

Masteroppgave

Designing knowledge boundary resources to assist large audiences of new complementors in learning to use the platform APIs.

Leif Erik Greftegreff Haakenstad

Informatics: Programming and System Architecture

60 credits
Spring 2023

Department of Informatics
The Faculty of Mathematics and Natural Sciences



Abstract

Platform owners must ensure that complementors are able to extend their platforms. An increasingly popular approach is through the provision of web APIs. However, knowledge boundaries emerge as complementors require knowledge of how to use these APIs to develop platform applications. These knowledge boundaries are particularly pronounced in enterprise platforms, which often possess a high degree of functionality and complexity of usage. Complementors encounter significant complexity when using these APIs. Consequently, platform owners must provide resources that support complementors and address these knowledge boundaries, referred to as knowledge boundary resources (KBRs). Previous research has classified approaches for provisioning KBRs in platform ecosystems and general strategies for supporting API learnability; however, we lack knowledge on how to design resources that efficiently supports larger groups of complementors in learning APIs of enterprise platforms.

In response to this lack of knowledge, this thesis addresses the following research question: *“How can enterprise platform owners design knowledge boundary resources to support large audiences of new complementors in learning to use the platform APIs?”* Based on design science research and over a 1.5-year research period, artifacts were designed to support 120 new complementors of an enterprise platform in learning to use its APIs. The study explores how these designed artifacts supported the complementors learning. Based on the analysis of empirical findings, this thesis identifies four design considerations that can serve as valuable guidance for platform owners when designing KBRs to support new complementors learning the platform’s API: (1) Offering a simple testing tool, (2) Facilitate exporting to code, (3) Provide interactive tutorials, and (4) Support esoteric terminology comprehension. Furthermore, the findings contribute by offering platform owners examples of how the considerations can be designed.

This research contributes to existing research on enterprise platforms by extending knowledge on what earlier literature has conceptualized as broadcasting KBRs. Specifically, the thesis extends this knowledge by designing broadcasting KBRs and exploring how these address knowledge boundaries.

Acknowledgments

First and foremost, I would like to extend my gratitude to my supervisor Petter Nielsen, and co-supervisor Magnus Li. Their continuous guidance, support, and insightful feedback have been invaluable throughout this research project.

I would like to express my appreciation to all the participants in this research project. Your willingness to express experiences and perspectives has enriched the quality of this work

To my fellow students, particularly those from room HB229C, thank you for the great discussions, the shared laughter, and the competitive table tennis matches. You've enriched my academic journey and made me a better table tennis player.

Finally, I would like to thank my friends and family for their love and support.

Leif Erik

University of Oslo

May 2023

Table of Contents

Abstract	2
Acknowledgments.....	1
1 Introduction	1
1.1 Research question.....	3
1.2 Chapter summary	5
2 Related literature.....	7
2.1 Platform literature	7
2.1.1 Platform ecosystems	7
2.1.2 Enterprise platform ecosystems	8
2.1.3 Boundary resources.....	9
2.1.4 Knowledge boundaries.....	11
2.1.5 Knowledge boundaries resources	13
2.2 API learnability	16
2.2.1 Programmers experience learning APIs.....	16
2.2.2 Designing APIs with learnability in mind	16
2.2.3 API documentation	17
2.3 Chapter summary	19
3 Research Approach.....	20
3.1 Case description	20
3.1.1 HISP and DHIS2.....	20
3.1.2 DHIS2 as platform	21
3.1.3 DHIS2 Design lab	21
3.1.4 Development in Platform Ecosystems	22
3.1.5 DHIS2 Design lab: developing and evaluating learning resources	23
3.2 Research methodology	24
3.2.1 Design science research	24
3.3 Data collection.....	29
3.3.1 Diagnosis stage	29
3.3.2 Design and demonstration stage	30
3.3.3 Evaluation stage	30
3.3.4 Summary of data collection	31
3.4 Data analysis	32
3.4.1 Preliminary analysis.....	32
3.4.2 Design analysis	33

3.4.3	Evaluation analysis	34
3.4.4	Constructing contribution.	36
3.5	Ethical considerations	36
3.6	Chapter Summary.....	37
4	Findings from Diagnosis	38
4.1.1	Complex data model	38
4.1.2	Complicated terminology.....	39
4.1.3	Limited opportunities for learning by doing	40
4.1.4	No query-to-code translation	42
4.2	Chapter Summary.....	43
5	Findings from Design and Demonstration.....	44
5.1.1	API testing tool	45
5.1.2	Introduction tutorial to the DHIS2 API	51
5.2	Chapter Summary.....	54
6	Findings from Evaluation	55
6.1	API testing tool.....	55
6.1.1	Diagnosed challenge: Limited opportunities for learning by doing	55
6.1.2	Diagnosed challenge: No query-to-code translation.....	56
6.2	Introduction tutorial to the DHIS2 API.....	58
6.2.1	Diagnosed challenges: Complicated terminology and complex data model	58
6.3	Chapter Summary.....	61
7	Design considerations.....	62
7.1	Offering a simple testing tool.....	63
7.2	Facilitate exporting to code	66
7.3	Provide interactive tutorials	67
7.4	Support esoteric terminology comprehension.....	69
8	Discussion and Contributions	70
8.1	Contribution to research	72
8.1.1	Broadcasting API KBRs	73
8.1.2	Hosted developer sandbox	75
8.1.3	Platform and Resource level design	78
8.2	Contributions to practice	82
8.3	Limitations	83
9	Conclusion.....	84
10	References	85

