aid in trying to understand the different metadata entities and their relationships with each other. These types of examples may make it easier to understand what kind of strategy is necessary in order to receive the data they want, and that way find out what kind of API callback functions they need to use. They also expressed that figures and models may be beneficial for structural and architectural understanding of the API. The participants also expressed that visualizing through use case examples can also explain the API in real contexts, rather than describing the use in a generic way. By doing this, writers can ensure that users understand how to actually use the API in more complex scenarios, potentially reducing the amount of misunderstandings and wrong use. They also implied that use cases can be an efficient way to describe how to conquer common difficulties and obstacles, hence helping the user utilize the API optimally.

Interactive examples

There was a broad agreement that interactive examples simplify and enhance the understanding of functionality of the API. Seeing the technical details put in practice makes API use easier to envision and understand. The documentation may seem more focused on actual use than only displaying technical information as a consequence. An example of such a tool can be the native Data Query Playground provided by DHIS2 (see figure 5.2), or a similar service like Postman, that simultaneously display the definition of a query that you want to execute on the left, and the response the individual received for that API request on the right.

The importance of making the examples interactive was also mentioned multiple times, as "getting the solution" does not necessarily lead to actual understanding and usability of the API. Connecting the examples to a sandbox environment, allowing the user to play around with the different parameters, was mentioned as something that could greatly improve the learning outcome of the documentation. The examples should provide information about what the responses should look like and what response codes the user can expect to receive. The aim of these examples are to provide the user a low threshold opportunity to try the API in use without the need for creating a program for testing it. The examples do not 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.

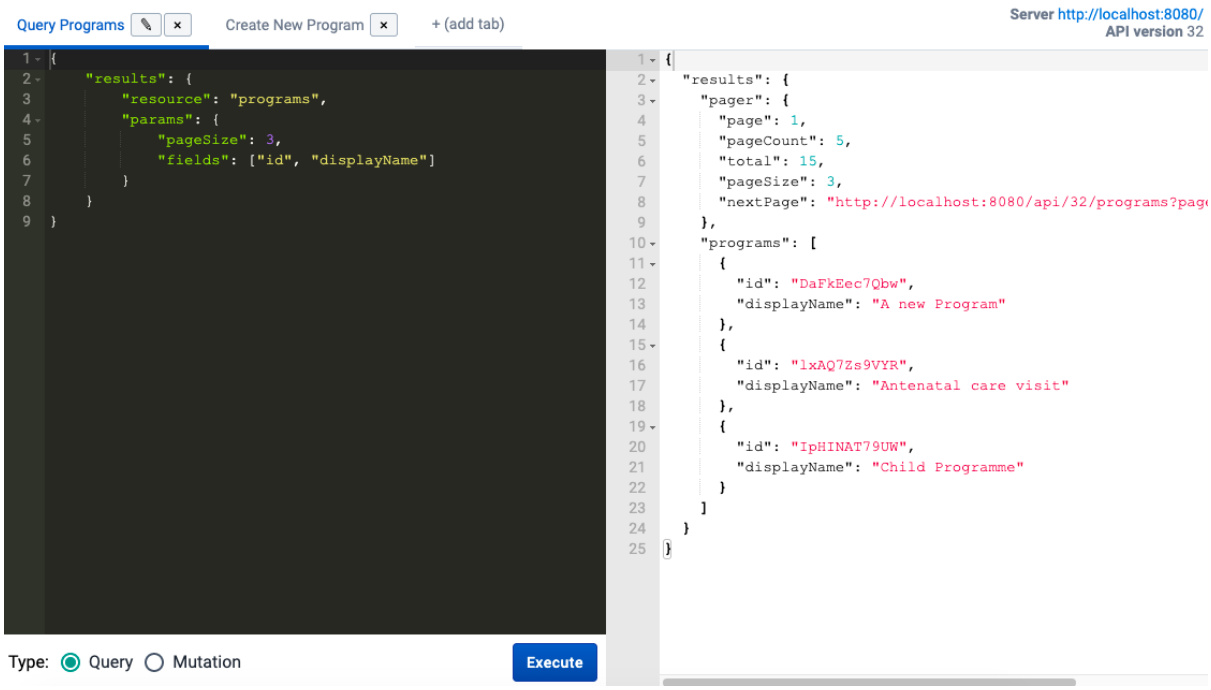

Figure 5.2: The Query Playground that enables testing of various REST API requests to define or experiment with query writing.

Concepts and terminologies

The developers expressed that the terms used in the documentation sometimes pose a challenge. They pointed out that the term 'data' may be too vague if the reader is not provided with the context the word is presented in. Other participants have expressed that context is key for understanding, regardless of whether the obstacle is a term or conception in the documentation. The academic participants mentioned that students in most cases are incapable of grasping all necessary concepts concurrently, especially very technical terms. Therefore, they propose beginning with the easiest conventions and gradually increasing the level of difficulty. However, they noted that the students have varying degrees of subject matter comprehension which poses a challenge for instructors who create learning material. This presents students with a remarkable obstacle in development courses that contain short-term projects. They also express that the writers of documentation should make sure to define important terms in order to avoid confusion. However, the definitions need to be included carefully, as too much information may increase the risk of information-overflow. A possible solution proposed by a participant could be to link to a separate documentation 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.

Document navigation

The developers indicated that a table-like structure for the documentation would make searching through the documentation more efficient. Text-searching capabilities therefore seem like a necessary aspect of documentations. One of the developers mentioned that they frequently use the built-in 'find' functionality of the browser (accessed by pressing 'Control+F'), type their keyword and inspect for matches. However, they stated that this may be counter-intuitive when the document is searched for all occurrences of the word, as frequently used terms will appear nearly everywhere throughout the document (e.g. the word 'data'). There is also the challenge of not knowing exactly how to search for the desired information. According to this participant, using search engines like Google provide easier opportunities for navigation (for example, by typing "DHIS2 web API data store") as the engine is capable of locating approximate, relevant information.

5.2.3 Data structure comprehension

Understanding the data structure

Some participants mentioned that the API documentation which may be easier to comprehend if the content was presented differently. Incorporating visual aids, such as diagrams and illustrations, may facilitate comprehension of the metadata, as suggested by the academic participants. In turn, this may enhance the ability to utilize the API effectively in various projects. The DHIS2 developers emphasized the importance of understanding the metadata model (depicted in figure 5.3). The DHIS2 Team has implemented a manual onboarding procedure for new developers to facilitate their learning of the metadata model. Although the DHIS academy provides resources on the metadata model, developers make limited use of them.

As previously mentioned, the majority of developers with experience of working with DHIS2 agreed that the learning curve is steep initially. The complexity of the data structure makes it difficult to comprehend the actions required for development. The DHIS2 developers implied that it is challenging to create onboarding procedures that are not dense in information because most of the metadata is interconnected (see depiction in figure 5.4). The interconnectedness makes it challenging for the newcomers to understand the data structures and what they have to do in order to write desired functionalities. One suggestion by a participant was to describe the data elements and their relationships in supplementary documentation in order to acquire a better grasp of how the various callback methods affect the data in the databases without overbearing the learners.

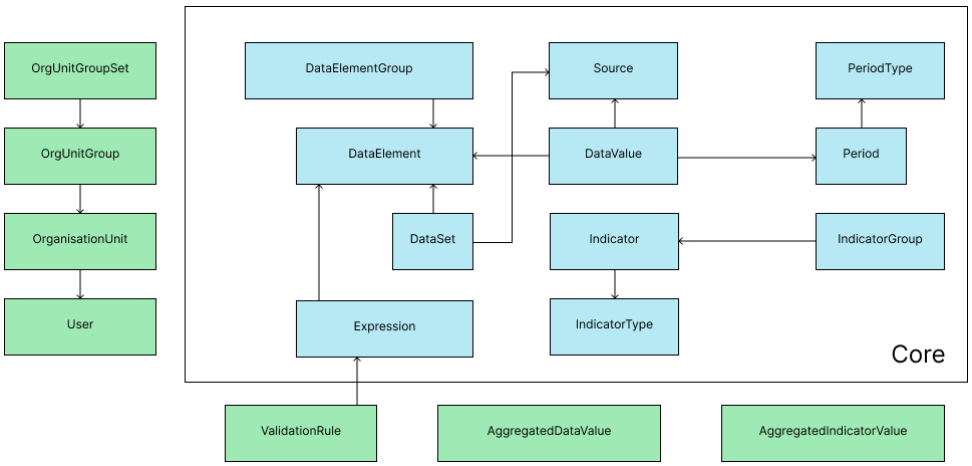

Figure 5.3: The metadata model of DHIS2 (DHIS2 Team, 2023c).

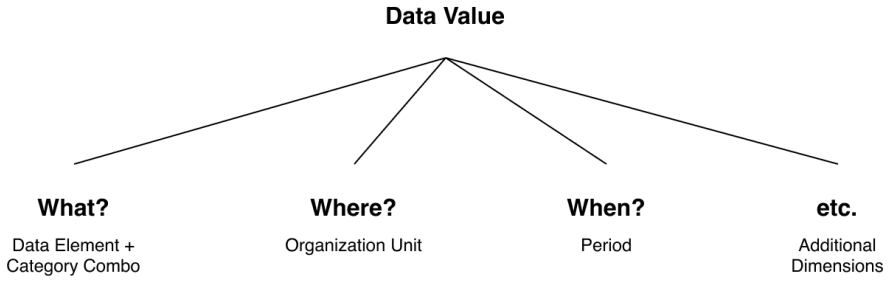

Figure 5.4: Data value structure, an example of different interconnected elements needed to receive the value of a data element. (DHIS2 Team, 2023b).

Version updates and the impact on comprehension

Several academic participants reported that novices encounter increasing challenges adapting older learning material to potentially new and significant changes resulting from API version updates. In student project work, this is exemplified by difficulties in constructing callback functions to obtain desired data as a result of the phase-out or discontinuation of the formerly taught method. Without clear indications of significant changes, it becomes needlessly difficult to update these courses. Developers also face issues when certain functionalities become outdated or obsolete without their awareness. One developer noted that identifying these errors can be tough because they must first troubleshoot in the coding environment before attempting to determine what may be wrong. Frequently, they discover that they must independently search for new changes in the documentation, despite the fact that this is not their initial course of action.

Other means of action

Several participants suggested other approaches for enhancing the instructional value of the API documentation. One suggestion was to emphasize programming experience with the API, which provides a more thorough understanding of how the API's logical components function in application development. They also implied that individual use cases presented in the API documentation can be extremely useful for demonstrating how to implement action flows in real-world situations. This strategy can provide novice application developers with an effective learning experience that is closely aligned with industry standards.

5.2.4 Pedagogical perspectives on API learning material

Handling version updates for educational material

For both app development course instructors and developers, keeping up with new version changes presents a significant challenge. When an API undergoes an update, the corresponding documentation must be updated. Due to the rapid pace of innovation, APIs can change at a fast rate, making it challenging to create up-to-date educational materials. In the context of Android development, for instance, one academic participant noted that the majority of course material can become obsolete before the end of the term. Creating guides that utilize third-party APIs, in particular, requires frequent updates; failure to do so may result in developers attempting to implement a callback function that is no longer supported or has become obsolete. This presents a challenge for teaching API usage best practices, as certain callbacks may be removed or altered to the point where existing applications break.

Observed challenges for novice developers in development courses

Academic participants identified obstacles that novice developers may face in educational settings. In particular, short-term application development projects may be difficult may be difficult to understand within the limited timeframe of the project for students due to the complexity of the architectural style. This can lead to a lack of understanding of the data architecture and a rush to find quick solutions, which can impact the maintainability and future development of the software. They also observed that students tend to extract more data than necessary from servers, indicating a lack of understanding of the callback function and an inability to determine the necessary callbacks for their use cases. Due to the extensive documentation of the DHIS2 web API, it can be difficult to identify the correct callbacks for manipulating data, and the API's capabilities may not be readily apparent to new users. This can lead to difficulty in performing actions not supported by the API and resorting to non-standard methods to manipulate data.

Onboarding and coaching experiences for DHIS2 developers

One participant suggested that, in order to enhance the onboarding experience for new developers, the structure of the documentation should be modified to make it simpler to locate relevant API callbacks. For new developers, the current structure has proven to be burdensome. The participant recommended introducing relevant API callback examples during manual coaching of the metadata model, which may also assist in the learning process. However, another participant expressed concern that providing examples might not necessarily address the broad range of scenarios that developers may encounter, given that there are numerous instances of the same problem. It may be difficult to provide a minimum number of examples that cover the majority of possible scenarios.

5.2.5 Content presentation

Some participants suggested that examining established corporate documentations, such as Google's Android development documentation, could provide ideas for designing the user interface and journey of digital documentations to enhance the existing DHIS2 web API documentation. One participant suggested that those in charge of creating the API documentation investigate the strategies used by notable companies to address issues with the DHIS2 web API documentation. They imply that these large-scale organizations may adhere to widely accepted documentation standards, which other developers may subconsciously anticipate in the DHIS2 documentation.

Detailed API operation description

Providing step-by-step descriptions of the different possibilities may be beneficial when targeting a less-experienced user group. There is no general consensus on how to present information, but simple and clear explanations of the step. Simple explanations of the steps may improve the efficiency of API usage, particularly for actions related to common activities such as authentication, access, and data receiving. The descriptions must be formulated and written correctly so that the user can comprehend the procedure. It is also important that the descriptions are comprehensive and include every action required to achieve the described objective. To avoid overwhelming users, participants suggest linking external resources, such as descriptions of how to use underlying technologies such as REST, to their respective, original resources.

The comprehensive operation descriptions should include a brief explanation of the various queries and parameters they accept. Participants noted that tables can be employed for a straightforward overview and comprehension of the various endpoints. Included in the descriptions should be what the query expects to receive (required), what it accepts (possible to send), and the endpoint's default value. It should also include a brief expla-

nation of the function and purpose of the query. The format of the descriptions depends on the context of the presented information. The following figure is an example concept based on the ideas presented by the participants (see figure 5.5).

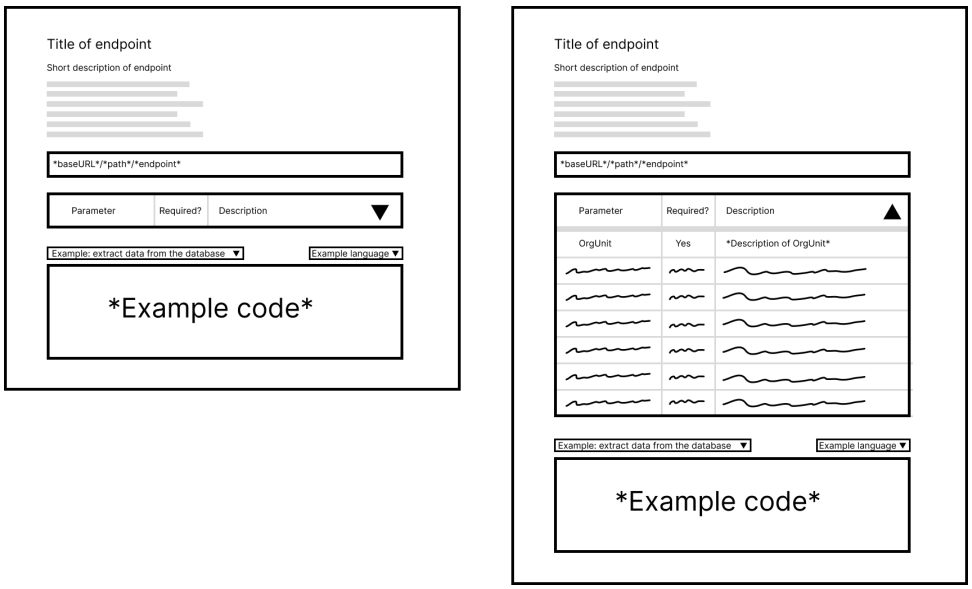

Figure 5.5: Conceptualization of a possible content layout with collapsible segments to avoid overwhelming the readers without removing information.

Content density and order

Content density and the impact on navigation

Participants in the study suggested that there should be a balance between a one-page, plain text presentation and a presentation with divided content that separates the various categories of information. They agreed that content division should not replace plain text presentation, but rather supplement it. Experienced users may prefer the ability to search through the documentation using keyword search, rather than navigating a divided structure, and fully replacing the complete overview can affect their efficiency. Without sufficient knowledge of APIs, their structure, and naming conventions, it may be difficult for novice users to search through a comprehensive list of all available endpoints. Google is also commonly used to find content on documentation; therefore, a content-presentation that is divided could be more advantageous for search engine results. Participants suggested incorporating a "similar suggestions" or "did you mean..." feature for searches that do not return exact results but rather approximate results that the user may have been seeking. This method allows users to explore and comprehend the documentation without having a solid grasp of the underlying structure.

Content order prioritization based on importance

According to the participants, it is important to prioritize and organize the documentation based on the importance and relevance of different sections, as developers frequently want to get started quickly without reading unnecessary material. Users must be able to navigate quickly to relevant sections. To facilitate a quick and straightforward introduction to the API, participants recommend beginning the documentation with an explanation of the system's most important and fundamental aspects, such as authentication, gaining access, and getting started. This will aid new users in gaining a fundamental understanding of the API so they can refer to the documentation when they encounter specific issues. The documentation should also include a concise, informative, and precise introduction that describes the API's overall functionality, the data and functionality it can provide, and how it obtains various results. Some participants claim that to facilitate the use of the API, it is essential to describe the key actions early in the documentation, along with brief descriptions of the various endpoints, parameters, requirements, and restrictions. Finally, the documentation should include less essential details such as rate limit and pagination. The precise content of this section depends on the API and its intended use. In order to provide an efficient and effective resource for developers, it is crucial to organize and prioritize the documentation based on its importance and relevance.