List of Tables

Table 2.1: Boundary resource model (Ghazawneh & Henfridsson, 2013, p. 4).....	10
Table 2.2: Addressing knowledge boundaries (Foerderer et al., 2019, p. 130).....	14
Table 2.3: Summary of terms and theories	19
Table 3.1: Summary of data collection	31
Table 3.2: Challenges identified in “Data Query Playground”.....	33
Table 4.1: Summary of challenges identified in the diagnosis	43
Table 5.1: Summary of artifacts responding to challenges.....	44
Table 7.1: Design considerations	62
Table 8.1: Summary of contributions	71

List of Figures

Figure 2.1: Platform ecosystem, inspired by Tiwana’s figure (2014, p. 6)	8
Figure 2.2: Model of boundary resource design Ghazawneh and Henfridsson’s (2013, p. 4) ..	9
Figure 2.3: Structures of knowledge boundaries (Foerderer et al. (2019, p.123)	11
Figure 3.1: The DHIS2 app hub shows some of the available applications.	21
Figure 3.2: Timeline of the course	22
Figure 3.3: The DHIS2 app course front page	23
Figure 3.4: Model for conducting DSRM (Peffer et al., 2007, p. 46)	25
Figure 3.5: Timeline of DSRM activities during the research period.....	25
Figure 3.6: Part of the table showing recognized challenges.....	32
Figure 3.7: Initial coding of the data collected	34
Figure 3.8: Part of the table showing categories affecting API learning.	35
Figure 3.9: Considerations evolved from an early iteration of the analysis	36
Figure 4.1: Representation of the data model	38
Figure 4.2: Description in the DHIS2 app course of how to convert a query.....	42
Figure 5.1: DHIS2 app course frontpage, including the API testing tool.....	45
Figure 5.2: The API testing tool page	46
Figure 5.3: API testing tool showing available parameters.	46
Figure 5.4: Option in API testing tool to show optional parameters.	47
Figure 5.5: Example of an API error in the API testing tool	47
Figure 5.6: Input field in the API testing tool with syntax highlighting and syntax checks....	48
Figure 5.7: API testing tool showing the response from the DHIS2 API.....	49
Figure 5.8: API request converted to a code snippet	50
Figure 5.9: DHIS2 app course front page, including the DHIS2 module.....	51
Figure 5.10: The data model is mapped to a real-life scenario	52
Figure 5.11: A part of the introduction to the DHIS2 API	53
Figure 6.1: Illustration in tutorial with no relatable example	59
Figure 6.2: Illustration in tutorial with a relatable example.....	60
Figure 7.1: Example of a simple testing tool as designed in this research project	63
Figure 7.2: An API request where response is not formatted.	64
Figure 7.3: An API request where the response is well formatted	65
Figure 7.4: Example of a code example produced using the export-to-code function	66
Figure 7.5: A part of the interactive tutorial as designed in this research project	67
Figure 8.1: Design levels for supporting complementors	79

1 Introduction

Enterprise systems (ES) are complex package-based applications designed to meet the many needs of organizations (Strong & Volkoff, 2010). Using these packages and adapting them to the organizations' needs is an increasingly popular approach in software acquisition (Pollock et al., 2003; Strong & Volkoff, 2010). Vendors of ES may want to open its technology, allowing outsiders to access information and build applications for the system (Foerderer et al., 2019; Tiwana, 2014). When opening the technology, a “platform ecosystem” emerges consisting of a *platform* providing core functionality, *complementary apps* extending the core functionality, and *interfaces* mediating between the platform core and its complementary apps (C. Y. Baldwin & Woodard, 2008; Tiwana, 2014). Platform ecosystems challenge the traditional approach of developing digital products within the firm by allowing a massive network of outside firms with unique expertise to access the platform and build complementary app (Tiwana et al., 2010). This strategy is shown to outperform traditional development in various industries, including the mobile operating system market, where iOS gained massive success by opening up for third-party development (Kauschinger et al., 2021). Many top-ranked companies by market capitalization, such as Apple and Microsoft, use the platform strategy in their business (Cusumano et al., 2020)

Platform owners face a significant challenge in drawing enough users and complementors to benefit from two-sided network effects, which cause the platform's value to increase as more products and services are added and more users join (Cusumano, 2010; Tiwana, 2014). Overcoming this challenge involves adapting the platform strategy to facilitate the onboarding of complementors that can develop complementary apps, adding value to the platform (Foerderer et al., 2019; Tiwana, 2014). However, this is not a straightforward task. The platform owner must shift the focus from in-house development to providing adequate resources for complementors to be capable of extending the functionality (Bergvall-Kåreborn & Howcroft, 2014; Foerderer et al., 2019).