5.2.6 Coding results: API learning experiences

The following table 5.2.6 summarizes the coding results for this iteration. Key notions that emerged from the data are *content engineering*, *levels of expertise*, *content presentation and structure*, *navigation* and *user generated resources*.

Category	**Factors**
Navigation	A table-like structure for the documentation would make searching through the documentation more efficient
	Without sufficient knowledge, searching through a large overview of all available endpoints may be very time consuming
User generated resources	Manual coaching (i.e. by mentors) of the metadata model
	Supplementary documentation that explains the metadata model

Category	Factors
Content engineering	Introduce what the API does and what overall functionality it provides
	Tension between simplicity and depth of content
	Tension between efficiency and comprehensibility
	Address the intended target group of the documentation
	Introduce relevant API callback examples during the manual coaching
	Detailed API Operation Overview
	Ensure understanding of relevant terms while still avoiding the risk of cognitive overflow
	Challenging to create resources that are informative enough but not too overwhelming in content density
	Harder to teach best practice of the use of said API when some callbacks may be removed and others may be changed
	The API's capabilities are not as apparent
Levels of expertise	What are the intentions of the documentation?
	Degree of difficulty
	Examples need context
	Examples: visualizations, illustrations, code examples
	Gamification to boost engagement with learning material
	Students may be prone to not giving enough time to understand the data architecture, and are in a rush
Content presentation / structure	Content order prioritization based in importance
	Division of content should not fully replace the plain text-presentation of the documentation, but rather supplement it
	Content density should not compromise search functionalities
	Organize according to the relevance and importance of the different sections
	Look at existing documentations by large corporations for ideas on how to design

Table 5.1: Emerging phenomena of API learning experiences.

5.2.7 Going forward

In conclusion, the purpose of this iteration was to investigate the difficulties novice developers face when using web API documentation under time constraints. The findings, however, suggest that the API documentation cannot be viewed in isolation and must be viewed alongside other platform application development resources. Developers utilize a variety of resources to solve development issues, and these resources can affect the API learning experience. The next iteration of this study will concentrate on enhancing the existing API documentation to ensure that it provides understandable information that is not presented in a manner that causes cognitive overload. Additionally, the organization of content and navigation journey, as well as the impact of other API learning resources on the learning experience, will be emphasized. The subsequent iteration will investigate content engineering, problem-solving strategies, data structure comprehension, and other learning resources to improve the API learning experience.

5.3 Iteration 2: Qualitative interviews with students and DHIS2 developers

This iteration reports topics from the first iteration of the data collection, including problem-solving strategies and pedagogical contexts.

5.3.1 Problem-solving strategies

Participants in the study indicated that a developer's problem-solving skills are closely related to their API navigation skills. They viewed this ability as crucial for application development success. According to one participant, acquiring a "level of expertise with problem-solving [the] technology" can significantly aid in understanding and managing the various complex scenarios that arise during a project, such as comprehending the metadata structure.

Trial-and-Error

According to the results of the study, seven out of eight participants use Trial-and-Error as their primary method for learning the API. Oftentimes, this is their only option, particularly when the documentation is inadequate or the API functions do not behave as expected. In addition, Trial-and-Error can help developers learn how the system handles various components. Some participants argue that excessive documentation can discourage developers from reading it thoroughly, leading them to rely instead on trial-and-error. However, they also recognize the importance of guidance in locating the necessary information to overcome implementation obstacles. Mentors, forums, and resource libraries may be included. In the absence of such guidance, they frequently rely on brute forcing and Trial-and-Error.

Furthermore, Trial-and-Error may be advantageous for developers who prefer practical exercises to theoretical reading; yet, participants emphasized the need for a balance between the two approaches in learning environments. The context in which developers employ Trial-and-Error may vary. Others use it to inspect data requests and responses, while others use it programmatically to achieve desired actions. Understanding metadata and their relationships is regarded as the most difficult aspect of learning the DHIS2 API, and Trial-and-Error is a common technique for gaining an understanding of the platform's metadata structure.

Test-driven approach

Two of the participants described a test-driven method for acquiring API knowledge. A test-driven approach, unlike Trial-and-Error approaches, establishes a learning objective prior to experimentation, making it a proactive method. This entails composing requests that meet specific requirements with the intent of gaining knowledge and locating solutions to the existing issue. In contrast, Trial-and-Error is viewed as a technique of pure experimentation employed when a problem is not fully understood. Participants who employ the test-driven methodology believe it can be advantageous when they anticipate specific API responses from requests they are formulating.

The steps of the iterative procedure are the following:

- *Define Test Cases:* Create visual representations, such as tables or flowcharts, outlining desired API functionality, inputs, outputs, and edge cases.
- *Write and Execute Test Code:* Implement test cases using a testing framework before writing API code. Execute tests, visualize results with indicators (e.g., checkmarks, X marks).
- *Implement API Code:* Write API implementation code based on test cases.
- *Refactor and Repeat:* Continuously run tests, refactor API code until all tests pass.

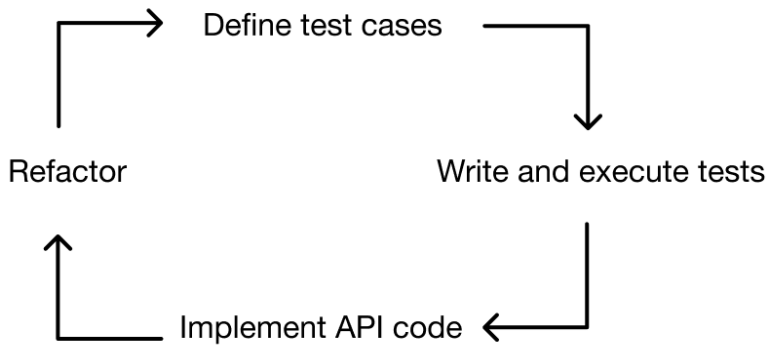

Figure 5.6: Illustration of the iterative test-driven procedure.

Establishing a Minimum Viable Payload (MVP)

Minimum Viable Payload (MVP) is the minimal set of data or parameters required to make a valid API request and receive a meaningful response in API application development. The objective is to maximize the API's efficiency by reducing the amount of data that must be transferred and processed while still providing sufficient information to the developer. This may be important in situations where speed and efficiency are crucial. One developer mentions establishing a minimum viable product as a requirement for writing API requests to help reduce the complexity of API responses and filter out irrelevant data. The provided visual example illustrates the exact extraction of an organization unit, with only the required data elements for their task (see figure 5.7), while the remaining data is not provided in the response.

Figure 5.7: An example of Minimum Viable Payload from the API.

Asking the community

Four participants reported using the platform development community or a mentor, including teaching assistants for students, to increase their knowledge of API application development and seek guidance when confronted with obstacles. There are a variety of ways in which they can receive guidance:

1. One individual receives the responsibility for another's understanding.

2. A development team shares their competencies internally, amongst each other.

3. A group of developers discuss while pair-programming, teaching each other through these discussions.

Furthermore, it has been reported that certain participants use online development forums to seek assistance and inspiration for their individual development projects. Nonetheless, as highlighted by two Malawians, the timeliness of receiving responses to questions posted on these forums may be inadequate, resulting in the resolution of issues prior to the receiving of meaningful input.

According to the Malawian developers, their application projects may be similar to those in other countries; thus, utilizing the broader development community for guidance can be advantageous because "[there is] always someone doing something similar to what you do (...),so if you encounter a problem, there is a good chance that someone else has encountered the same problem." The limited scope of the DHIS development community, coupled with the localized implementation of projects, makes it difficult to reach a sufficient number of individuals to provide support for questions. Consequently, relying on user-generated documentation, such as the community forum StackOverflow, was deemed preferable to requesting direct assistance from others, who may have their own pressing responsibilities and time constraints to contend with.

The localized application projects may impede the possibility of subcommunity-wide cross-learning. For instance, if Country A encounters a developmental obstacle, it may be unaware that Country B has already encountered a similar or even identical problem and successfully resolved it. Therefore, Country A may waste time and resources attempting to solve the issue on its own. To avoid such inefficiencies, a participant recommended the promotion of knowledge sharing between the different subcommunities to enhance learning and ultimately facilitate optimal problem-solving.

Mentors and teaching assistants have been identified as valuable resources for gaining

knowledge about APIs, particularly for beginners who may encounter difficulties. The assignment of a mentor to new team members is a common practice, according to a Malawian developer who teaches mobile software development at a university. The mentor assists newcomers by recommending resources and providing advice for overcoming challenges. Furthermore, it is deemed essential to encourage continuous learning, and so mentors provide resources that facilitate the acquisition of both theoretical and practical knowledge.

Alongside the difficulties associated with learning the API and its associated material, experienced instructors have observed that students frequently overlook important material and concepts in their haste to complete assignments and projects. Consequently, many students struggle to comprehend the underlying mechanisms in application development courses that employ APIs.

Mixed problem solving techniques

An experienced developer shared their method for learning the DHIS2 web API, which starts with identifying the necessary inputs for the program or application being developed. The data structure of the resulting query is then examined to eliminate irrelevant data elements that do not correspond to the desired result. The participant's approach to problem-solving can be described as a form of Trial-and-Error approach, although they begin by establishing a goal and defining the desired outcome in order to facilitate the learning process. This method is also applicable when analyzing queries from similar applications and studying how they acquire and manipulate data to achieve the desired results. Three participants shared similar strategies for structuring their approach to problem-solving when confronted with unfamiliar APIs. These insights provide a practical guide for problem-solving tasks involving unfamiliar APIs.

Learning guideline: An approach to understanding an unfamiliar API

1. Look at what's possible
   (a) Look at the documentation and Trial-and-Error interchangeably
   (b) Google resources
   (c) Asking community, use learning academy
   (d) Study existing apps (i.e. DHIS apps)
2. Test queries independently (i.e. in Postman)
3. Establish a minimum viable payload (MVP)
4. Implement in the application

Furthermore, the Malawian developer with teaching experience emphasized the significance of teaching problem-solving skills to entry-level developers in learning environments. In particular, they argued that demonstrating the process of problem-solving APIs, providing examples, and teaching students how to execute it could equip them with the tools and thought processes necessary to tackle more complex API application development tasks. They believed that students would learn more effectively if they observed various cases and independently attempted to implement them.

5.3.2 Understanding and learning the data structure

The majority of the developers claimed that understanding the API core is crucial to best understand the data model provided by the platform. The data model is the generalized structure of the metadata that all the collected data is shaped after, and is in most cases what the query parameters consist of . Three developers suggested entry-level developers to read through the metadata documentation that corresponds with the topic of interest (a data element, i.e. a given disease) to see graphical representation of the system design as it explains the different data sets for that given data element.

One of the challenges is locating this information in the official web documentation, as it is hidden in the navigation toolbar and most visible on the topic dashboard, which is rarely accessed (see figure 5.8). The DHIS2 Academy offers several resources, including self-paced courses for web and android app development. Malawian developers rely primarily on the DHIS2 Academy to learn about the system design, and only one participant mentioned using metadata documentation as part of their educational process. Norwegian students were unaware of the metadata documentation, while other developers relied on explanations from other team members and Trial-and-Error to gain a quick overview.

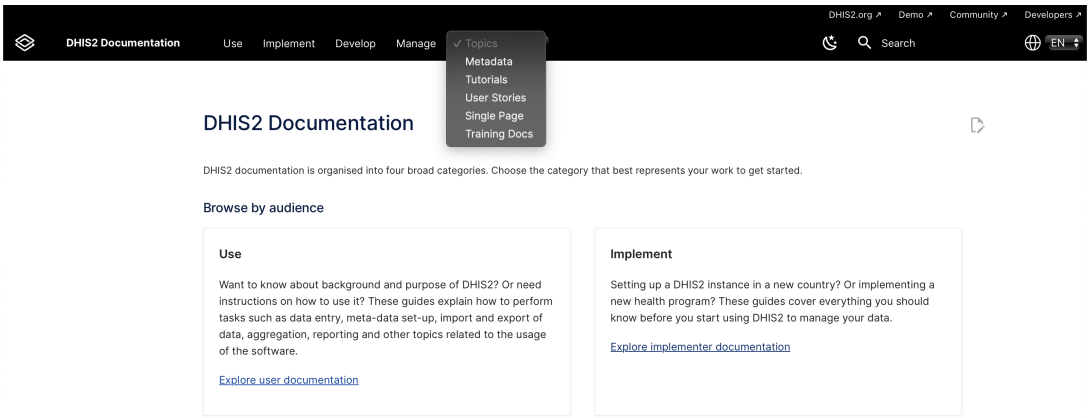

Figure 5.8: Snapshot of the metadata webpage hidden in the navigation bar.

All participants agreed that it is important to understand how data elements interact with each other before learning how to write callback requests through Trial-and-Error, as the elements are interconnected. Nonetheless, two participants suggested that practical learning, such as request inspection, can aid in the understanding of the relationships between the data elements. By limiting search queries to be highly specific (using request filters and parameters) and examining the data with a specific case or problem, they claim to have gained a more comprehensive understanding of the metadata than by reading theory alone. As one developer stated, "theory gives what is possible, but having a problem to understand technology would help." Another developer, however, noted that without structured courses to follow, it may be more difficult to learn, as individual motivation and the ability to search for appropriate resources are crucial.

When participants were asked about what the balance between theory and practice should be for effectively learning API application development, most participants conveyed it should be either 50 % theory and 50% practice, or a slightly higher percentage of practice (i.e., 70%) in order to get familiar with problem solving by themselves. However, the participant who argued for the 30/70 split vouched for education that could make students more “industry-ready” and be at a skill level to be able to develop applications after the course ends.

Comprehending individual data elements

Two of the Malawian developers also mentioned that utilizing the three dimensions of DHIS2 data when examining the data structure can aid in comprehending how to compose requests and examine data responses. The three dimensions are ’what’, ’where’, and ’when’, and they are used to describe the phenomenon of interest, the organizational unit associated with the phenomenon, and the time period during which it was recorded (DHIS2 Team, 2023a). They claim that these data dimensions may make the metadata structure simpler to understand, given that they are the fundamental building blocks of the data gathered in DHIS2. Additionally, one participant also expressed that understanding the API was less challenging after examining the query responses themselves.

When attempting to understand how to create applications that aggregate manually collected data, the participant also mentioned that “one has to understand what kind of data is being collected and what needs to be collected”. Specifically, they suggested that developers should have a clear understanding of the required data before coding the functionality into the application. This approach can lead to greater efficiency and reduced labor for developers, as it enables them to gain a better understanding of the underlying data structure of DHIS2 through repeated practice.

The situations of use of API documentation

When the working environment is unfamiliar or when Trial-and-Error fails to produce the expected results, all of the Malawian developers indicated consulting the API documentation. It's crucial, according to one participant, "(...) to see the opportunities when you start to use something new." However, after a few unsuccessful attempts at reading the documentation, they decide to use brute force to achieve their goals.

Notably, three participants criticized the DHIS2 API documentation, stating that it fails to convey the full range of API capabilities in an intuitive manner. One participant remarked that the documentation "assumes people know things, but [that is a] big problem if you try to learn [the API] from scratch." Another participant similarly said "[the] documentation doesn't always say what you can do in a lot of cases." Therefore, a weak point of the DHIS2 API documentation is this unclarity of the APIs capabilities. Consequently, the lack of clarity regarding API capabilities is a deficiency in the DHIS2 API documentation, which may lead some users to rely on Trial-and-Error approaches.

Other knowledge resources

Along with to the DHIS2 API documentation, participants mentioned a number of additional resources that they employ to guide the development process. These include StackOverflow, Google, and YouTube, which provide user-generated content to supplement the official documentation for the platform.

According to the participants, YouTube is utilized to gain a deeper understanding of API implementation processes and fundamental concepts, such as metadata and core platform features. StackOverflow is commonly used to access code examples that are pertinent to specific project requirements. Particularly, none of the official YouTube resources created by the DHIS2 team provide implementation and development guides. Google is frequently used to search for StackOverflow and YouTube resources, and half of the participants use it to effectively navigate the API documentation. For example, a participant may conduct a Google search for "DHIS2 API authentication" to directly access the authentication section of the documentation and reduce the number of required navigational steps on the platform's official documentation website.

5.3.3 Improvement suggestions

Using the API documentation as a starting point, the sample group described how problem-solving processes could be improved for API application development projects. The themes touched upon by them are (1) the presentation of the content, (2) Visualization of information and examples, and (3) onboarding and instructional courses.

(1) Content Presentation

One developer suggested that presenting the sequence of actions required to complete a task and the interactions between API components could aid in the understanding of information-rich APIs. One developer suggested that presenting the sequence of actions required to complete a task and the interactions between API components could aid in the comprehension of APIs that provide extensive amounts of information. They suggested that dividing data into smaller portions and interacting with them to achieve specific objectives is more manageable than receiving large data sets that must be filtered afterwards. To facilitate understanding of the API operation, it is necessary to provide a detailed API operation description that includes information on the behavior of callback functions and the steps that must be taken following their execution. Such comprehensive explanations can be especially useful for entry-level developers, as it enables them to comprehend the purpose of callbacks and acquire the fundamental knowledge necessary to modify their functions to achieve the desired results. This strategy would necessitate the incorporation of more exhaustive examples of callback functions and the clarification of their operation.