Platform owners must provide *boundary resources* that serve as the interface between the platform owner and external parties, allowing complementors to engage with the platform's core functionality (Ghazawneh & Henfridsson, 2013). These boundary resources include Application Programming Interfaces (API), allowing complementors to utilize the platform's functionality by simply referencing the API (commonly referred to as "calling an API"). Platform owners providing APIs enable complementors to use the platform's capabilities without being familiar with the implementation details inside the platform (Dekel & Herbsleb, 2009; Tiwana, 2014). The complementors benefit from valuable ignorance by simply providing the required parameters when making API calls without needing to understand how the API processes their request internally (Tiwana, 2014).

The utilization and implementation of web-based APIs, commonly referred to as "web APIs," has witnessed considerable growth in recent years (Gat et al., 2013; Tan et al., 2016). These APIs are designed to be accessed through the internet utilizing standard web protocols such as HTTP. Large enterprise platforms, such as SAP, heavily rely on web APIs, with a substantial part of their functionalities dependent on these interfaces (*REST / APIs / SAP Business Accelerator Hub*, n.d.). However, for efficiently interoperating with the platform interfaces and extending its functionality, it's crucial to onboard complementors with how to use these APIs (Danielsen & Jeffrey, 2013). Developers in all ranges of expertise, from beginners to experts, spend significant time learning to use new APIs (B. A. Myers & Stylos, 2016). Enterprise APIs carry additional challenges as these systems are complex, covering many functionalities (Foerderer et al., 2019). Jeong et al. found that participants had limited success finding relevant API elements in an enterprise API (Jeong et al., 2009).

Enterprise software vendors face the challenge of furnishing third parties with development knowledge to facilitate participation in the development and innovation for the platform. In this context, platform strategies inherently impose "knowledge boundaries" between platform owners and complementors (Foerderer et al., 2019). Specifically, these boundaries emerge between platform owner and complementors. The complementors must understand how to access, combine and extend platform functionality to develop add-on products and services. Foerderer et al. (2019) emphasize the need to address these knowledge boundaries, stating: "If platform owners do not identify and address these knowledge boundaries effectively, the success of the platform strategy is likely to be endangered." (p. 120). As a response, the platform owner needs to provide various resources to address these knowledge boundaries, which Foerderer et al. (2019) refer to as *knowledge boundary resources* (KBR). Among some

common examples of KBRs, we find technical documentation and sample code, which have received high attention in the research field of conveying API knowledge (e.g., Bianco et al., 2014; Jeong et al., 2009; Robillard, 2009).

However, there is a lack of research identifying the effectiveness of particular types of KBRs in addressing knowledge boundaries (Foerderer et al., 2019), including how these support the learning of APIs. A more recent study about requirements of good-quality API documentation finds “*low research attention on documentation artifacts that present high value to developers which are also generally missing from vendor documentation*” (Cummaudo et al., 2022, p. 13), calling for more research on documentation artifacts (to complementors). This includes how to produce them, how they should be communicated, and the most efficient means for developers to consume such information.

To address this lack of knowledge, the thesis aims to explore how to design KBRs for supporting new complementors in learning the APIs of an enterprise platform.

1.1 Research question

The thesis will extend the existing literature on designing of KBRs for enterprise platforms by addressing the following research question:

How can enterprise platform owners design knowledge boundary resources to support large audiences of new complementors in learning to use the platform APIs?

To answer this research question, the thesis reports from a 1.5-year design science research project. The project involved studying new complementors learning to develop applications for DHIS2, a globally adopted, open-source, web-based enterprise health management information platform. The focus of this study was specifically on how these complementors learned to use the platform’s API. To study this, the research project took part in the “Development in Platform Ecosystems” course at the University of Oslo. During this course, 120 students were introduced to developing applications for DHIS2, including learning its APIs. Utilizing this course, the research project aimed to understand how to support these new complementors in learning the APIs by focusing on the following objectives:

- a) Identifying key challenges students have with understanding and using the DHIS2 API

- b) Design an artifact targeting the current challenges and examine how this artifact influences students' understanding.
- c) From designing and evaluating this artifact, identify considerations for guiding platform owners in designing knowledge boundary resources supporting complementors in learning APIs.

By studying the widely used DHIS2 platform and its associated course, this thesis provides valuable insight into the design of knowledge boundary resource that supports the learning process for a large audience of new complementors. From these experiences, the thesis offers four design considerations that can serve as valuable guidance for guide platform owners in designing KBRs supporting complementors in learning the platform APIs: (1) Offering a simple testing tool, (2) facilitate export to code, (3) Support esoteric terminology comprehension and (4) Provide interactive tutorials. The thesis does also contribute to existing research on knowledge boundary resources, including (1) an extended knowledge of KBRs directed towards API learning, exploring how various types of KBRs address the knowledge boundaries of new complementors, (2) delving into a specific type of KBR, providing a description of such types of artifacts, how it supports complementors and design considerations for implementation, and (3) highlighting different levels of design in platforms for addressing knowledge which contributes to an understanding of how knowledge boundaries can be addressed in various ways. Finally, the thesis identifies further avenues for research in the field of knowledge boundary resources and API learning.

1.2 Chapter summary

This thesis is structured as follows:

Chapter 2: Related literature

This chapter describes platform literature which serves as a foundation for understanding the nature of these ecosystems, the challenges complementors face when utilizing them, and how platform owners can address these challenges. It also describes the existing literature on strategies for supporting complementors in learning APIs.

Chapter 3: Research Approach

This chapter describes the 1.5-year research project. It begins with a case description to give the context and background of the research. Then it delves into the research methodology used for the research. Following, it details how data was collected, and finally, it discusses how the data was analyzed.

Chapter 4: Findings from Diagnosis

This chapter describes the findings from diagnosing existing challenges students had when learning to use the DHIS2 APIs. Specifically, it details four key challenges that emerged from this diagnosis.

Chapter 5: Findings from Design and Demonstration

In this chapter, the focus shifts to designing and demonstrating two artifacts designed to address the challenges identified in the previous diagnosis chapter. These artifacts include (1) An API testing tool and (2) an introductory tutorial to the DHIS2 API

Chapter 6: Findings from Evaluation

This chapter describes how the artifacts designed in the design and demonstration chapter address the diagnosed challenges.

Chapter 7: Design considerations

This chapter describes four design considerations derived from the diagnosis, design and demonstration, and evaluation. These considerations are presented with a detailed discussion about their importance.

Chapter 8: Discussion and Contributions

This chapter describes how the findings of this thesis contribute to research about KBRs and their role in addressing the knowledge boundaries, especially for new complementors. The chapter presents three contributions to research and a set of practical contributions, extending current knowledge of how platform owners can support large audiences of new complementors learning to use the platform APIs.

Chapter 9: Conclusion

This chapter summarizes the thesis and its key findings.

2 Related literature

The thesis seeks to understand how enterprise platform owners can design resources for supporting large audiences of new complementors learning the platform's API. To understand how platform owners can support its complementors, a review of relevant literature was conducted. The first section of this chapter delves into the platform literature, which serves as a foundation for understanding the nature of these ecosystems, the challenges complementors face when utilizing them, and how platform owners can address these challenges. In the last section, existing literature on strategies for supporting developers in learning APIs is explored, as this can also be relevant in a platform context of supporting complementors learning the platform APIs.

2.1 Platform literature

2.1.1 Platform ecosystems

A platform-based ecosystem consists of two parts, the platform itself and its complementary applications, which interoperate through interfaces (Tiwana, 2014). "Most platform definitions focus on the reuse or sharing of common elements across complex products or systems of production" (Baldwin & Woodard, 2008, p. 6). The software *platform* acts as a foundation that serves core functionality, allowing complementary applications to use and extend this functionality to their own needs. This core exhibits a low variety and high reusability (Tiwana et al., 2010). By "opening" the software architecture, the vendor opens for third-party complementors to develop contributing content for the platform (Gizaw et al., 2017; Tiwana, 2014).

Interfaces acts as the intermediary in platform ecosystems, facilitating the interactions between the core functionality and the complementary applications. They are the platform's visible information which complementors can interact with. These interfaces include resources such as APIs and protocols specifying how apps and the platform interact and exchange information. They establish a set of rules that ensure apps can efficiently interact with the platform (Tiwana, 2014). The interfaces must remain relatively stable and unchanging to the components that depend on them (C. Baldwin & Woodard, 2008; Tiwana, 2014). Changes to the interfaces may break applications that depend on them.

Applications (sometimes referred to as “app”) is an add-on software that connects to the platform through interfaces and extends the platform’s functionality (C. Baldwin & Woodard, 2008; Tiwana, 2014). These apps exhibit a high variety and low reusability within the platform ecosystem (C. Baldwin & Woodard, 2008). The collection of the platforms, apps, and interfaces constitutes a *platform ecosystem*, as illustrated below in Figure 2.1 (Tiwana, 2014). End-users represent the diverse group of current and potential adopters of the platform using its various apps (Tiwana, 2014).