All participants emphasized how exhaustive the API documentation is and how difficult it is to locate the information they require. They suggested avoiding extremely lengthy pages on each topic of the API documentation, as it becomes difficult to find the information and guidance they need without using the search functions that are implemented on the web API documentation site. Two participants also mentioned that the API documentation's native search tool may not always yield the desired results. It can be difficult to construct a precise search query that uses the same language as the API documentation.

The Diversity of Need for Guidance

Some developers stated that excessive hand-holding in the documentation could hinder more seasoned developers who are already familiar with the system. These participants suggest that the documentation for the DHIS2 API should strike a better balance by providing essential operational information to users with higher levels of expertise without overwhelming them with excessive detail. Likewise, the documentation should ensure that users with lower levels of expertise have enough information to understand the API's functionality. This balance is essential because the technical abilities of the participants vary, leading to divergent opinions regarding the amount of information provided.

Entry-level developers in the sample group deemed the DHIS2 web API documentation insufficient when they required in-depth explanations of particular occurrences. However, they felt that the documentation contained an excessive amount of information that was irrelevant to their objectives. The dissatisfaction with the amount of information provided by API documentation is primarily influenced by the technical skills and goals of the audience.Therefore, the sufficiency of the DHIS2 web API documentation in terms of the information provided is subjective and dependent on the reader's usage context and goals.

Community-contributive documentation

Consistent with previous findings, two participants highlighted the potential benefits of incorporating functionality that encourages community contributions into the web API documentation. They suggested that allowing users to propose their own solutions to various scenarios and provide feedback would enrich the documentation and improve the readers' learning experience. This approach would allow the documentation to include a broader range of examples, tips, and tricks that have become more pertinent over time, fostering a collaborative and dynamic environment for knowledge sharing. By leveraging the expertise of the community as a whole, the API documentation can be continuously enhanced and expanded to meet users' changing requirements.

Textual Narrative Document Design

The Malawian developers prefer the DHIS2 Developer Manual to the current version of the web API documentation, primarily due to its organization. The developer manual's table of contents at the beginning of the document and its book-like structure were cited as advantages over the current version's fragmented two navigation panels with different headings. Participants felt that the developer manual's organization made it easier to navigate and comprehend. More precisely, one of said participants expressed that the API documentation writers should "structure it as if it was a book. It should

be straightforward". In order to improve its usability, the documentation for the DHIS2 API should have a more straightforward and less fragmented structure. The impacts of fragmentation on navigation is implied by the double navigation bars on the web API documentation (see figure 5.9).

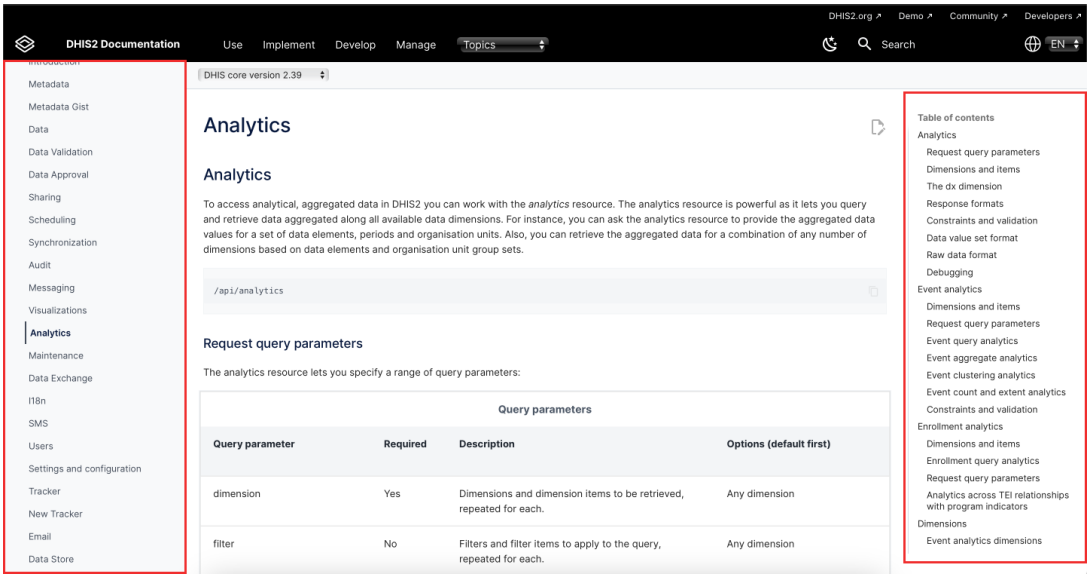

Figure 5.9: Double sidebars in the DHIS2 web API documentation; one for navigation on the web page while the other is the table of contents for the selected web page.

(2) Visual and Illustrative Examples

Seven out of eight participants indicated that more visualization of the metadata information can increase the likelihood of acquiring API concepts more quickly and effectively; in particular, the use of relational diagrams to improve the comprehension of metadata information in the API documentation. They believe that visual representations can aid in understanding how different data elements interact and are interconnected within the metadata structure of an information system. These diagrams would provide an abstracted view that would assist developers in aligning their projects with the underlying structure.

Visualized approaches in the API documentation, according to one participant, not only teach how to use the technology but also aid in teaching conceptual understanding. By utilizing visualizations, readers are better able to understand how to manipulate their queries to obtain smaller and more accurate data responses, as opposed to having to sift through large, potentially irrelevant responses. Thus, incorporating more visual explanations into the API documentation would contribute to faster and more efficient learning, allowing developers to better comprehend the underlying concepts and utilize the API for their specific requirements.

Code examples

All participants agreed that the API documentation would benefit greatly from the addition of additional code examples. They emphasized the need for a balance between textual information and practical examples to alleviate the documentation's burdensome nature. By providing more code examples, the documentation could provide shorter and more precise descriptions, making it simpler for developers to understand and implement API capabilities. Participants also suggested that the code examples should include alternative use cases, demonstrating various scenarios and methods for utilizing the API. It was noted, however, that these additional examples should not overwhelm the content or obscure a straightforward overview of the various callbacks.

(3) Onboarding and Instructional Courses

Participants emphasized the significance of a mastery-based learning strategy when instructing and assisting novices. They proposed that a pedagogical structure that combines theoretical and practical learning can aid entry-level developers in grasping the concepts over time. A participant mentioned, for instance, the effectiveness of introducing theoretical knowledge first to establish a foundational understanding, followed by practical exercises or lab programming sessions that gradually incorporate the theoretical knowledge. This approach permits continuous learning by alternating between theory and practice, allowing developers to apply their knowledge in real-world scenarios and reflect on phenomena that arise during practical work.

Regarding the DHIS2 onboarding, three participants expressed concern about the absence of formal onboarding procedures. One of these participants claimed that they "get thrown into it and have to learn the system on the go". Two participants emphasized the need to rely on more experienced developers within their teams in order to acquire the essential practical skills and conventions for developing applications on the platform. This indicates a lack of structured onboarding and highlights the development teams' reliance on informal knowledge sharing.

Collaborative practical learning

One participant also proposed the idea of introducing more collaborative group work for the lab programming sessions. The participants recommended incorporating group projects, code reviews, and mentorship programs into the onboarding and learning processes. These collaborative methods may foster a sense of community and knowledge-sharing within the DHIS2 developer community, allowing entry-level developers to more efficiently learn from their more experienced peers and develop their skills and understanding of the DHIS2 API system. Encouraging entry-level developers to use *Code*

Reviews as an approach to learning more about platform application development can potentially aid in simplifying the process of comprehending the system. Furthermore, during discussions, certain participants advocated for increased efforts towards encouraging the DHIS2 community to publish more of their application projects to the DHIS2 marketplace specifically for this purpose.

5.3.4 Coding results: API learning and problem-solving

The following table summarizes the pain points identified in the dataset that impair entry-level developers' API learning experience (see figure 5.3.4). These pain points have been thematically categorized to identify the specific problem areas that impact learnability. It is important to note that a significant portion of the identified pain points are related to the DHIS2 API documentation.

Category	**Factors**
Content engineering	Lack of formal definition of API
	Lack of sufficient (code) examples with information
	Unclear descriptions of data elements and their relations between each other
	"Enough [information] to understand what you can use, but too little to understand how it is used"
	"Too simple use cases, should have some more complex ones"
	Insufficient information about topics and issues
	Information and practices are not standardized. "Everyone is doing things differently"
	Content written in a way that implies that it expects the reader already knows the metadata structure or how the API works
	Not all information is documented or up to date
	"Can be difficult to interpret examples written by very experienced developers. They often take shortcuts, and don't necessarily write with best practice"
	Web API documentation does not clarify the APIs capabilities

Category	Factors
Content presentation / structure	Badly categorized content. Challenging to know where you can expect to find information you want
	"Too much text, it's overwhelming"
Navigation	"Hard to know what you have to search for [in order to find the information you need to solve your obstacles]"
	No universal search function that searches throughout the website. Need to search on each web page individually
User generated resources	Hard to find the right resource to gain the answers you need
	Developer community is not very engaged and it takes time to receive answers

Table 5.2: Pain points that impact API learning.

The table demonstrates that API documentation is a major source of frustration for entry-level developers, as it often lacks practical code examples, contains too much text to read, and fails to provide adequate information and updates. Additionally, the documentation assumes prior knowledge of the metadata structure, which can be difficult for new users. Due to their use of syntactical or logical shortcuts without adequate explanation, some participants find it challenging to comprehend examples written by experienced developers.

Intriguingly, the participants' perspectives on the presented data are contradictory. On the one hand, they believe there is sufficient information to comprehend the API's available tools, but insufficient information to comprehend how to utilize these tools effectively. On the other hand, they believe that the API documentation does not adequately clarify the API's capabilities, leaving users unsure of the available tools. Other identified pain points include difficulties in locating necessary information, either because it is unavailable or because it is not organized in a way that makes it easy to locate. When confronted with obstacles, participants also struggle to identify the appropriate resource from which to seek guidance. Finally, difficulties in engaging other individuals for information sharing and delays in receiving responses are mentioned.

5.4 Summary of results

The overall topics covered in the first iteration relates to the structure of the documentation, indicating that the informative structure poses the greatest barrier to obtaining the needed information. Other topics include explanations and examples, the degree of difficulty for those unfamiliar with the documentation, content, navigation, and user generated resources. In the second iteration where all the individuals were exposed to DHIS2 application development, but with differing levels of expertise, the themes of content engineering and presentation, data structure and functionality comprehension and user engagement were the most emphasized.

The emergent categories identified in the data sets following the final coding process is visually represented in the following tables:

Core category: Entry-level API learning experiences

Category	**Code**	**Emerging themes**
Data structure and API functionality comprehension	Understanding the data structure	• Asking the community • The use of API documentation • User generated resources • API version updates' impact on existing knowledge
	Learning paths and pedagogical contexts	• Student courses, onboarding and coaching • Collaborative practical learning • Handling version updates

Table 5.3: Part I: Impacts on entry-level API learning experience.

Category	Code	Emerging themes
Content engineering and presentation	Content simplicity and depth	• Efficiency affects simplicity, while depth affects comprehensibility • Impact on navigation • Detailed API operation description for efficient use and implementation processes, troubleshooting and callback comprehension
	Visualizations and examples	• Visualizations, illustrations • Code examples • Interactive examples • Impact on content presentation and navigation
	Presentation: documentation structure	• Content density and the impact on navigation • Content order prioritization based on importance • Textual narrative document design
User engagement	Community-contributive documentation	• Encourage more user involvement • Encourage sharing applications for study purposes
	Engagement and incremental learning	• Gamification

Table 5.4: Part II: Impacts on entry-level API learning experience.

Category	Code	Emerging themes
Differing levels of expertise	Intended documentation usage	• Addressing whether documentation should present API capabilities or explain how the API functions
	Subject difficulty and expertise level mismatch	• Attuning the content difficulty to the different levels of expertise
	Concepts and terminologies	• Risk of information overflow • Hard to grasp concepts concurrently with implementation
	Problem-solving strategies	• Information searching abilities • Test-driven approach • Trial-and-Error • Establishing a 'Minimum Viable Payload'
	Differing need for guidance	• Differences based on prior knowledge

Table 5.5: Part III: Impacts on entry-level API learning experience.

Additionally, some participants gave tips to entry-level developers in how they approach unfamiliar APIs, which has been constructed as an API learning guideline (see figure 5.3.1).

6 | Discussion

The the data collection revealed that the four most important themes that affect the entry-level API learning experience for these data subjects are content engineering and structure, user engagement, differing levels of expertise and data structure comprehension. Additionally, the study also revealed an API learning guideline constructed by developers with higher levels of expertise in comparison to entry-level developers.

6.1 Content engineering and presentation

Content engineering and presentation highlight the impact of content curation and presentation style on the readers' comprehension abilities and how this impacts the learning experience. As previously mentioned, the phenomena that have the greatest impact on a reader's comprehension abilities are the content's level of depth, how easy it is to comprehend the information, how well the documentation is updated and the changes are communicated to the users, how much input the documentation's users have in the content provided, how information is formulated and knowledge is presented, and how the content is presented visually.

6.1.1 Content simplicity and depth

The findings revealed that the participants would benefit from higher levels of content depth. A way of achieving so would be to present detailed, in-depth API operation descriptions to aid the learners in understanding the logics of the callback functions, troubleshoot their queries and to learn more about how to efficiently implement the functions programmatically in their applications. On the other hand, some data subjects also implied that a significantly high quantity of information is counter-intuitive, and can become overwhelming and disorienting for the users. This finding supports the adverse effects of significantly verbose descriptions in API documentations instead of simple and straightforward descriptions (Uddin & Robillard, 2015). High amounts of information that exceed cognitive processing constraints may result in sub-optimal decision-making and performance (Malhotra, 1982); it may also make it difficult to determine what is relevant and what is not (Moy et al., 2018).

As noted in section 2.3.3, Robillard and DeLine (2011) similarly received findings suggesting that 'boiler-plate' type of descriptions of methods do not contribute to valuable insight for their sample groups, and would instead be a waste of time to read through.

Purchase and Worrill (2002) argue that the assistance provided should be easy to understand and prevent frustrating the readers. By these claims, it is implied that the learnability of the learning resource is dictated by simple and retainable information. There is therefore a tension between the levels of simplicity and levels of depth needed to create an optimal knowledge resource for entry-level developers

The challenge is addressing how in-depth the information can be before it compromises the learning productivity of the readers. In order to facilitate efficient reading material, the content will have to be as simple as possible so that retaining the information becomes easier and boost the transfer from similar API learning contexts to the current one (Nelson & Eliasz, 2023). If the content is too simplified and lacks sufficient details pertaining to a given topic, it may be difficult for developers to comprehend and acquire knowledge regarding the rationale behind executing particular operations in a particular order.

Thus, it is necessary to investigate methods for achieving optimal levels of reading and learning efficiency in order to solve the problem of creating simple (i.e., easy to understand) content. To accomplish this objective, it may be necessary to investigate the efficiency metric that data subjects associate with the concept of readability in reading-oriented scenarios. Moreover, efficient content also entails optimizing the intrinsic load, as in adjusting difficulty of the subject (Fuhrman, 2017), in order to alleviate the cognitive load (van Gog et al., 2011). Based on the concept of instructional efficiency, 'easy to understand' learning resources are those that produce the greatest learning outcomes with the least amount of time and effort. To put it differently, these learning materials exhibit a high degree of simplicity. As indicated by the data subjects, their limited use of the DHIS2 web API documentation can be related to a number of factors, including their inability to obtain prompt responses to their questions. Several of the previously mentioned data subjects also emphasize the difficulty of comprehending the API documentation.

Accordingly, it is suggested that the documentation content should prioritize presenting the most important information required for operating the API functionality first, and present the content in a way that minimizes the users' reading time, thereby enhancing the API-related learning resources. To achieve this objective, the content must be designed in a way that addresses the fundamental knowledge requirements for API functionality operation, thereby reducing the intrinsic load, and presented in a less cognitively demanding format, thereby reducing the extraneous load. This can be accomplished, for instance, by incorporating more visual examples and organizing the information according to its operational relevance. The result would be a knowledge resource that prioritizes simplicity to ensure precise and clear knowledge conveyance through visual examples, followed by more detailed information.

As mentioned, some data subjects also expressed the need for detailed API operation descriptions in order to learn how to use the API efficiently. This sentiment suggests that lengthy textual descriptions can be shortened and broken down into more cognitively manageable pieces by providing detailed operation descriptions and code snippets so that readers can follow the operation process and comprehend the procedure. Blayney et al. (2015) also expressed in their research that novice learners benefit from the detailed, step-by-step instructions as it reduces the working memory load, especially if the operation description is presented before interactive learning activities. Purchase and Worrill (2002) similarly stated that online help should be designed to provide help in a procedural manner that addresses exactly what the user needs to do. Furthermore, the effectiveness of detailed API operation code snippets that illustrate usage patterns and programming logics has reported to be higher than documentation that are highly textual in the findings. Step-by-step procedural instructions may therefore aid in providing higher levels of content depth by making the information more cognitively manageable, given that the fundamental information presented is clear and precise.

The reader's attention span and the impact of the content on cognitive load may be a possible explanation for the increased learning effectiveness provided by detailed API operation descriptions. Attention is intimately related to the processes of memorization and learning, as it determines which portions of the working memory will be encoded and thus learned (Chen & Lin, 2016). The learning processes of individuals can be influenced by presenting content in a manner that minimizes the effort required to understand and process it. Several data subjects have suggested that the API's various endpoints can be better understood by providing clear and precise operation descriptions in a table-like format, as well as brief explanations of the accepted queries and parameters. Thus, dividing the content into smaller, more easily processed informational chunks, as opposed to lengthy textual descriptions, could improve the API learning process. However, it can also be argued that even though the detailed operation descriptions are useful for novice learners, it may increase the cognitive load for more experienced learners as they would have more material to read through Blayney et al. (2015). Therefore, a careful consideration of who the documentation is supposed to be addressing plays an important role in being able to give them the help they need.

In addition, the data subjects argued that while it is important for the operation descriptions to be comprehensive with other external technologies that the API depends on, it is also essential to link information about these external resources to their respective original counterparts in order to reduce the documentation's length and make it less overwhelming. In essence, providing concise descriptions of external resources and links to relevant information in the original documentation would reduce the amount of content

presented in the API documentation, thereby easing the cognitive load of readers. By reducing the amount of information requiring sustained attention, working memory loads would also decrease, which may improve reading comprehension (Chen & Lin, 2016).

In conclusion, prioritizing the instructional effectiveness of the content structure can greatly improve user comprehension and make API learning less intimidating. The content must provide users with sufficient instructional detail to acquire substantial knowledge, while also being concise and presented clearly. For it to improve the learnability of the API, the tension between the levels of simplicity and depth of the content must be resolved. To accomplish this, it will be necessary to consider the learning context of the documentation, how the resource will be presented to the audience, and who the intended audience is. It may be wise to develop distinct documentation resources for different target groups, or to find alternatives to filtering based on the characteristics of target groups.

6.1.2 Visualizations and examples

The results indicated that visualizations and illustrations could facilitate the communication of information regarding the intricate relationships and interactions between entities present in both data and programs. The data subjects were confident that API documentations require more visualizations and illustrations of fundamental data structure concepts in order to make the information more comprehensible while simultaneously increasing the instructional efficiency. Visualization of information for the purpose of knowledge transfer must improve the conceptual understanding for the learners (Kuljis & Baldwin, 2000) in a manner that others can comprehend (Fadiran et al., 2018). According to the data subjects, understanding the various concepts and entity relationships is one of the most challenging aspects of API learning. Therefore, visualization may aid in enhancing information transfer while requiring less cognitive effort from the learner and reducing information overload on readers. As these visualizations and illustrations can be abstractions of the complex topics of the API, they may assist in creating problem solving schemata that can increase their understanding of better implementation practices. Additionally, the visualizations are easier to remember (Nelson & Eliasz, 2023) and may therefore optimize the working load memory for retention (Gao et al., 2020).

Fadiran et al. (2018) aggregated factors that constitute a 'good' visualization that are similar with the metrics mentioned by the data subjects during both iterations of data collection. For knowledge visualization, five attributes were identified to have a major impact on how well a visualization is perceived (Fadiran et al., 2018) that can be applicable for conveying knowledge:

1. understanding the data domain to create meaningful illustrations,

2. clarity of the content with clear symbols

3. easy comprehensibility in a way that required as little previous knowledge as possible to comprehend the content,

4. clear boundaries in visual cues to separate different elements, and

5. the relationship between the concepts.

The following figures 6.1 and 6.2 visualizes the importance of structuring the information and showing procedural explanations for the user with the aggregated factors in use in order for the visualizations and illustrations to be effective. For API documentations, these visualization guidelines can be used to model examples in a clear and systematic fashion to demonstrate relationships and interdependencies between different API functionalities or metadata relations. The figure to the left illustrates a chaotic 'before' visualization, while the picture to the left illustrates a systematic 'after' state.

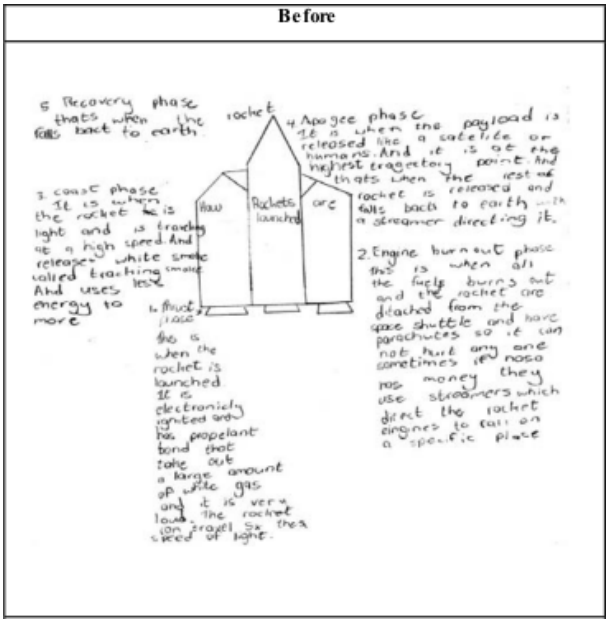

Figure 6.1: Samples of learners' visualization before visualization guidelines brief (Fadiran et al., 2018, modified).

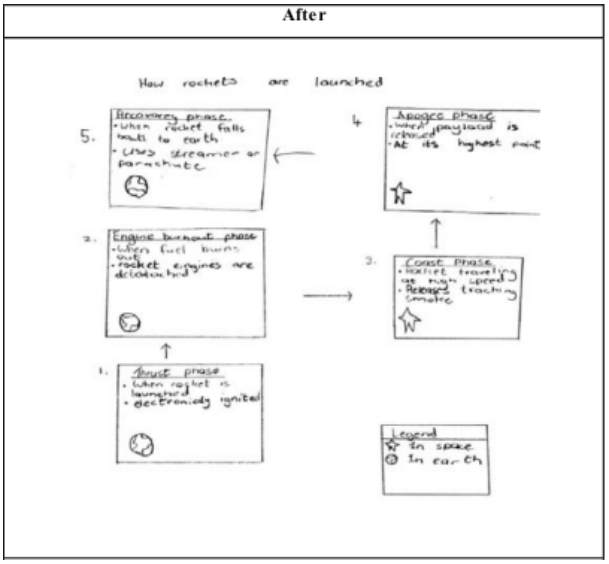

Figure 6.2: Samples of learners' visualization after knowledge visualization guidelines brief (Fadiran et al., 2018, modified).

These metrics are comparable to what the data subject described as the criteria with the greatest impact on the API documentation learning experience. Consequently, it is evident that simply adding more examples and figures to the documentation is insufficient to improve the knowledge transfer rate of the API; these supplements must also be designed using the same principles as the documentation itself. However, determining how to visually represent technical knowledge, such as programming concepts and algorithms, is more difficult (Kuljis & Baldwin, 2000) due to the varying degrees of expertise and experience among individuals. As suggested by the data subjects, this may be due to people's varying preferences and understanding of metaphors and analogies used to explain abstract concepts, as well as their varying preferences for technical examples, such as code examples and interactive examples.

Learning from more practical examples, such as code- and interactive-based examples, is a crucial aspect of the API learning process for developers, as they instruct on the API's design and behavior (Robillard, 2009). However, reviewing examples may result in frustration if they are insufficiently tailored to the desired outcomes (Robillard, 2009). This was also implied by several study participants. Therefore, carefully engineering examples to match the most prevalent task objectives and use cases in order to provide genuine value to the documentation's end users. For example, in the case of an API that provides information from a database, it will be necessary to create examples that provide insight in primitive operations such as data retrieval and data manipulation.

Simplicity and clarity are also attributes that have significant impact on examples and

visualizations on their usefulness since they also are media used to convey information. In order to create clear examples and visualizations, it will be important to optimize the intrinsic cognitive load (van Gog et al., 2011). While it is important to teach learners through problem-solving to give them the tools to deconstruct and apply knowledge to other similar scenarios, it is equally important to contribute with appropriate examples that efficiently convey the knowledge in practice (van Gog et al., 2011). Expanding on this point of view, the participants in this study expressed a need for explicit problem-solving strategies and a lack of sufficient examples in the DHIS2 web API documentation. Moreover, the absence of either component has occasionally led to participant frustration. In turn, achieving a balance between problem-solving and examples could potentially improve the efficiency and efficacy of the learning process.

Code examples

The results of the study indicate that incorporating code examples can significantly enhance novice developers' API learning. Code examples can facilitate the segmentation of content, thereby improving the reader's comprehension of the subject matter. Providing code examples may also provide more contextual information that is specific to the API (Robillard & DeLine, 2011), provide packaged, reusable code functionalities (Gao et al., 2020), facilitate learning activities such as understanding the API's purpose and usage, and confirm or disprove their expectation of how the different API functionalities work (Robillard, 2009).

Previous research suggests that inadequately explained code examples are a common issue in API documentation (Uddin & Robillard, 2015). This study expands on this finding by highlighting the potential for code examples to reduce the amount of textual information that readers must comprehend to fully comprehend the content, thereby facilitating the creation of more precise and accurate descriptions. This emphasizes the importance of adequately explaining code examples in API documentation, an understudied area. In addition to less cognitive strain during the learning experience, using examples may also provide "best practice" approaches to entry-level developers in addition to providing rationale about the API's design (Robillard, 2009), which was also observed among the participants of this study.

One data subject drew attention to the disadvantages of overemphasizing code examples, as lengthy and numerous code examples can be difficult to work with. The statement suggests that shorter code examples that illustrate usage patterns and programming logics may be the most useful type of code examples that can be included in an API documentation. Robillard and DeLine (2011) argued for a similar sentiment in their study. The findings of this study supports the statement made by (Yin et al., 2021)

that developers seek longer and more detailed practical examples when code snippets do not satisfy their needs. Crowd documentation and other resources are also consulted when code snippets are insufficient for understanding the API, as they provide additional explanations for the code snippets and may provide additional information in the form of tutorials. Thus, the study confirms that tutorials and applications are commonly requested as examples in such situations.

Robillard (2009) defines a *tutorial* as a collection of code segments designed to instruct developers on particular API features (see chapter 2.3.5). It is evident from the findings that tutorial-formatted examples are the most beneficial for novice developers, considering their concerns about the shortage of examples for specific use cases. Also, it is important to be aware of the possibility of code snippets confusing entry-level developers in identifying which ones to adopt for their application and in translating the code snippets to their programming style (Gao et al., 2020), as is also evident in the findings regarding the DHIS2 API documentation usage experiences of developers with varying levels of expertise. However, it is not possible to cover all parts of the most common API use case scenarios, due to the quantity of API endpoints available in the API (Yin et al., 2021).

Robillard (2009) also mentions *applications* as a form of code examples that demonstrate API usage in bigger projects. These applications are valuable resources in providing developers a bigger picture of how the different API functionalities may be used and at the same time give examples of different programming and algorithmic perspectives. Some data subjects expressed the need for encouraging more developers to share their applications in the DHIS2 marketplace so that they can inspect and learn from existing solutions that use the same API for the above-mentioned reason.

However, as noted by data subjects, it is important to be aware of the fact that more experienced developers may not necessarily follow best practice approaches and might take several shortcuts in their codes. As a result, entry-level developers who *Code Review* these applications for learning purposes may end up becoming more confused about logical leaps or be subject to learning approaches that can increase the technical debt later in the development process. As a result, it is essential to consider the learning resource context and the type of example that developers use to comprehend the API, as these factors may affect the quality of information conveyed by the examples.

Furthermore, the relevance of the examples in order to facilitate a positive learning experience needs to be considered. As some data subjects have noted, there are times when developers discover information that turns out to be obsolete or deprecated. Hence, they

may find that the examples no longer adequately serve their purpose or that they are unable to learn from the examples provided. As the API's creators are responsible for updating its resources, it is crucial that its learning resources are also kept up-to-date. The difficulty lies in addressing out-of-date or deprecated code examples in crowd-sourced documentation, as the creator of the example is not required to continuously update the information provided. Consequently, it can be argued that relying solely on crowd documentation may not always provide the most up-to-date and efficient methods for achieving particular objectives. This observation is consistent with the claims of Robillard (2009) that examples in crowd documentation may become obsolete and no longer reflect current best practices.

Interactive examples

Data subjects expressed that interactive examples, such as sandbox developer environments, put technical details into practice and thus makes the information easier to understand and envision in actual use. As previously mentioned, these types of examples should illustrate how the user can expect to communicate with the API in actual use (see Chapter 2.3.5).

DHIS2 provides a sandbox developer environment for the various user interface components available through their react library (DHIS2 Team, 2023g), and some of the data subjects referred to this tool during the interviews when they discussed interactivity in learning environments. Therefore, a possible way of increasing API learning may be through providing an API sandbox environment to explore expected behavior during use. The data subjects mentioned interactive examples as an available tool, but very few data subjects discussed the idea of embedded interactive examples in the API documentation. Additionally, some data subjects noted the use of the API lifecycle tool Postman as a means to construct API requests before implementation; an interactive approach to understanding how to construct API requests. Thus, as they have already expressed the usefulness of interactive sandbox environments, a possibility is to introduce interactive API functionality examples that demonstrate the different results a callback function may provide with different states, filters and parameters as one can do in Postman.

Another method of incorporating interactivity into the API learning experience is to introduce coding platforms that provide learners with tasks to enhance their API-related development skills. A coding platform could provide a practical environment for developers to learn more about syntax, design, problem-solving principles and techniques, and their context (Zinovieva et al., 2021). Such a platform is available for learning web application programming in DHIS2; participants who had previously completed IN5320 referred to it as a "self-paced course." The course included interactive environments

in which students could explore the implementation of a given topic programmatically. Similar to the interactive examples offered in the self-paced course for learning web application development during the IN5320 course, a comparable tool for API testing could be advantageous for the API learning process.

Students can learn from interactive examples, but using numerous conceptual aspects simultaneously may increase their memory load. Blayney et al. (2015) discovered that it is more helpful to present the elements individually before demonstrating their relationship. Allowing learners to process and transfer knowledge in isolation organizes the knowledge in long-term memory, hence lowering cognitive load when they are required to investigate the relationship between the different elements. As a result, isolating certain interactive elements from each other may prove to be more effective for learners as there would be less reliance on working memory load which alleviates the cognitive load on the learners. Before exposing entry-level developers to larger use cases that incorporate multiple concepts simultaneously, it is important to consider presenting interactive examples for more fundamental concepts and processes.

6.1.3 Presentation: documentation structure

The format of the API documentation affected the attention span of learners and their ability to effectively navigate it, according to the study. Diverse opinions were expressed by respondents regarding the structure, ordering, and density of the content in both the documentation as a whole and the pages explaining various aspects of the API.

The impact of content presentation on learning

The data subjects have indicated that the content presentation hinders their growth and API learning when they are unable to locate the necessary information to overcome their obstacles. According to the study, the majority of data subjects find the DHIS2 API documentation to be overwhelming due to its abundance of information. This is consistent with the claim of Kapenieks (2013) that the design of web API documentation plays a crucial role in the API learning process because it can influence readers' cognitive processes, such as their perceptual abilities and working memory load. When designing the interface of web API documentation, it is crucial to consider cognitive processing levels in order to optimize the learning experience. Achieving a balance between the quantity of information conveyed and the usability of the interface can facilitate quick and effective API learning.

Participants believe that the design and structure of the documentation interface, as well as the prioritization of content order, have an impact on the learning process and expe-

rience. These topics highlight the significance of designing API resources with cognitive load theories in mind, especially for novice developers who may require more cognitive effort to comprehend the material than more experienced developers.

Documentation structure

Some data subjects expressed that presenting the documentation in a textual narrative (book-oriented) structure would make the reading process more straightforward and thereby reduce the cognitive burden of comprehending the material. Accordingly, a textual narrative structure reinforces the notion of presenting a cohesive story and demands less cognitive effort from readers in locating information. They also compared the textual narrative structure of the DHIS2 Developer Manual with the structure of DHIS2 web API documentation, clearly indicating a preference for the continuous narrative presentation rather than a fragmented approach. Content fragmentation has been reported to be a setback of using API documentations in previous research (Uddin & Robillard, 2015). Participants suggested that the inclusion of visualizations and code snippets does not disrupt the textual narrative structure because they are an essential component of the presented content.

Also, the results indicate that the fragmentation of content is a significant factor in the preference of the data subjects for the DHIS2 Developer Manual over the DHIS2 web API documentation (see chapter 2). As previously mentioned, the fragmentation of content makes information less discoverable, and developers most often look for coherent and linear presentations of the documentation to find the information they seek (Aitman, 2019; Robillard & DeLine, 2011). The data subjects expressed that the same factor was the reason for their preference for the DHIS2 Developer Manual.

By cognitive load theory it can be reasoned that the content presentation can dictate the amount of extraneous cognitive load that an individual has to use in order to process the information. This is particularly true if the content fragmentation forces users to seek through numerous fragmented pieces of content, and additional information that is unrelated to their objectives (Aitman, 2019). The preference of the participants for a continuous narrative stems from the detrimental effect of content fragmentation on increasing extraneous cognitive load. Thus, it can be argued that participants benefit from a continuous and cohesive narrative writing style, as it reduces cognitive load and improves API learning effectiveness. This would mean presenting documentation with as little fragmentation as possible when it is reasonable, ensuring that most content does not require heavy navigation.

The data subjects also suggested the need to mitigate the presentation of lengthy pages

of content to minimize the need for searching functions to locate the relevant information, if content segmentation is unavoidable. Similar findings are reported by Purchase and Worrill (2002), indicating that documentation should not require lengthy search processes and scrolling to obtain the "help" that it is intended to provide. One explanation for this could be that readers who read online often 'scan' the pages (Moran, 2020). In the case of programmers, they might completely skim through or ignore textual documentation and might favor using a list of techniques instead (Stylos & Myers, 2008).