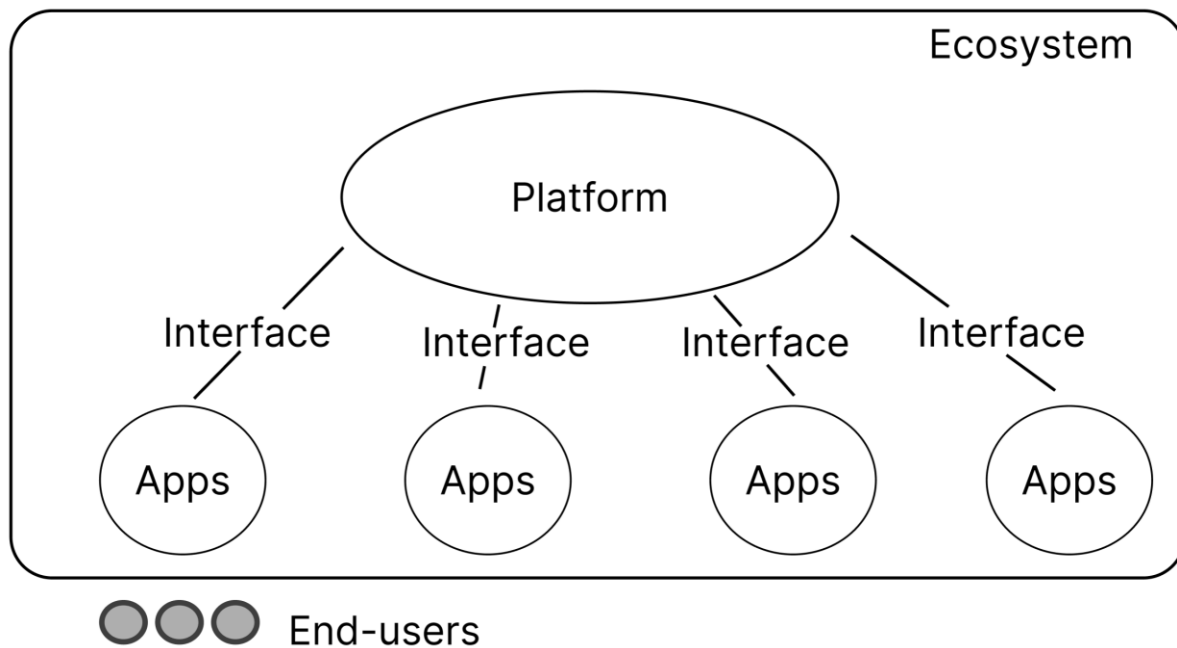


Figure 2.1: Platform ecosystem, inspired by Tiwana’s figure (2014, p. 6)

2.1.2 Enterprise platform ecosystems

Developing apps for enterprise software platforms presents a significant challenge. Enterprise software platforms are complex systems encompassing a wide set of functionalities, architectures, and business lines which makes it anything but trivial for complementors to develop apps for the platform (Foerderer et al., 2019). Hence, a challenge for enterprise platform owners is to furnish third parties with knowledge on developing apps for such complex systems (Foerderer et al., 2019).

2.1.3 Boundary resources

The platform owner must provision resources enabling complementors to build complementary applications. These resources are referred to as *boundary resources*, serving as an interface between the platform owner and complementors developing applications, enabling complementors to engage with and extend the platform functionality (Bianco et al., 2014; Ghazawneh & Henfridsson, 2013). These boundary resources include software tools and regulations, serving as the arm's-length relationship between the platform owner and the complementors. In software platforms, these resources typically include SDKs and a multitude of related APIs (Ghazawneh & Henfridsson, 2013).

Ghazawneh and Henfridsson (2013) propose a model of boundary resource design in third-party development. This model is shown in Figure 2.2, followed by Table 2.1, describing the constructs of this model. The model serves as a theoretical framework explaining how platform owners and third-party interact. As shown in Figure 2.2, the platform owner designs boundary resources (like APIs). Third-party developers use these to create applications. The model provides an intellectual structure assisting in understanding the role of platform boundary resource design and usage for third-party developers.

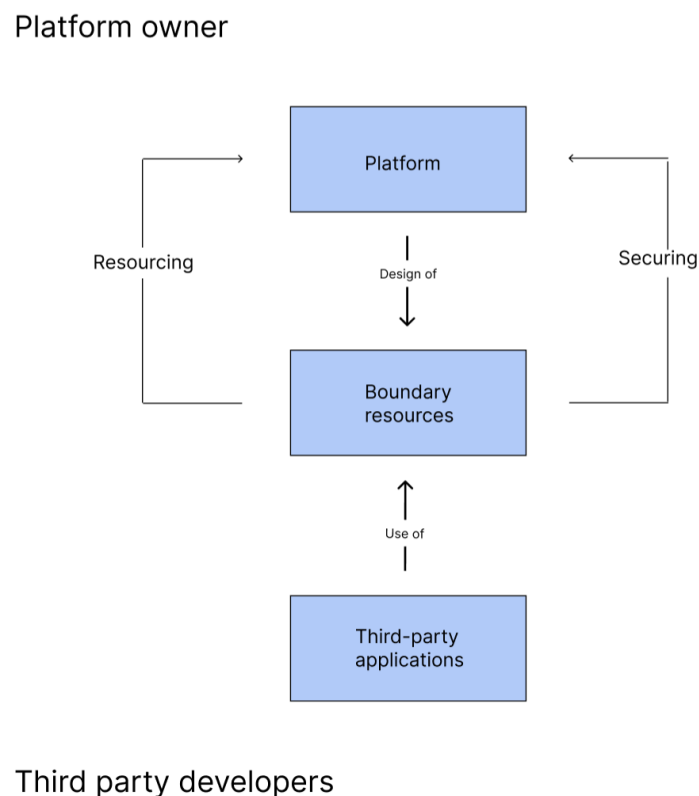


Figure 2.2: Model of boundary resource design Ghazawneh and Henfridsson's (2013, p. 4)