While some developers adhere to traditional reading where the reader fixates on most words that carry the most meaning in the text (the commitment pattern), layer-cake scanning is more frequently used by developers according to previous research (Pernice, 2019b). According to the data subjects, developers frequently want to get started right away without having to read more information than necessary which would make the layer-cake scanning pattern an effective strategy. Incorporating the principles of the layer-cake scanning pattern into the content presentation design may therefore be a complementary strategy for entry-level developers' API learning strategies.

The study results show that when documentation topics are severely lengthy, readers utilize search tool functions to scan the content to find what they are looking for. Several data subjects reported employing this technique with the *browser's* search tools, writing relevant keywords and quickly scanning the occurrences of sentences to determine if they are relevant to their knowledge acquisition. Using the browser search tools have proven to be a somewhat successful strategy when searching through methods and functions even when the search terms do not exactly match the content (Stylos & Myers, 2008).

Some data subjects also reported that it was difficult to formulate accurate search queries for the in-page search tool of the API documentation, as documentation writers may have used different terminology or synonyms to describe the information being sought. This can be a significant obstacle, especially for entry-level developers who lack experience with APIs and may find it difficult to generate search keywords and synonyms that accurately capture the desired knowledge. Additionally, the way the readers approach searching through the content may also impact the learning experience, as many individuals have a tendency of assuming the relevance of the content by skimming through page titles (Hwang & Kuo, 2011).

The findings also suggest that the order in which content is presented can have a significant effect on the ability of readers to locate relevant information and their perception of the documentation as being overwhelming. The data subjects indicated a need for a quick start to API use with the minimum amount of information necessary to read. As

with the previous arguments, this suggests that the order of the information also has an impact on the extraneous load of strenuous reading. The layer-cake scanning approach may provide a strategy to alleviate the strenuous attention needed to read and process the content, but also ordering the content based on the value that it should provide quick and efficient API use also needs to be considered. The approach may serve as a useful test to determine how content should be ordered to prioritize the most essential information at the beginning.

The data subjects emphasized the significance of organizing the information in the API documentation in a manner that facilitates quick access to the essential details required for proficient API usage. Reordering the content based on its importance for quick comprehension and deployment may reduce the reader burden caused by unnecessary information. For instance, the placement of methods is crucial, as omitting important details such as the required connection of certain functions to other classes can have a negative impact on API usability (Stylos & Myers, 2008). Therefore, the API documentation must make it simpler to locate information about dependencies of other classes (Stylos & Myers, 2008). Incorporating brief mentions of these crucial, related classes is important to ensure that the developers deploy the API as quickly and easily as possible. As previously mentioned, the sentiment of Blayney et al. (2015) that isolated-interacting elements may be transferred to this context, as concepts that interact with each other need to be exposed to the API learners, while also instructing the learners about these concepts individually at first.

Hwang and Kuo (2011) examined the web-based problem-solving skills of elementary school students and observed that high-achieving students from the sample group used techniques such as reading the first sentence of each paragraph of an article to quickly grasp its content. They, along with other researchers, have indicated that the ability to summarize the article's concepts is a significant factor in achieving effective learning (Hwang & Kuo, 2011). They observed that students with strong summarization abilities understood the significance of the first sentence in each paragraph, which is crucial for comprehension. (Hwang & Kuo, 2011, p. 302). Similarly, the concept of front-loading sentences in API documentations to achieve the same effect has also been expressed to be a beneficial strategy (Aitman, 2019) (see Chapter 2). Also, Hwang and Kuo (2011) additionally provide an information summarizing instruction strategy that emphasizes the importance of identifying similar synonyms within the keyword and continuously iterating on the process of identifying keywords and searching in order to find information that is relevant to the problem at hand.

In conclusion, encouraging the front-loading style of content presentation may reduce the cognitive load of extraneous information on readers, which may improve the reading comprehension of entry-level developers and ease the search process. This strategy, however, requires that readers be prompted to scan the first sentences of each paragraph and repeat the formulation of similar keywords. Such a strategy can increase the likelihood of acquiring knowledge. To facilitate this process, it would be advantageous to organize the content based on its relevance to learning and using the various API functionalities, especially for entry-level developers who may have trouble locating relevant search terms and concepts that apply to their circumstances. Moreover, reading impatience due to the cognitive burden of strenuous reading and the lack of applicable keywords may also present difficulties. Consequently, enhancing information-gathering skills and modifying the presentation to adhere to a pace that facilitates rapid reading may result in a more efficient problem-solving process in API learning.

6.2 User engagement

6.2.1 Community-contributive documentation

Social forms of documentation was a concept implied by several of the data subjects as some form of user engagement and involvement in the content presented in the documentations. Some mentioned crowd documentation types such as StackOverflow; that is, documentations in which developers collaborate online to generate content from multiple perspectives (Delfim et al., 2016; Parnin & Treude, 2011). As indicated by the data subjects, the API documentation may lack sufficient quantities of explanations, examples, and use case scenarios (Delfim et al., 2016; Yin et al., 2021), which makes knowledge transfer from the API documentation to previous development experiences and knowledge transformation to their own use cases extremely difficult.

The social interactions from crowd documentations such as web forums, wikis and blogs contribute to providing wide accessibility and creating more content that is of use for developers (Parnin & Treude, 2011). Consequently, the developers receive a space where they can seek practical examples from different perspectives that can be adapted to their own use cases, making the knowledge transfer process more accessible and easier for the developers. The results indicate that social forms of resources are frequently employed when the knowledge provided in API documentation proves difficult to apply to their own use cases. This is often due to the absence of practical examples and alternative problem-solving approaches, rendering the API documentation less effective for troubleshooting and debugging purposes. Also, crowd documentations become more attractive to use when the API documentation fails at transferring enough knowledge to the users.

The benefits of using crowd documentation for API learning may be the same as the ones for inquiry based learning strategies in educational contexts. The strategy entails carrying out a self-directed learning process with active participation and the responsibility for discovering new knowledge (Pedaste et al., 2015). Crowd documentations offer different views and perspectives that are relevant for the learner in order to understand more about the learning topic by enabling more practical guidance and quick communication with other developers (Kapenieks, 2013).

In this way, crowd documentations become collaborative resources that provide users with the opportunity to gain knowledge from a variety of informational formats, such as diagrams and instructional videos, that illustrate various problem-solving strategies for overcoming similar or identical obstacles. Laal and Ghodsi (2012) also noted that the learning strategy further developed the learners' problem solving skills through engagement in idea formulation, discussion and receiving immediate feedback. Therefore, by giving the users a contributive role in the documentation curation process, it might increase the likelihood of them learning different strategies from other developers and as a result learn new problem-solving skills that they can utilize for later API learning challenges.

As a form of social documentation, data subjects examined application codes associated with their desired use cases in order to acquire and implement techniques and knowledge for their own projects. This highlights an additional contribution of crowd documentations. Some data subjects suggested sharing custom applications for inspection purposes within the community as a strategy to promote learning, as this would provide greater insight into solutions from diverse perspectives. In addition, some data subjects noted that other community members may have already overcome the obstacle; therefore, promoting greater application and code sharing within the community could save developers time and effort. Examining diverse perspectives on problem-solving can also introduce entry-level developers to a variety of skills that can positively influence their programming and problem-solving strategies. Evidence of said effect is made apparent in the study by Smith and MacGregor (1992). However, as noted by Robillard (2009), the available user generated online resources may go unchecked for obsolescence, increasing the probability of the material being outdated or deprecated. Hence, crowd documentation cannot be the main resource to obtaining more knowledge at all times, and needs to be triangulated with the API creators' documentation for version control, which places more responsibility on the learner instead of the content creator.

It can be argued that encouraging greater community engagement and participation in documentation curation would improve the API learning experience for novice develop-

ers. Social interactions, such as allowing users to suggest code snippets or share their use case scenarios, can transform API documentation into a learner-centered environment beneficial to entry-level developers. Users must exercise caution when relying on crowd-sourced content and triangulate it with API documentation to ensure its accuracy and avoid outdated information.

6.2.2 Engagement and incremental learning through gamification

One of the data subjects mentioned gamification as a possible strategy for presenting information in a more user-friendly manner, tailored to the level of expertise of the target audience. It may be possible to create a less steep learning curve for entry-level developers and make the learning modules easier to comprehend by incorporating gamification elements into the documentation. Gamification has also been reported to increase engagement and motivation for its peers (Elshiekh & Butgerit, 2017; Falkner & Sheard, 2019). Though this tool is great in increasing the learners' performance through boosting motivation and providing immediate feedback (Elshiekh & Butgerit, 2017), it may only be appropriate to be applied to teaching programming related aspects of API learning, such as callback writing as those are instructional resources that only the entry-level developers have a use for.

6.3 Differing levels of expertise

According to the data subjects, there are varying levels of expected expertise for DHIS2 API learning resources. When interacting with various types of content, developers are presented with varying levels of expectations for their level of expertise. Consequently, some API learning materials are written in a way that is more accessible and understandable for entry-level developers, while others require a significant amount of knowledge about the data processed and provided by DHIS2. Some instructional materials emphasize the programming aspects of the API.

Providing instructions and material adjusted to the learners' level of prior knowledge may naturally generate better experiences in entry-level developers' learning process (Blayney et al., 2015). For experienced developers, this could become a burden and result in extraneous cognitive load as they have to navigate through content that is not relevant to their level of expertise (Blayney et al., 2015). Therefore, some learning materials will be more difficult to comprehend if the learner lacks fundamental knowledge about the various types of data stored in the database. In addition, the developers perceive differing expectations regarding what the learning material is meant to address, as the expected level of knowledge for a particular topic is not clear. Adjusting the learning material to

the learners' levels of expertise may result in reducing the extraneous cognitive load and thus improving the learning output (Blayney et al., 2015). Multiple documentations with varying degrees of content complexity can increase maintenance and revision requirements for documentation producers. A different approach would be to organize the information based on the most important takeaways for a given topic and present more advanced information as the reader progresses through the documentation. This strategy has the potential to reduce the need for multiple documentations while still catering to developers with varying levels of expertise.

6.3.1 Intended documentation usage

According to certain data subjects, the purpose of the DHIS2 web API documentation was unclear. This ambiguity was caused by the presentation style of the documentation, which made it unclear whether it was intended as a guide for operation or to define the various callback functions, their purpose, and usage. This ambiguity caused readers to have differing expectations regarding the knowledge they could obtain from the API documentation. Experienced developers expected a list of callback functions, while novice developers expected a comprehensive and detailed instructional guide. Similar results were implied in previous studies in a setting where the participants were separated based on exposure to "exploratory" problem solving tasks (Piccioni et al., 2013).

Developers' API learning experiences are significantly affected by their level of expertise, as the choice between the two documentation formats is dependent on prior knowledge. When presented with explanations of callback functions, novice developers unfamiliar with the data environment of the API may prefer an instructional guide with more information on metadata. Yet, more experienced developers who are already familiar with the metadata may find it tedious to follow lengthy instructions and may prefer a concise list of callback functions.

Thus, it is becoming evident that balancing the intrinsic, extraneous, and germane cognitive loads on individuals is one of the most significant issues for developers when utilizing API knowledge resources. The lower the extraneous load must be in order for entry-level developers to learn most efficiently, the greater the intrinsic load (Guadagnoli & Lee, 2004). When the purpose of the documentation is ambiguous in terms of its scope and approach, it can increase the cognitive load of the users. Therefore, it is essential to clearly state the content and objectives of the documentation and to direct users to the most relevant resources. This can reduce cognitive load and enhance the user experience overall.

6.3.2 Subject difficulty and expertise level mismatch

Several data subjects touched upon the differences in the degree of difficulty of the content and their own level of expertise. The sample group is diverse in their experiential level, and the findings reflect the differences of expectancy between the groups as a result. Particularly, the less experienced subjects compared the content of the documentation with their own prior knowledge, and all of the data subjects experienced that the information provided was either at a lower or a higher level than they anticipated from the API documentation. It is when these anticipations are not met that they often are burdened with more extraneous load as they have to find information elsewhere, or identify what knowledge is missing for them to be able to understand the API documentation. To counteract this, it is essential to align the documentation with the readers' expectations and adjust the level of difficulty accordingly in order to reduce extraneous load and improve learning outcomes.

Both Nelson and Eliasz (2023) and Guadagnoli and Lee (2004) bring up the concept of Desirable Difficulty (DD) in relation to educational strategies to optimize the learning performance using a Challenge Point Framework (see figure 6.3); it builds on the idea that long-term learning occurs after the learners have encountered and solved an optimal challenge.

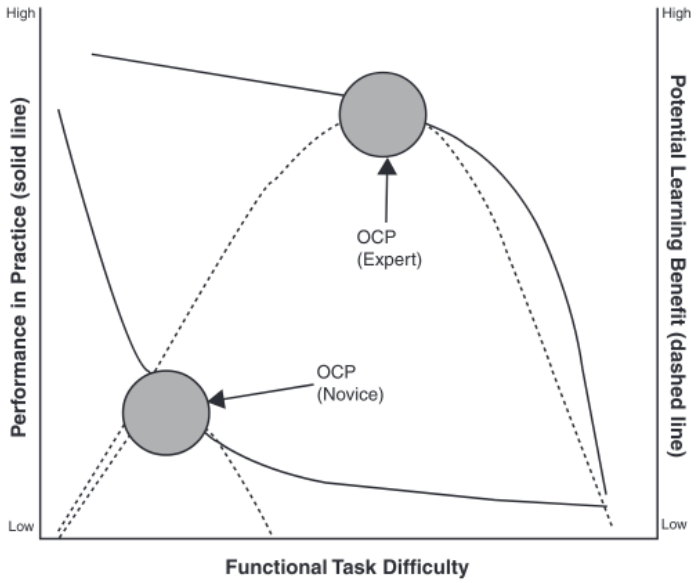

Figure 6.3: A visual representation of the learners' performance on a learning task (cognitive effort) and the level of difficulty of the task that affects long-term learning, and the differences between novice and expert learners (Guadagnoli & Lee, 2004, p. 217). OCP = Optimal Challenge Point.

This concept can be applied to determine the appropriate level of difficulty for the intended audience in educational materials. As depicted in figure 6.3, the greater the effort a learner must exert to comprehend the material, the greater the cognitive capacity they must employ. When the material presents a high level of intrinsic load due to its complexity, the learner must exert considerable effort to comprehend the concepts (Nelson & Eliasz, 2023). Efficient learning necessitates ensuring that the complexity of the content does not exceed the learners' capacity for sustained cognitive activity. In other words, for learning to take place, there must be a suitable level of intrinsic load, but the gap between the learners' current level of knowledge and the desired level of knowledge after utilizing the material cannot be excessively wide (Costley & Lange, 2018). The desired level of difficulty implies that there must be an intrinsic challenge in order for learning to take place, but too much of a challenge undermines students' motivation and performance.

The assessed performance and task complexity required for optimal learning varies significantly between novice and experienced learners, As shown in figure 6.3. Finding a Desired Difficulty that corresponds with both target groups needs to be addressed in order to provide them with API learning materials that are efficient in use. An alternative could be providing separate documentations for the target groups. As reported by Nelson and Eliasz (2023), it is therefore also important to ensure that their performance does not drop significantly, as it will result in demotivation that eventually leads to poor learning outcomes. Through this theory, it is evident that learners' performance and motivation are strongly determined by their prior knowledge before using these resources.

Thus, providing learners at the introductory level with excessively advanced content or challenging tasks can negatively affect their performance and result in a suboptimal learning experience. Alternatively, if it is not possible to accommodate both ends of the target audience, it may be more beneficial to create separate documentations that address varying levels of complexity in order to avoid demotivation and poor performance, at the expense of increasing maintenance for the documentation producers. To achieve this, it is essential to tailor the content to the diverse learner population. Due to the varying degrees of performance decline over time among different target groups and the need to create content that is sufficiently advanced to satisfy both groups, resolving this tension can be difficult. As the learner progresses through the information, incrementally adapting the content to reflect the increasing level of difficulty is one potential solution.

6.3.3 Concepts and terminologies

The findings suggested that context is crucial for understanding concepts and terminologies, and the lack of providing context to certain terms make them too imprecise. Baumann and Graves (2010) noted the adverse setbacks that occur from not providing enough context by differentiating between domain-specific academic vocabulary and general academic vocabulary. They explain that domain-specific vocabulary, also referred to as technical vocabulary, are terms and expressions found in technical writings (Baumann & Graves, 2010, p. 6). As noted in the findings, the data subjects expressed that the DHIS2 web API documentation would benefit from specifying key terms in order to reduce confusion.

The data subjects claimed that specifying and explaining key, technical vocabulary for the different contexts the documentations are adapted to would reduce vagueness of the information and improve the understanding. The terminologies that are noted to be vague for novice learners are from the generic academic vocabulary, for example the term 'data' as mentioned in the findings. These kinds of words are abstract and often change their meaning in different content areas, and instructors often assume that the learners know these meanings (Baumann & Graves, 2010, p. 6). Attempting to understand the true meaning of these words in a specific domain may therefore present more cognitive load on the entry-level developers who have minimal exposure to the domain. Hence, an important remark is ensuring that the vocabulary used in the documentations matches the readers' expectations as much as possible to avoid misinterpretations (Novick et al., 2009).