Construct	Description
Platform	“The extensible codebase of a softwarebased system that provides core functionality shared by the modules that interoperate with it and the interfaces through which they operate” (Tiwana et al. 2010, p. 676)
Boundary Resources	The software tools and regulations that serve as the interface for the arm'slength relationship between the platform owner and the application developer
Third-Party Applications	Executable pieces of software that are offered as applications, services, or systems to end-users of the platform
Boundary Resources Design	The platform owner’s act of developing new, or modified, boundary resources as a response to perceived external contribution opportunities and control concerns
Boundary Resources Use	The third-party developer’s act of developing end-user applications drawing on boundary resources offered by the software platform owner
Resourcing	The process by which the scope and diversity of a platform is enhanced
Securing	The process by which the control of a platform and its related services is increased

Table 2.1: Boundary resource model (Ghazawneh & Henfridsson, 2013, p. 4)

APIs are a popular type of boundary resource (Bianco et al., 2014; Tiwana, 2014, Ghazawneh, 2012). The API serves as an abstraction layer that hides complexity, internal implementations, and logic for the caller. API consumers gain the benefit of *valuable ignorance* by simply providing the required parameters when making API calls without needing to understand how the API process their request internally (Tiwana, 2014). Platform owners offer APIs to enable complementors to leverage the platform’s capabilities without being familiar with the implementation details inside the platform (Dekel & Herbsleb, 2009; Tiwana, 2014).

Various types of APIs exist, but when this thesis uses the term “API,” it refers to web APIs. Web APIs are a widely-used subset of APIs that can be accessed through web services, such as HTTP (Tan et al., 2016). Web API commonly uses REST as a communication protocol and JSON as a content format (Tan et al., 2016). While APIs are a popular approach for exposing a platform’s functionality, developers encounter several difficulties interacting with such APIs (Sohan et al., 2017).

2.1.4 Knowledge boundaries

Complementors need sufficient knowledge in accessing and extending the platform's boundary resources, such as its APIs, for developing contributing applications. This imposes “knowledge boundaries” between the platform owner and complementors. The integration of knowledge across organization boundaries is a critical issue in knowledge management for distinguishing the platform from its competitors and impacting organizational outcomes (Foerderer et al., 2019). Both platform owners and complementors aim to integrate knowledge to maximize product development, a subject addressed by Carlile’s (2004) framework, which examines knowledge boundaries between two different groups engaged in new product development. This thesis uses Foerderer et al. (2019)’s adoption of Carlile’s framework as it offers a structured understanding of the causes of differences in knowledge boundaries. Figure 2.3 illustrates this framework.