It is also important to note that the general academic vocabulary may aid the readers in searching and identifying information needed for their problem solving when they are not as familiar with the domain and its vocabulary. The readers need to evaluate and determine the accuracy of the search terms they use and their relevance of the information they find to the obstacle they need to overcome (Hwang & Kuo, 2011). Hence, completely avoiding general academic vocabulary is not only unattainable, but it also decreases the searching abilities of the readers. Domain-specific vocabulary and general vocabulary with abstract meaning must be specified and interpreted for readers without adding excessive reading strain. If not in a separate document, discussing the easiest conventions and progressing in a descending order of significance to understand the knowledge provided for certain topics may help solve this problem.

Hence, specifying key terms and concepts reduces the intrinsic load on learners because they use less cognitive capacity to interpret imprecise terms to the domain. The decrease intensifies the unnecessary load on learners since they must read more information before

reading the information that may answer their questions. Linking terms and concepts to an external document with definitions and explanations may help. If not, briefly defining the basic academic terminology for the domain context and including it in the text are also possible solutions to make understanding the topic easier.

6.3.4 Problem-solving strategies

Various participants in the data gathering process described how they approach problem solving in their projects when using APIs. In iteration 2, a significant number of developers stated that learning problem-solving skills is essential for overcoming the majority of API implementation obstacles. These problem-solving strategies can be viewed as the processes an individual participates in by engaging in goal-directed cognitive activities in order to transform an existing situation into a desired situation (Aukrust, 2011). In this study, the goal would entail successful operation of the API callback function that assists other functionalities in their applications, or some way of data manipulation.

As facing and overcoming challenges is associated with long-term learning (Nelson & Eliasz, 2023), developing robust problem-solving skills is crucial in tackling these obstacles and supporting an individual's long-term growth and development. Hence, understanding how developers engage in problem-solving activities may be beneficial in order to guide entry-level developers in their API learning journey. According to the findings, two essential components of their learning journey are locating the appropriate information and whether they engage with their challenges proactively or reactively.

Information searching abilities

One aspect of problem solving is information gathering to comprehend enough about the situation and finding approaches to overcoming obstacles. As previously mentioned, some learners may not be efficiently forming several search queries with synonyms (Hwang & Kuo, 2011). A phenomenon reported in the findings is some data subjects utilizing search engines such as Google to find the relevant information in the API documentations as they provide results that also consider synonyms (Net Age, 2010; Southern, 2022), hence providing results to the reader when they search with specific domain terms.

Consequently, it is important to make readers aware of improving their problem solving processes by encouraging them to use more synonyms when they search for information to gain the most out of a single learning resource. If that is not possible, the documentation producers need to consider implementing or improving internal search engines provided by the documentation to incorporate the use of synonyms and provide appropriate results.

Additionally, the data subjetcs' dependence on web search engines to find learning material tightly tied to the technology has also been briefly mentioned has also been observed in the study by Parnin and Treude (2011). Formulating accurate search queries is an important skill that more experienced learners become proficient at due to their domain knowledge. Therefore, novice learners should opt to increase their own knowledge of synonyms tied to concepts and terms they know occur often in most documentations.

Proactive and reactive strategies

A commonality between the different sample groups is the implication that entry-level learners often retrieve more server data than needed to accomplish their goals, causing challenges with segregating data. Two proactive problem-solving strategies were mentioned by the more experienced developers in the findings; the strategy of establishing a minimum viable payload, and the test-driven programming strategy. The purpose of both of these techniques is to restrict API responses and mitigate extraneous data that is not necessary to meet their goals of their functionalities. The key understanding from these proactive strategies is an attitude change towards reducing redundancy and producing resilient functionalities by iterating on the solutions by testing until they receive a resilient solution that meets their end-goal.

The positive impact these approaches bring to the learning developers is a reflection on how and why they solve their problems the way they do. They are made to reflect on whether their approach is resilient and that it is not redundant. As a result, they may notice implementation patterns that can be generalized into other API implementation scenarios. By doing so, the germane cognitive load may become optimized by reducing the cognitive effort of re-imagining a solution procedure every time they encounter similar API implementation goals. Additionally, inspecting requests, whether their own or by others, may teach the learners more about data elements and their interactions with each other. Thus, the entry-level developers may discover these techniques are more efficient and effective in reaching their goals faster (van Gog et al., 2011). These proactive strategies may play the role as *worked examples* over time, and provide developers with a problem solving schema that increases their understanding of better API implementation practices. The self-explanatory nature of the approaches guide them in identifying information and managing the intrinsic load for the learners (Gao et al., 2020).

The findings indicate that Trial-and-Error is a prevalent problem-solving strategy, particularly among novice data practitioners. When the API documentation is unclear about how the functions operate or when they have trouble understanding the metadata relationships, they frequently resort to brute-force methods to obtain the data required for their operations. In contrast to the other tactics, Trial-and-Error is a type of means-end

analysis in which individuals search through many options without a plan to approach the desired end state (van Gog et al., 2011). The method is a poor problem-solving skill on its own because there are typically no aims of learning and implementing best practices, but rather a desire for a quick achievement of an end result.

With less thought on what they are doing and why they are making decisions, the output of learning is drastically diminished. The strategy may eventually result in a solution while imposing the learners, especially the entry-level learners, to significant amounts of extraneous load as a consequence. Hence, in order to aid entry-level developers in their API learning journey it will be important to encourage reflection while in action and for instructional API learning material to provide high degrees of instructional support (Blayney et al., 2015) when they are learning by doing. Switching between exploration and using instructional guidance may increase the successful outcomes of the problem solving strategies (Andrade et al., 2009). However, it is also important to keep in mind that significantly adjusting the documentation for individuals with more instructional support may increase the extraneous load for individuals who possess more prior knowledge (Blayney et al., 2015).

It may be claimed that more experienced developers use proactive problem-solving tactics than less experienced developers, thereby enhancing their productivity. While entry-level developers rely on reactive methods such as Trial-and-Error, their learning output is drastically reduced because they do not build a strategy nor reflect on the methods they are employing to get results. In essence, it is crucial to encourage entry-level developers to think on and design problem-solving schemata that can be applied when they are introduced to new APIs. By revisiting their worked examples, they may become more adept at identifying patterns and more efficient in overcoming obstacles.

In order for the documentation to accommodate both sides of the intended audience, it would be advisable to train entry-level developers on how to create schemata for problem-solving when they find issues, and to use these situations as examples for future study. By doing so, it will decrease the instructional support of the documentation resulting in less extraneous load and an optimized germane load for the more experienced developers. Thus, it will be advantageous for API learning to reflect while in action and establish schemata for problem-solving techniques of the present circumstance.

6.3.5 Differing need for guidance

The data subjects imply that there is, naturally, a difference between the developers with little prior knowledge and the experienced developers. It is evident that the more experienced developers were burdened by documentation that provided high degrees of instructional support, and claimed that these instructions became counterproductive, implying the increase in extraneous cognitive load when they are required to process redundant information or knowledge that is irrelevant for their scenarios. The differences between the need for guidance between entry-level and experienced developers may be the result of the expertise reversal effect (Blayney et al., 2015) which points out the inverse correlative relationship between the amount of instructional guidance and the extraneous load caused by redundancy and irrelevance.

Experienced individuals create schemata from problem-solving that assist the breakdown of obstacles into sub-problems and the adaptation of identified solutions to future situations (Kuljis & Baldwin, 2000). In a sense, this enhances the domain expertise of the persons who develop these schemata. This exemplifies one of the most major distinctions between novice and seasoned developers; their methods for organizing domain information differs (Kuljis & Baldwin, 2000). The observation by Blayney et al. (2015) that more experienced learners achieve decreased learning outcomes when using documentations with a high degree of instructional support may be explained through the aforementioned observations.

It becomes clear that one of the major implications for newcomer developers is the appropriate creation of problem solving schemata when they use practical approaches to understanding the API. Not surprisingly, they lack sufficient prior knowledge to create effective problem-solving schemata to overcome the challenges encountered during their API learning journey and require more instructional support than their more experienced peers. Very thorough instructions may be beneficial for novice developers, but they are counter-intuitive for experienced developers. Hence, encouraging entry-level developers to understand how to efficiently create problem-solving schemata could be a potential answer to this obstacle. In order to reduce the extraneous load on the more experienced developers and adapt the documentation for the less experienced, it may be advantageous to provide instructional guidance when it is absolutely necessary under the documentation topic and redirect them to a more detailed approach using hyperlinks if they require more information.

6.4 Data Structure and API Functionality Comprehension

The process of learning the data structure has been recognized as one of the greatest obstacles for entry-level developers that engage in DHIS2 platform application development. To achieve their goal of developing an application for the health platform, it is also necessary for them to comprehend how certain callback functions and other practical tools should be utilized. Based on the findings, understanding the data structure can be believed to increase the intrinsic load and level of difficulty of the API learning experience for entry-level developers. This may occur when they independently read the API documentation and other knowledge resources, or when they receive instructional guidance from developer onboarding processes, mentoring, or similar activities.

6.4.1 Understanding the data structure

As mentioned, understanding data structure and concepts has the greatest impact on the intrinsic workload of entry-level software developers. The experienced data subjects recommended that entry-level developers learn the interrelationships between the various data elements in order to attain a level of knowledge that allows them to solve problems effectively. Abstracting the knowledge would allow them to perform effective problem solving by reducing the intrinsic load on the individuals, thereby optimizing the working memory load for information acquisition and retention as they would not be required to comprehend the entire rationale (Falkner & Sheard, 2019). Additionally, it may reduce the extraneous load by presenting the data structure concepts as much simpler, abstract entities, and increase the germane load that aids in learning which is beneficial for API learning (Gao et al., 2020).

However, as suggested by the entry-level data subjects, a significant challenge is transferring this knowledge to long-term memory when they cannot independently abstract the concepts presented in text. Without prior knowledge of these concepts, it is difficult to abstract and comprehend the information being read (Srull, 1983). In order to improve the learning experience of entry-level students, it is the responsibility of the learning material providers to guide students with the most fundamental knowledge to abstract concepts that they can utilize to understand the rest of the API. As evidenced by the findings, the issue may not always be a lack of information, but rather information that is inadequately accessible or invisible to users. An example of this situation, as mentioned in the findings, is the concealment of the metadata information page in the toolbar of the DHIS2 web API documentation. Hence, the documentation needs to make other appropriate learning resources more visible and guide them in a proper learning path.

The emergence of obstacles later in the learning process constitutes an additional constraint. As mentioned by data subjects, this can occur when developers discover that certain API features are no longer supported, deprecated, or removed after an update. Not only must the developers revisit the API knowledge resources to identify the location of the change, but they must also adjust their understanding of the API. When inexperienced developers are required to adapt recently acquired knowledge to potentially significant changes, the obstacle manifests; their mental schemata, which are a component of their prior knowledge, must adapt to the new changes. The difficulty is overcoming bias and erroneous interpretations in their learning journey (Srull, 1983)), which can be especially difficult for entry-level developers who lack the experience to recognize the errors that need to be corrected. One possible solution would be to provide the developers with additional information on how to alter their solutions for a smoother transition to new changes.

In sum, it is evident that understanding the data structure and concepts associated with the API increases the difficulty of learning it. It is essential to manage the workload of entry-level developers in order to improve their experiences. Exploring the constraints of understanding the data structure reveals that concept abstraction is a necessary skill for entry-level developers in order to decrease the intrinsic- and extraneous load while increasing the germane load, to facilitate a smoother knowledge transfer to the long-term memory and enhance the learning output. Also, obstacles that arise later in the learning process as a result of knowledge- and API updates necessitate providing developers with clear communication about the updates so that they can update their mental models and solutions to reflect contemporary knowledge.

6.4.2 Learning paths and pedagogical contexts

The data subjects provided feedback on the various contexts surrounding API learning and how they may affect or benefit the process for entry-level developers. The findings identified three types of instructional programs, such as student courses that facilitate a development project, onboarding processes, and team coaching. In addition, variations in collaborative practical learning programs were also identified. All of these instructional program variants rely on more experienced developers transmitting their knowledge and best practices to a less-experienced group of learners, or on the collaboration of multiple developers to produce learning outputs.

Instructional design of learning programs

There are several challenges related to these types of courses that may stem from the course design itself or the mental processes of the students (Garousi et al., 2020). The effect of the distribution of theory and practice in student courses is one of the most important findings. The majority of respondents agreed that a 50%/50% split between API application development theory and practice would be optimal for courses that include application development projects. Some also argued that increasing the amount of API usage practice in a course could improve the learning of problem-solving strategies by forcing students to think more independently. Nonetheless, research indicates that the distribution of theory and practice in courses influences how students perceive "industry-aligned" programs (Garousi et al., 2020). The challenge is addressing enough theory so that students can comprehend the rationale behind why various API aspects are the way they are, and providing enough practical experience so that they can use the API.

The split between theory and practice will be affected by another factor; the complexity of the course (Garousi et al., 2020). As easier theoretical learning could potentially decrease the time needed by the students to understand what they are going to do in their projects, the time to facilitate practical exercises may increase. Yet, it is challenging to address API problem-solving strategies without considering that the students additionally need to understand other aspects of application development and programming languages. By decreasing the intrinsic load of the theoretical learning material, it may therefore lead to less time spent on theory and more time for practical learning which gives the students a self-explorative approach to API learning which can increase the likelihood of them learning problem-solving strategies. Conversely, switching between self-exploration and using instructional guides may support more effective problem solving for the entry-level developers that lead to more successful outcomes (Andrade et al., 2009).

Other challenges with student courses can be low motivation, increased cognitive load and challenges with tools (Garousi et al., 2020). As explained with the level of difficulty, student motivation has a direct impact on their learning performance (Guadagnoli & Lee, 2004; Nelson & Eliasz, 2023). Low motivation leads to low germane load, thus reducing the learning output for students especially when the intrinsic load is high (Costley & Lange, 2018). Moreover, when students are unable to carry out the planned actions with unfamiliar tools, they become less motivated. As a result, students may not fathom what they need assistance with and what they must learn to achieve a desired learning goal. One data subject emphasized the importance of facilitating learning through incremental development, such as through continuous exchange between theory and practice similar to iterative sprints, so that students can formulate small, focused problems that make it easier to identify the problem area. Other studies have reached the same sentiment

(Garousi et al., 2020). Additionally, van Gog et al. (2011) also discovered that instructions that rely heavily on example studies are more effective for less-experienced learners; and as previously mentioned incorporating Code Reviews or application inspection may therefore be beneficial (see section 2.3.5).

Challenges with finding learning paths

One implied challenge by the data subjects is finding the appropriate learning path to understand the API. The findings suggest that some individuals are for example unaware of where to find in-depth information about the metadata in DHIS2. This has have implications for individuals who rely on self-exploration for learning, such as application development project students. These individuals may be unaware of the significance of comprehending the complex data structure in order to effectively extract and manipulate data from the various callback functions. Similar findings have also been reported where inexperienced developers struggle to find proper learning entries to learn the API (Yin et al., 2021). They also note that it can be challenging for inexperienced developers to figure out which foundational callbacks they need to study to learn the API as these callbacks may have complex relations between each other which are discursive to understand with little to no prior knowledge (Yin et al., 2021).

In order to improve the API learning experience for entry-level developers, it would be valuable to present the learners with a proper learning path when they first encounter the API learning material to aid in self-learning processes. Providing the readers with a guide or a prioritized list of resources for understanding the API could greatly enhance their learning experience. This would involve identifying which knowledge resources are suitable and for what purpose. For DHIS2, for instance, it may be appropriate to provide a link directory beginning with metadata information, followed by documentation for instructional courses, and concluding with API documentation. By providing the directory as a starting resource, individuals with more experience can determine which resource to begin with based on whether or not they possess sufficient prior knowledge to skip resources. Cross-referencing these resources with each other would also increase the likelihood of documentation readers to find proposed learning paths when they engage with particular learning resources.

Content obsoleteness

As implied earlier, the obstacles may sometimes not be in learning something new, but in updating existing knowledge. These situations occur when the API functions are modified in a way that significantly affects the operation of callback functions. As a result, instructional programs that teach these technologies may not be aware of the changes when instructing the learners, resulting in an increase in the learners' extraneous load. It also diminishes the quality of the learning resource if it fails to adequately address how the changes affect the applications. To mitigate the problems caused by outdated documentation, it will be essential for the documentation producers to provide sufficient information on how the changes impact solutions that use older versions of the API and how they should modify their solutions if the changes significantly impact the data operations or functionalities. Furthermore, instructors must ensure that their instructional materials are up to date and, in the case of recent API changes, be willing to learn alongside their students and engage in collaborative exploration to aid comprehension.

Collaborative practical learning

In addition to learning contexts where the learners teach each other theoretical knowledge, there has also been suggestions of practical collaborative learning such as group project work or assignments. Such techniques can be pair programming, code reviewing or lab programming sessions that can provide the students an opportunity to collaborate in order to understand programming logic and reflect together (Costley & Lange, 2018; Falkner & Sheard, 2019). The positive effects of collaborative practical learning may be increased task engagement, reflection and meaningful engagement with the learning content (Costley & Lange, 2018). It can be implied that motivation and performance may be increased through collaborative techniques also in situations that involve more programming and development in order to increase learning output.

Other perspectives derived from the findings indicate that collaborative practical learning provides students with colloquium groups in which they can assist one another in understanding practical API usage. This would also lead to meaningful interactions where they teach each other how to write callback functions. The use of collaborative practical learning strategies would facilitate inquiry-based learning for the learners, which requires active participation and self-directed investigation of new knowledge (Kapenieks, 2013; Pedaste et al., 2015). To construct their knowledge, learners would be required to follow practices similar to those of more experienced developers (Kapenieks, 2013), and reflecting in collaboration with other learners would provide additional perspectives during the process.

6.5 API learning guidelines

Tracing back to the concatenated guideline the developers provided from the findings (see figure 5.3.1), some edits can be made to facilitate a stronger process. As mentioned earlier, information summarizing is an important aspect of finding and understanding the content through summarization or abstraction (Hwang & Kuo, 2011). For the reiterative guidelines, it will be important to make sure the learners adequately search for the information they need in an effective manner. The guidelines consider practical (figure 6.4) and theoretical (figure 6.5) learning of the API.

Practical guideline for API learning

1. **Look at what's possible**
    - Look at the documentation and Trial-and-Error interchangeably
        - Set a learning goal for Trial-and-Error
        - Promote reflection of input and output of the API requests
        - Collaborative approaches: pair-programming, lab sessions
    - Use community-contributive (crowd) documentation
        - Ask developer communities, use learning academy
        - Use web forums, wikis and blogs
    - Study existing apps
        - Code reviewing
2. **Test queries independently (i.e. in Postman)**
    - Do the results meet the desired end-goal?
3. **Establish a minimum viable payload (MVP)**
    - Create as accurate and small API payloads as possible
    - Refactor to optimize API requests from independent testing
    - Avoid over-extraction of data to minimize additional filtering through programming
4. **Implement in the application**

Figure 6.4: Learning guidelines of a practical approach to understanding API through use

Theoretical guideline for API learning

- **Study visualizations and use cases**
  - Study illustrations, code examples and interactive examples
- **Search for additional knowledge resources**
  - Select keywords that match the desired concepts and create appropriate search queries
  - Evaluate the search results and investigate the relevance to the problem
  - If search was unsuccessful, identify possible synonyms and reiterate construction of appropriate search queries
- **Use community-contributive (crowd) documentation**
  - Ask developer communities, use learning academy
  - Use web forums, wikis and blogs

Figure 6.5: Learning guideline of a theoretical approach to understanding API concepts

7 | Conclusion

The theory generated from the grounded theory process provides an answer to the research question of this study. The sub-questions have been used to address smaller intricacies needed to adequately address the main research question. The chapter is followed by the contributions of the study, limitations, validity and reliability evaluation and future research avenues. The main research question will be addressed first, which is then followed by the sub questions.

RQ: *How can API knowledge resources be optimized using usability principles for entry-level platform application developers?*

The study concludes that the usability principles have a significant impact on the learner's intrinsic, extraneous, and germane cognitive loads. To optimize the knowledge resources, it is important to handle these cognitive loads via these usability principles. The learnability of the content presented in an API knowledge resource is influenced by the intrinsic load of the information, which entails the information's simplicity or depth, the prior knowledge level of the readers, and the variety of engagement methods exposed to the learner. The presentation of the information, the accessibility of information, and the use of examples and use cases that are tailored to the learners' levels of expertise all have an impact on the learning efficiency, which is correlated with the extraneous load. The memorability principle, which determines whether the knowledge conveyed in an API knowledge resource can be retained and transferred to long-term memory, is influenced by the content's degree of difficulty and the learning path strategies learners employ to approach information retention. The research demonstrates that addressing these learning factors in order to optimize the intrinsic and germane load and reduce the extraneous load for both sides of the target audience is necessary for fostering positive learning experiences.

SQ 1: *What specific obstacles do entry-level application developers face when attempting to learn APIs and how can they be addressed?*

When attempting to learn APIs, novice application developers face a number of obstacles, including unclear displays of capabilities, a lack of formal definition, and insufficient information on topics and issues. It can be challenging to ensure term comprehension

while avoiding cognitive overload. Incomplete examples and a lack of supporting documentation that explains the metadata model can also present difficulties. In addition, content is frequently written with the assumption that readers are already familiar with the metadata structure or API, and not all information is documented or up-to-date. It can be difficult to interpret examples written by highly-skilled developers, and it can be challenging to find the information needed to overcome obstacles. Also, the developer community, which is not always very active, may take some time to respond.

SQ 2: *How do entry-level developers overcome their obstacles currently?*

To overcome obstacles, entry-level developers utilize a variety of techniques, including trial-and-error, asking community members, and crowd-sourced documentation. Due to the fact that trial-and-error is a reactive problem-solving strategy in which developers rely on experimentation and repeated attempts to solve problems, it does not encourage reflection and may not provide efficient learning. Another strategy is to request assistance and direction from community members, either online or within their own teams. However, the response time of online forums may further hinder their learning environment. Developers also consult community-contributed documentation, such as YouTube and StackOverflow, in search of applicable tutorials and code snippets.

SQ 3: *How can an API documentation be written to support learning and efficient problem-solving?*

This study employs usability principles to highlight API documentation considerations that promote learning and effective problem solving. Consider creating documentation that allows filtering by the target audience to avoid multiple API documentations and provide a clearer view of expected levels of prior knowledge for each topic's difficulty level. Separating theoretical knowledge (such as metadata information) and technical knowledge enables readers to be more selective in what they choose to read and to skip over material they already know. Specifying key terms and concepts in a definition database with explanations, adjusting search capabilities to provide proximity word suggestions, presenting content in a clear and organized manner, and using examples and visualizations enhance API documentations and facilitate effective problem solving. Using code examples, interactive examples, and front-loading sentences, the API documentation assists novice developers in problem-solving. Involving the community in the problem-solving process by having them suggest code snippets or share their applications and use case scenarios for documentation review may speed up the problem solving process.

SQ 4: *How should the knowledge resource producers communicate with and guide the development community, given the diversity of the user base?*

To effectively guide and communicate with the development community, vendors of learning materials can provide forums or other open platforms for users to collaborate and ask questions. Moreover, segmenting theoretical and technical knowledge, concepts, and definitions in documentations can facilitate content reuse and prevent redundancy. In addition, vendors can make content filterable based on the user's level of expertise in order to provide a comprehensive and individualized learning experience. To meet the diverse needs of the user base, it is also important to provide clear communication about the intended usage of the documentation, which could be accomplished, for example, by front-loading sentences and ordering content based on how fundamentally important the knowledge is to efficiently learn and use the API. Additionally, encouraging application sharing in community resources (i.e. application marketplaces related to the platform vendor) for code reviewing is also beneficial for self-directed learning. Lastly, instructional learning paths should encourage learners' reflection and collaboration to improve their individual and collaborative performance.

7.1 Contributions

Prior research (Piccioni et al., 2013; Robillard & DeLine, 2011; Uddin & Robillard, 2015) has primarily focused on experienced developers, and there is a lack of information from a beginner's perspective. While previous studies examine how to make these API learning resources more effective to employ the API, they do not place significant focus on how the learning process is designed and ensued for inexperienced developers. Additionally, they do not consider the varying levels of expertise and its effect on the learning experiences of the documentation users. This research has contributed with a usability-focused perspective on technical knowledge resources related to API learning processes to fill this knowledge gap caused by the lack of representation of inexperienced developers in the studies. The learning resources must address both novice and experienced developers with varying levels of application development expertise and experience in order to be an effective resource for the different target audiences

The study employed an interdisciplinary approach by incorporating pedagogical and psychological theories, such as the cognitive load theory, to make entry-level technical knowledge resources more user-friendly and productive. In addition, this study focuses more on

problem-solving strategies than other API documentation studies, which emphasize the interface and information, in order to determine what kinds of tools must be provided to newcomers for them to learn new APIs more effectively and efficiently. Lastly, the study contributes with two guidelines for both theoretical and practical API problem solving strategies that learners can use to seek inspiration during their learning process.

7.2 Limitations

There are a number of factors, including sample size, subjectivity, sampling bias, and contextual factors that may influence the results and therefore the validity of this study. A fundamental limitation is that the study is based on an intrinsic case, which shapes the findings in accordance with the organizational characteristics and values of DHIS2. Also, the number of participants and iterations of data collection restrict theoretical saturation and theory scope and depth. Constructivist grounded theory generally examines the subjective experiences of a group, making its application to larger populations or different contexts challenging. The methodology is also susceptible to bias because it is subjective and open to multiple interpretations.

Additionally, sampling bias and generalizability are of concern as the study was conducted with a mixed sample group in which some participants may not have been fully representative of the API development experience level that the study was designed to examine. Working with two contextually distinct sample groups, one from Norway and one from Malawi, may impose limitations on the study, such as language barriers when discussing ambiguous concepts and cultural differences that influence values that influence the research topic.

7.3 Validity and reliability

Examining the validity and reliability of the study would entail evaluation of the quality and credibility of the study (Golafshani, 2003; Runeson & Höst, 2009). Through this evaluation, potential flaws in the study design, data collection, or analysis can be identified. Validity is the examination of the trustworthiness of the study, and can consist of internal-, external-, and construct validity (Runeson & Höst, 2009). A study that is deemed valid should effectively exhibit the existence of a phenomenon, while a valid instrument or measure should accurately assess the construct it is intended to evaluate (Brink, 1993). On the other hand, reliability is the consistency of the reproduction of the results, and needs to be evaluated in conjunction (Golafshani, 2003).

Internal validity

The concept of internal validity pertains to the examination of causal relationships within a given study (Runeson & Höst, 2009); whether the research is a true reflection of reality instead of consequences of extraneous factors (Brink, 1993). The potential for internal validity to be compromised in this study exists due to various factors, including selection bias and confounding variables. Demographic variables and cultural background are examples from this study of confounding variables that affect the sampling bias. The utilization of random sampling techniques is a promising approach to strengthening internal validity. This method has the potential to effectively reduce the influence of selection bias, thereby enhancing the credibility and accuracy of research findings.

The study's internal validity is strengthened by the inclusion of sample groups with diverse cultural backgrounds, including both Norwegian and Malawian developers, as well as individuals with varying demographic values such as differing ages and levels of expertise. Additionally, triangulation with integrative theoretical sampling was performed in this study, which is suggested to improve the validity and reliability of qualitative studies (Brink, 1993).

External validity

External validity refers to the extent to which the findings can be generalized and are of relevance for other cases (Runeson & Höst, 2009); whether the representations of the study are applicable across groups (Brink, 1993). The study is susceptible to external validity threats due to factors such as sample size and the research form. Specifically, a small sample size may not accurately reflect the characteristics of the larger population, and interpretive qualitative studies provide subjective meanings that are challenging to generalize. These limitations could potentially undermine the generalizability of the study's findings to the broader population and to defining a theory.

In order to enhance the external validity of the study, it is crucial to expand the sample size and diversify it, while also gaining a large enough sample size to generalize some of the important aspects of the subjective meanings that occur in the study. This study encompasses a diverse range of participants with varying levels of expertise. However, the incorporation of supplementary entry-level developers would greatly improve its external validity and strengthen the generalizability of the findings. Additionally, to compensate for the smaller sample size, it triangulates the subjective statements in the findings through integrative literature reviews.

Also, this study places a heavy weight on the web API of the health information platform DHIS2. Transferability is essential in assessing the external validity of this study because it allows the determination of the extent to which the findings can be applied to other APIs in different contexts. By recognizing the general usability and cognitive load principles that are likely to apply to all APIs, this study provides valuable insights into the learning experiences of entry-level developers that may be applicable beyond the specific case of DHIS2. The findings of this study contribute to our understanding of the external validity of research in this field and have important implications for the design and development of APIs in a variety of domains.

Construct validity

The concept of construct validity refers to the degree to which the measures employed in a study effectively capture the intended concept and research questions of the investigator (Runeson & Höst, 2009). The assessment of construct validity in this study involves an examination of the appropriateness and relevance of the measures utilized in relation to the constructs under investigation. The evaluation, in this study, involves assessing the API learning experiences constructed through a targeted inquiry approach that prioritizes interview questions pertaining to API knowledge resources that are deemed influential to the learning experience. This study's interview questions were developed with usability principles in mind to ensure that the research questions were addressed and the investigated constructs were effectively captured. Participants were able to explain their own experiences and provide insight into the API learning experiences as a result of these questions.

Semi-structured interviews facilitate the collaborative construction of shared comprehension by affording opportunities for reciprocal inquiry and clarification from both parties. Furthermore, the utilization of a flexible conversational structure during the interviews serves to mitigate the burden on the interviewees to reciprocate inquiries to the interviewer. Additionally, the triangulation with the integrative literature reviews aid in directing the interview questions and the analysis to ensure the study captures the intended concepts and research questions. Even though the construct validity could be increased further by confirming the findings and analysis obtained with the informants (Brink, 1993), the constant theoretical comparisons aid in ensuring effective capture of the intended concepts.

Reliability

The concept of reliability concerns the degree to which the data and analysis theoretically reproduce the same outcomes by other researchers (Brink, 1993; Golafshani, 2003; Runeson & Höst, 2009). In this study, reliability could be assessed by examining whether the interview questions produced consistent results (test-retest reliability) and whether different researchers would come to the same conclusions (inter-rater reliability). The present study adopts a constructivist paradigm that places significance on the existence of diverse realities that individuals construct in a subjective manner (Golafshani, 2003). Consequently, it is unsuitable to replicate identical results based on test-retest reliability, given that the interview format is intended to investigate diverse viewpoints on identical subjects.

The study's generalizability will facilitate an assessment of whether other researchers arrive at comparable conclusions. To achieve this, the study employs a triangulation approach that integrates interviews and literature reviews. To enhance the dependability of the research, it would be advantageous to increase the sample size in order to enable more rigorous generalization of the results. Additionally, including the same informants in several iterations would also increase the reliability of the study even more.

7.4 Future research avenues

This study sheds light on learning processes and the problem-solving techniques used by entry-level developers when learning APIs, as well as the obstacles they face and guidelines that can be used to overcome them. However, additional research is necessary to fully comprehend the potential implications and applications of these findings. Some future research avenues may be to:

1. *Evaluate the efficacy of the API problem-solving guidelines:* The guidelines proposed in this study have not been evaluated in practice for effectiveness. Future research could investigate whether these guidelines can effectively enhance the problem-solving procedure and increase the effectiveness of knowledge acquisition.

2. *Examine alternative strategies for addressing diverse audiences in documentation:* This study illuminated the tension between addressing multiple audiences in a single document. Alternative, more effective and efficient methods for addressing these audiences efficiently for the documentation vendor could be explored through additional research.

3. *Investigate the differences in practical problem-solving methods:* This study identified various practical problem-solving methods, such as trial-and-error and seeking community assistance. Future research could examine the efficacy and limitations of these methods in various contexts.

4. *Assess the impact of teaching problem-solving strategies in pedagogical contexts:* This study revealed the potential advantages of teaching problem-solving strategies in pedagogical contexts. Future research could investigate the impact of such interventions by incorporating a learning module that teaches and suggests various learning strategies to assist students with practical projects.

These research avenues provide a starting point for future studies that will build on the findings of this study and contribute to the field's advancement of knowledge. Researchers can improve the efficacy of learning APIs and address the difficulties faced by entry-level developers by evaluating guidelines and exploring alternative approaches. Incorporating problem-solving techniques into pedagogical contexts can also yield valuable insights into how to better prepare students for practical projects.

Bibliography

Abdelmalak, M., & Trespalacios, J. (2013). Using a Learner-Centered Approach to Develop an Educational Technology Course. *International Journal of Teaching & Learning in Higher Education*. https://scholarworks.boisestate.edu/edtech_facpubs/96

Aitman, B. (2019). Writing effective documentation | Beth Aitman | #LeadDevBerlin. Retrieved February 16, 2022, from https://www.youtube.com/watch?v=R6zeikbTgVc

AltexSoft. (2022). API Documentation and Why it Matters. Retrieved April 4, 2023, from https://www.youtube.com/watch?v=39Tt1IkLiQQ

Anaç, M. (2022). Simplicity Design Principle. Retrieved April 30, 2023, from https://bootcamp.uxdesign.cc/simplicity-design-principle-8d11fee2dbc0

Andrade, O. D., Bean, N., & Novick, D. G. (2009). The macro-structure of use of help. *Proceedings of the 27th ACM international conference on Design of communication*, 143–150. https://doi.org/10.1145/1621995.1622022

Ardito, C., Costabile, M. F., Marsico, M. D., Lanzilotti, R., Levialdi, S., Roselli, T., & Rossano, V. (2006). An approach to usability evaluation of e-learning applications. *Universal Access in the Information Society*, *4*(3), 270–283. https://doi.org/10.1007/s10209-005-0008-6

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

Aukrust, V. G. (2011). *Learning and Cognition*. Elsevier.

Bannert, M. (2002). Managing cognitive load—recent trends in cognitive load theory. *Learning and Instruction*, *12*(1), 139–146. https://doi.org/10.1016/S0959-4752(01)00021-4

Baumann, J. F., & Graves, M. F. (2010). What Is Academic Vocabulary? *Journal of Adolescent & Adult Literacy*, *54*(1), 4–12. https://doi.org/10.1598/JAAL.54.1.1

Baxter, P., & Jack, S. (2008). Qualitative case study methodology: Study design and implementation for novice researchers. *The qualitative report*, *13*(4), 544–559.

Blayney, P., Kalyuga, S., & Sweller, J. (2015). Using cognitive load theory to tailor instruction to levels of accounting students' expertise [Publisher: JSTOR]. *Journal of Educational Technology & Society*, *18*(4), 199–210.