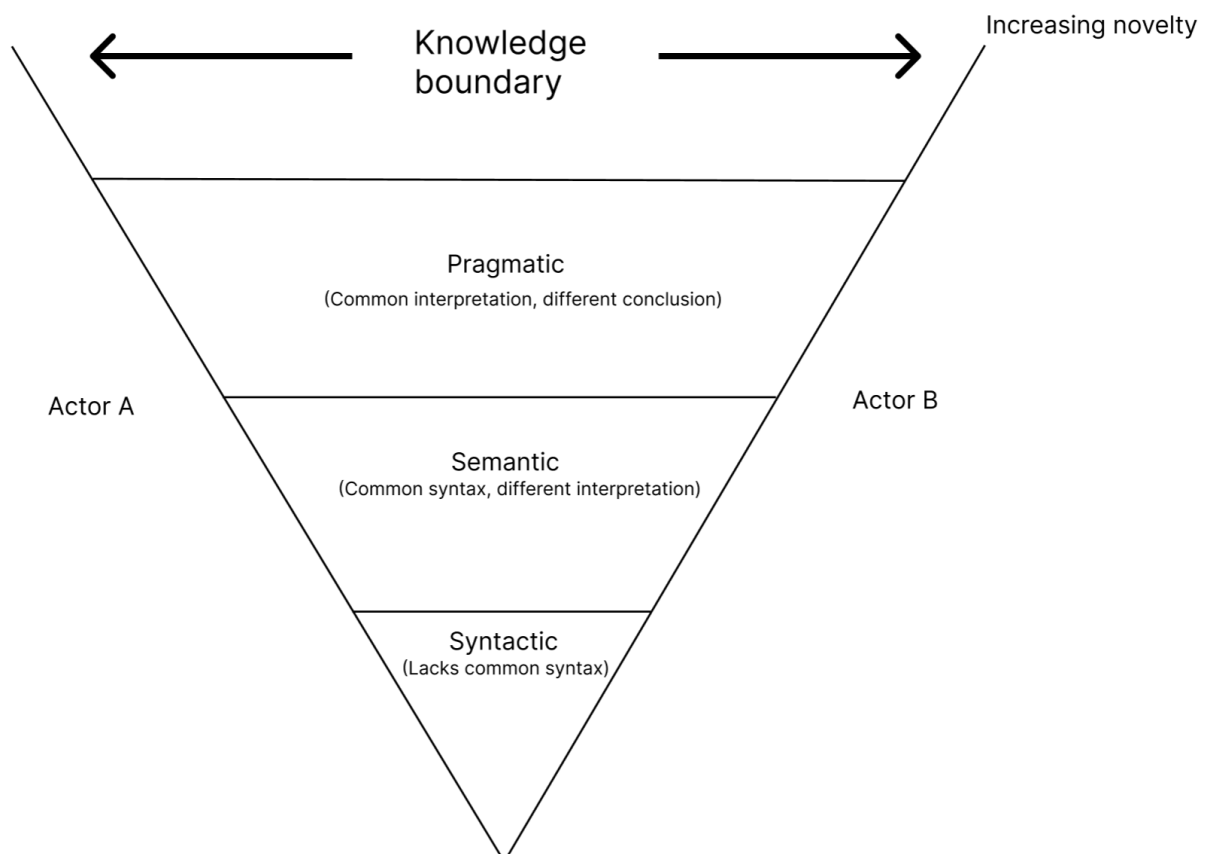


Figure 2.3: Structures of knowledge boundaries (Foerderer et al. (2019, p.123))

As novelty arises, this widens the knowledge boundary. The framework separates the gaps in knowledge into three categories:

Syntactic boundaries refer to actors lacking a shared vocabulary and grammar integrating knowledge. For example, one actor may have a specialized language that another actor is unfamiliar with, leading to difficulties in communication and collaboration. For example, the platform API may include terms unfamiliar to complementors.

Semantic boundaries refer to actors who share a common syntax but interpret what is being communicated differently. The information being communicated is usually context-specific. For example, complementors with different backgrounds may have a different interpretation of syntax used in the API, such as “data points.”

Pragmatic boundaries refer to actors sharing a common syntax and interpretation of what’s being communicated but drawing different understandings or conclusions. For example, pragmatic boundaries can arise when complementors understand the same API error message yet draw different conclusions on how to solve this.

As suggested by several, failing to address the categories for managing knowledge gaps may hinder successful development (Carlile, 2004; Foerderer et al., 2019). This framework can help structure approaches for overcoming knowledge boundaries, guiding platform owners and complementors in integrating knowledge, and maximizing product development. Foerderer et al. (2019) identified various types of resources referred to as *knowledge boundary resources* that platform owners can provide in response to these knowledge boundaries. The following section will delve more into these types of resources.

2.1.5 Knowledge boundaries resources

The type of knowledge boundary (syntactic, semantic, or pragmatic) affects the way objects and activities need to be designed to overcome these boundaries and achieve successful product development and achieve innovation (Carlile, 2004). Foerderer et al. (2019) refer to “knowledge boundary resources” (KBR) as an approach to overcoming knowledge boundaries. This includes “*objects and activities employed by platform owners in order to overcome knowledge boundaries and enable effective product development outcomes as KBR, both in relation to the work on platform governance from a boundary perspective as well as regarding the notion of boundary objects and boundary spanners (ie, human resources that enable boundary spanning) in the sociological literature*” (Foerderer et al., 2019, p. 125).

Foerderer et al. (2019) found that KBRs can be categorized based on the scale & scope, resulting in three distinct categories: *broadcasting*, *brokering*, and *bridging*. Table 2.2 summarizes the different types of categories, their scale and scope, and empirical examples of each category. This table can guide platform owners by distinguishing different types of knowledge boundary resources and to which scale and scope they target complementors. Provisioning these resources can be considered a trade-off in selecting the right amount of scale and scope for supporting complementors, as it is necessary to provision KBRs at the right scope while having the resources scalable for a larger audience.

	Broadcasting	Brokering	Bridging
Description	Standardized, based on resources that provide knowledge via transferable objects or through storage in centralized databases. Accessible by complementors without having to interact with the platform owner.	Intermediate type of knowledge boundary resources that provides meta-knowledge in terms of knowing where complementors may obtain help. Implemented via dedicated, semi-formalized interaction. Brokering mediates between platform owner and complementors.	Individualized mode of knowledge boundary resources based on intensive, frequent, and sometimes informal exchanges between platform owners and developers. Organizational members of the platform provider form a direct link with organizational members of complementors.
Scale	Entire ecosystem of complementors	Subset of complementors	Individual complementor
Scope	Factual, generally applicable knowledge, not necessarily problem specific	Meta-knowledge, gives direction on where/how to obtain problem solution	Specific problem-solving capabilities
Empirical examples	<ul style="list-style-type: none"> ● technical documentation ● information portals ● handbooks ● sample code ● Modelling guidelines ● massive online courses ● communities of practice ● hosted developer sandbox 	<ul style="list-style-type: none"> ● helpdesks ● account manager 	<ul style="list-style-type: none"> ● one-to-one assistance ● technological coaching ● co-innovation activities ● alignment workshops between third-party developers and platform owners