Brink, H. I. L. (1993). Validity and reliability in qualitative research [Number: 2]. *Curationis*, *16*, 35–38. https://doi.org/10.4102/curationis.v16i2.1396

Chen, C.-M., & Lin, Y.-J. (2016). Effects of different text display types on reading comprehension, sustained attention and cognitive load in mobile reading contexts [Publisher: Routledge _eprint: https://doi.org/10.1080/10494820.2014.891526]. *Interactive Learning Environments*, *24*(3), 553–571. https://doi.org/10.1080/10494820.2014.891526

Corbin, J. M., & Strauss, A. (1990). Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology*, *13*(1), 3–21. https://doi.org/10.1007/BF00988593

Costley, J., & Lange, C. (2018). The Moderating Effects of Group Work on the Relationship Between Motivation and Cognitive Load. *The International Review of Research in Open and Distributed Learning*, *19*(1). https://doi.org/10.19173/irrodl.v19i1.3325

Crang, M., & Cook, I. (2007). *Doing Ethnographies*. SAGE Publications Ltd. https://doi.org/10.4135/9781849208949

de Jong, T. (2010). Cognitive load theory, educational research, and instructional design: Some food for thought. *Instructional Science*, *38*(2), 105–134. https://doi.org/10.1007/s11251-009-9110-0

Delfim, F. M., Paixão, K. V. R., Cassou, D., & de Almeida Maia, M. (2016). Redocumenting APIs with crowd knowledge: A coverage analysis based on question types. *Journal of the Brazilian Computer Society*, *22*(1), 9. https://doi.org/10.1186/s13173-016-0049-0

DHIS2 Team. (2023a). About data dimensions - DHIS2 Documentation. Retrieved February 8, 2023, from https://docs.dhis2.org/en/use/user-guides/dhis-core-version-237/understanding-the-data-model/about-data-dimensions.html

DHIS2 Team. (2023b). Additional data dimensions - DHIS2 Documentation. Retrieved April 26, 2023, from https://docs.dhis2.org/en/use/user-guides/dhis-core-version-237/understanding-the-data-model/additional-data-dimensions.html

DHIS2 Team. (2023c). Chapter 4 DHIS2 Technical Architecture: 4.5 The Data Model. Retrieved April 26, 2023, from https://docs.dhis2.org/archive/en/2.30/developer/html/techarch_data_model.html

DHIS2 Team. (2023d). DHIS2 Academy: Training & Capacity Building with DHIS2 Experts. Retrieved April 4, 2023, from https://dhis2.org/academy/

DHIS2 Team. (2023e). DHIS2 Documentation. Retrieved April 6, 2023, from https://docs.dhis2.org/en/home.html

DHIS2 Team. (2023f). DHIS2 Online Academy - Level 2 Academies. Retrieved April 25, 2023, from https://dhis2.org/academy/level-2/

DHIS2 Team. (2023g). DHIS2 UI Library | DHIS2 Developer Portal. Retrieved April 25, 2023, from https://dhis2.github.io/docs/tutorials/ui-library/

Elshiekh, R., & Butgerit, L. (2017). Using Gamification to Teach Students Programming Concepts [Number: 8 Publisher: Scientific Research Publishing]. *Open Access Library Journal*, *4*(8), 1–7. https://doi.org/10.4236/oalib.1103803

Fadiran, O. A., van Biljon, J., & Schoeman, M. A. (2018). How can visualisation principles be used to support knowledge transfer in teaching and learning? *2018 Conference on Information Communications Technology and Society (ICTAS)*, 1–6. https://doi.org/10.1109/ICTAS.2018.8368739

Falkner, K., & Sheard, J. (2019). Pedagogic Approaches. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 445–480). Cambridge University Press. https://doi.org/10.1017/9781108654555.016

freeCodeCamp. (2022). What is an API and How Does it Work? APIs for Beginners. Retrieved March 30, 2023, from https://www.freecodecamp.org/news/how-apis-work/

Fuhrman, J. (2017). Cognitive Load Theory: Helping Students' Learning Systems Function More Efficiently | Franklin University. Retrieved March 15, 2023, from https://www.franklin.edu/institute/blog/cognitive-load-theory-helping-students-learning-systems-function-more-efficiently

Gao, G., Voichick, F., Ichinco, M., & Kelleher, C. (2020). Exploring Programmers' API Learning Processes: Collecting Web Resources as External Memory [ISSN: 1943-6106]. *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 1–10. https://doi.org/10.1109/VL/HCC50065.2020.9127274

Garousi, V., Rainer, A., Lauvås, P., & Arcuri, A. (2020). Software-testing education: A systematic literature mapping. *Journal of Systems and Software*, *165*, 110570. https://doi.org/10.1016/j.jss.2020.110570

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

GitHub. (2023). GitHub Actions Artifacts. Retrieved March 30, 2023, from https://ghdocs-prod.azurewebsites.net/_next/data/AuKqRyFHHhF6mVNnmGnSM/en/free-pro-team@latest/rest/actions/artifacts.json?apiVersion=2022-11-28&versionId=free-pro-team%40latest&category=actions&subcategory=artifacts

Golafshani, N. (2003). Understanding reliability and validity in qualitative research [Publisher: Canadá]. *The qualitative report*, *8*(4), 597–607.

Google. (2022). Android API reference. Retrieved February 16, 2022, from https://developer.android.com/reference

Guadagnoli, M. A., & Lee, T. D. (2004). Challenge Point: A Framework for Conceptualizing the Effects of Various Practice Conditions in Motor Learning [Publisher:

Routledge _eprint: https://doi.org/10.3200/JMBR.36.2.212-224]. *Journal of Motor Behavior*, *36*(2), 212–224. https://doi.org/10.3200/JMBR.36.2.212-224

HISP. (2023). About DHIS2. Retrieved March 30, 2023, from https://dhis2.org/about/

Hoffman, B., & Schraw, G. (2010). Conceptions of Efficiency: Applications in Learning and Problem Solving. *Educational Psychologist*, *45*(1), 1–14. https://doi.org/10.1080/00461520903213618

Hwang, G.-J., & Kuo, F.-R. (2011). An information-summarising instruction strategy for improving the web-based problem solving abilities of students [Number: 2]. *Australasian Journal of Educational Technology*, *27*(2). https://doi.org/10.14742/ajet.971

Hygraph Team. (2022). How do APIs Work? A Beginner's Guide. Retrieved March 30, 2023, from https://hygraph.com/blog/how-do-apis-work

Jeng, J. (2005). Usability Assessment of Academic Digital Libraries: Effectiveness, Efficiency, Satisfaction, and Learnability [Publisher: De Gruyter Saur Section: Libri], *55*(2-3), 96–121. https://doi.org/10.1515/LIBR.2005.96

Joyce, A. (2019). How to Measure Learnability of a User Interface. Retrieved March 29, 2023, from https://www.nngroup.com/articles/measure-learnability/

Kalyuga, S. (2007). Expertise Reversal Effect and Its Implications for Learner-Tailored Instruction. *Educational Psychology Review*, *19*(4), 509–539. https://doi.org/10.1007/s10648-007-9054-3

Kalyuga, S., Ayres, P., Chandler, P., & Sweller, J. (2003). The Expertise Reversal Effect [Publisher: Routledge _eprint: https://doi.org/10.1207/S15326985EP3801_4]. *Educational Psychologist*, *38*(1), 23–31. https://doi.org/10.1207/S15326985EP3801_4

Kapenieks, J. (2013). User-friendly e-learning Environment for Educational Action Research. *Procedia Computer Science*, *26*, 121–142. https://doi.org/10.1016/j.procs.2013.12.012

Kuljis, J., & Baldwin, L. P. (2000). Visualisation techniques for learning and teaching programming [Publisher: Fakultet elektrotehnike i računarstva Sveučilišta u Zagrebu]. *Journal of computing and information technology*, *8*(4), 285–291.

Laal, M., & Ghodsi, S. M. (2012). Benefits of collaborative learning. *Procedia - Social and Behavioral Sciences*, *31*, 486–490. https://doi.org/10.1016/j.sbspro.2011.12.091

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

Lawrence, N. (2020). UI/UX Design: Simplicity. Retrieved April 30, 2023, from https://uxplanet.org/ui-ux-design-simplicity-f1cbfbfffb4b

Lee, G., Eastman, C. M., Taunk, T., & Ho, C.-H. (2010). Usability principles and best practices for the user interface design of complex 3D architectural design and

engineering tools. *International Journal of Human-Computer Studies*, *68*(1), 90–104. https://doi.org/10.1016/j.ijhcs.2009.10.001

Maalej, W., & Robillard, M. P. (2013). Patterns of Knowledge in API Reference Documentation [Conference Name: IEEE Transactions on Software Engineering]. *IEEE Transactions on Software Engineering*, *39*(9), 1264–1282. https://doi.org/10.1109/TSE.2013.12

Malhotra, N. K. (1982). Information Load and Consumer Decision Making. *Journal of Consumer Research*, *8*(4), 419–430. https://doi.org/10.1086/208882

Marshall, B., Cardon, P., Poddar, A., & Fontenot, R. (2013). Does sample size matter in qualitative research?: A review of qualitative interviews in IS research [Publisher: Taylor & Francis]. *Journal of computer information systems*, *54*(1), 11–22.

Merriam-Webster Inc. (2023). Definition of CLARITY. Retrieved April 30, 2023, from https://www.merriam-webster.com/dictionary/clarity

Mills, J., Bonner, A., & Francis, K. (2006). The Development of Constructivist Grounded Theory [Publisher: SAGE Publications Inc]. *International Journal of Qualitative Methods*, *5*(1), 25–35. https://doi.org/10.1177/160940690600500103

Moran, K. (2020). How People Read Online: New and Old Findings. Retrieved March 15, 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

Mvungi, J., & Tossy, T. (2015). Usability Evaluation Methods and Principles for the Web. *13*(7).

Myers, M. D. (1997). Qualitative Research in Information Systems. *MIS Quarterly*, *21*(2), 241–242. https://misq.umn.edu/supplements/.

Nelson, A., & Eliasz, K. L. (2023). Desirable Difficulty: Theory and application of intentionally challenging learning. *Medical Education*, *57*(2), 123–130. https://doi.org/10.1111/medu.14916

Net Age. (2010). Synonyms In Google Search. Retrieved March 23, 2023, from https://www.netage.co.za/articles/synonyms-in-google-search

Nielsen, J. (2012). Usability 101: Introduction to Usability. Retrieved March 30, 2023, from https://www.nngroup.com/articles/usability-101-introduction-to-usability/

Novick, D. G., Andrade, O. D., & Bean, N. (2009). The micro-structure of use of help. *Proceedings of the 27th ACM international conference on Design of communication*, 97–104. https://doi.org/10.1145/1621995.1622014

Parnin, C., & Treude, C. (2011). Measuring API documentation on the web. *Proceedings of the 2nd International Workshop on Web 2.0 for Software Engineering*, 25–30. https://doi.org/10.1145/1984701.1984706

Pedaste, M., Mäeots, M., Siiman, L. A., de Jong, T., van Riesen, S. A. N., Kamp, E. T., Manoli, C. C., Zacharia, Z. C., & Tsourlidaki, E. (2015). Phases of inquiry-based learning: Definitions and the inquiry cycle. *Educational Research Review*, *14*, 47–61. https://doi.org/10.1016/j.edurev.2015.02.003

Pernice, K. (2019a). The Layer-Cake Pattern of Scanning Content on the Web. Retrieved April 4, 2023, from https://www.nngroup.com/articles/layer-cake-pattern-scanning/

Pernice, K. (2019b). Text Scanning Patterns: Eyetracking Evidence. Retrieved March 15, 2023, from https://www.nngroup.com/articles/text-scanning-patterns-eyetracking/

Piccioni, M., Furia, C. A., & Meyer, B. (2013). An Empirical Study of API Usability [ISSN: 1949-3789]. *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 5–14. https://doi.org/10.1109/ESEM.2013.14

Preece, J., Sharp, H., & Rogers, Y. (2015). *Interaction design: Beyond human-computer interaction*. John Wiley & Sons.

Purchase, H. C., & Worrill, J. (2002). An empirical study of on-line help design: Features and principles. *International Journal of Human-Computer Studies*, *56*(5), 539–567. https://doi.org/10.1006/ijhc.2002.1009

Robillard, M. P. (2009). What Makes APIs Hard to Learn? Answers from Developers [Conference Name: IEEE Software]. *IEEE Software*, *26*(6), 27–34. https://doi.org/10.1109/MS.2009.193

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. (2011). A field study of API learning obstacles. *Empirical Software Engineering*, *16*(6), 703–732. https://doi.org/10.1007/s10664-010-9150-8

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164. https://doi.org/10.1007/s10664-008-9102-8

Smith, B. L., & MacGregor, J. T. (1992). What is collaborative learning [Google-Books-ID: 0gFnVPlcwqMC]. In G. A. Olson & S. I. Dobrin (Eds.), *Composition Theory for the Postmodern Classroom*. SUNY Press.

Snodgrass, E., & Winnie, S. (2019). API practices and paradigms: Exploring the protocological parameters of APIs as key facilitators of sociotechnical forms of exchange [Publisher: University of Illinois]. *First Monday*, *24*(2).

Soloway, E., Guzdial, M., & Hay, K. E. (1994). Learner-centered design [Place: New York, NY : Publisher: Association for Computing Machinery,]. *Interactions.*, *1*(2), 36–48.

Southern, M. G. (2022). How Does Google Search Use Synonyms? Retrieved March 23, 2023, from https://www.searchenginejournal.com/how-does-google-search-use-synonyms/433700/

Srull, T. K. (1983). The Role of Prior Knowledge in the Acquisition, Retention, and Use of New Information. *ACR North American Advances*, *NA - Advances in Consumer Research Volume 10*. Retrieved March 26, 2023, from https://www.acrwebsite.org/volumes/6183/volumes/v10/NA-10/full

Stylos, J., & Myers, B. A. (2008). The implications of method placement on API learnability. *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 105–112. https://doi.org/10.1145/1453101.1453117

Thomas Bush. (2019). 5 Examples of Excellent API Documentation (and Why We Think So) | Nordic APIs |. Retrieved May 4, 2022, from https://nordicapis.com/5-examples-of-excellent-api-documentation/

Torraco, R. J. (2016). Writing integrative literature reviews: Using the past and present to explore the future [Publisher: Sage Publications Sage CA: Los Angeles, CA]. *Human resource development review*, *15*(4), 404–428.

Tray.io. (2023). How do APIs work? An in-depth guide. Retrieved March 30, 2023, from https://tray.io/blog/how-do-apis-work

Twitter. (2023). Twitter API Documentation. Retrieved March 30, 2023, from https://developer.twitter.com/en/docs/twitter-api

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

van Gog, T., Kester, L., & Paas, F. (2011). Effects of worked examples, example-problem, and problem-example pairs on novices' learning. *Contemporary Educational Psychology*, *36*(3), 212–218. https://doi.org/10.1016/j.cedpsych.2010.10.004

Verne, G., & Bratteteig, T. (2018). Inquiry when doing research and design: Wearing two hats. *Interaction Design and Architecture(s)*, (38), 89–106. https://doi.org/10.55612/s-5002-038-005

Vike, H., & L'orange Fürst, E. (2020). Forskningsetikk og forskningens frihet: Utfordringer for antropologifaget [Publisher: Universitetsforlaget]. *Norsk antropologisk tidsskrift*, *31*(3), 165–176. https://doi.org/10.18261/issn.1504-2898-2020-03-02

Walker, D., & Myrick, F. (2006). Grounded theory: An exploration of process and procedure [Publisher: Sage Publications Sage CA: Thousand Oaks, CA]. *Qualitative health research*, *16*(4), 547–559.

Walsham, G. (2002). Interpretive Case Studies in IS Research: Nature and Method. In *Qualitative Research in Information Systems* (pp. 100–113). SAGE Publications, Ltd. https://doi.org/10.4135/9781849209687.n6

Walsham, G. (2006). Doing interpretive research. *European Journal of Information Systems*, *15*(3), 320–330. https://doi.org/10.1057/palgrave.ejis.3000589

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

Yin, H., Zheng, Y., Sun, Y., & Huang, G. (2021). An API Learning Service for Inexperienced Developers Based on API Knowledge Graph. *2021 IEEE International Conference on Web Services (ICWS)*, 251–261. https://doi.org/10.1109/ICWS53863.2021.00043

Zinovieva, I., Artemchuk, V., Iatsyshyn, A. V., Popov, O., Kovach, V., Iatsyshyn, A. V., Romanenko, Y., & Radchenko, O. (2021). The use of online coding platforms as additional distance tools in programming education [Issue: 1]. *Journal of physics: Conference series*, *1840*, 012029.

Appendices

Appendix A: Preliminary interview guide

Preliminary interview

Introduction

- How much programming experience do you have?
- Languages? How did you learn these languages?
- Do you have experience in developing/programming for DHIS2?
- How important are the learning resources for you to develop a system you are satisfied with?
- What learning resources have helped you with the development of systems?

Main segment

- What do you think is the purpose behind documentations like API documentation?
- Do you have experience with API documentation? Possibly mention some you have used.
- What are your general experiences with DHIS2 learning resources?
    - What other resources when this didn't help?
- Have you used the DHIS2 API documentation during the project time in IN5320?
    - If yes. How often?
- What was your first impression of the documentation?
- What kind of information were you looking for when you searched the documentation?
    - Did you get answers to what you needed more information on?
    - If so, why not?
- What other learning resources did you use during the project time?
    - Which of these resources taught you the most or answered what you needed help with?

Closing segment

- If you were to do the same project again, what resources would you have used the most to solve your challenges in your experience?
- If you could improve the API documentation, what would you do?

Appendix B: 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?

Appendix C: 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?