Table 2.2: Addressing knowledge boundaries (Foerderer et al., 2019, p. 130)

In an elaboration of the presented table, the broadcasting category is a particular type of KBR characterized by being applicable for a broad audience of complementors, highly standardized, and having large scalability. Broadcasting KBRs offers a standardized and organized pool of knowledge that complementors can access without interacting with the platform owner. It has limitations of scope due to not being able to address problem-specific areas and is commonly used in generally applicable knowledge. Broadcasting KBRs resources are a known way to address developer requirements that are anticipated, recurring, and well-understood. This is done by providing complementors with knowledge in the form of predefined and objective facts, which can be accessed through transferable objects. Syntactic boundaries between platform owners and complementors can efficiently be addressed through provisioning broadcasting KBRs (Foerderer et al., 2019).

Brokering KBRs involves dedicated resources such as help desk or account managers providing technical information through personal interactions and mediating between platform

owner and complementors. It is strongly formalized yet has limitations on scale. Brokering includes boundary objects and boundary-spanning activities by having personal interactions.

Bridging involves ingoing and frequent interactions between platform owner and complementors. It relies much on boundary-spanning activities, i.e., personal interactions. Due to personal linkages, this limits the scale of individual exchanges. Examples of bridging activities are one-to-one assistance or technological coaching. Semantic and pragmatic knowledge boundaries are addressed through brokering and bridging KBRs (Foerderer et al., 2019).

Addressing knowledge boundaries is crucial for the success of a platform strategy (Foerderer et al., 2019; Ghazawneh & Henfridsson, 2013). Changes to platform design can create knowledge boundaries for its complementors, and by failing to account for these in KBR, the platform owner risks losing innovation in the platform (Foerderer et al., 2019). Changes to the technical characteristics of the platform, such as its interfaces, cause different types of knowledge boundaries. Providing appropriate KBR by scale and scope based on the knowledge boundaries remains an important aspect of managing knowledge in platform development (Foerderer et al., 2019). This contributes to the simplicity of usage from complementors and remains a desired property of platform architectures (Tiwana, 2014)

2.2 API learnability

API learnability refers to the ease with which developers can understand, learn and use an API to build applications (Meng et al., 2018; Tello-Rodríguez et al., 2020). Prior studies on API learnability can be categorized into three broad streams: programmers' experiences when learning APIs, designing APIs with learnability in mind, and API documentation.

2.2.1 Programmers experience learning APIs

Prior research has found various challenges developers experience when learning an API. Robillard & Deline (2011) explore intent documentation as a knowledge barrier, meaning why certain decisions are made and how the API is intended to be used. Several studies find difficulties in integrating multiple API elements when solving a task (Duala-Ekoko & Robillard, 2012; Robillard, 2009; Robillard & DeLine, 2011). Matching APIs with scenarios is another barrier to learning to use a new API (Robillard & DeLine, 2011). This illustrates the user's ability to match a challenge to the documentation provided. The API's capabilities of allowing exploration and understanding its parts also affect the user's experience learning the API (Robillard & DeLine, 2011).

Gao et. Al (2020) observed participants' experience learning an API. They classify three stages of activities that learners alternate between when learning a new API: *information collection*, *information organization*, and *solution testing*. During the observation, it was noted that participants tended to move straight from collecting information to testing potential solutions after locating a relevant information source. In the solution testing phase, developers frequently iterated between editing & testing the code. The study implies that code emulators likely encouraged more code comprehension, allowing participants to interact with the code & perform solution testing.

2.2.2 Designing APIs with learnability in mind

Another perspective of improving API learnability involves emphasizing the creation of more well- designed APIs. Even though APIs are used to build distributed software systems, they must also be understood and used by complementors using such APIs. In the same way there's spent time designing the user experience for user interfaces, it's essential to enhance the user experience of using the provided APIs. APIs must offer the necessary features and capabilities; however, they may become impractical and unusable due to poor design (B. A. Myers & Stylos, 2016). Research has suggested that improving learnability can be addressed by adapting the

API (Piccioni et al., 2013). By utilizing usable APIs, developers can increase their productivity, as these APIs are more intuitive, require less documentation browsing, and encourage reuse (Piccioni et al., 2013). Usability influences adoption. If the API takes too long to learn, programmers may use another API or write functionality from scratch (B. A. Myers & Stylos, 2016; Tello-Rodríguez et al., 2020).

Enterprise APIs context adds an additional layer of complexity due to the large number of services, the variation of internal data structures, and services which is depending on each other (Beaton et al., 2008). The APIs of these enterprise systems are particularly interesting to study as they are often large and complex, carrying issues of scale and targeting a large group of developers (Jeong et al., 2009). Due to these circumstances, enterprise APIs are identified as having many usability challenges (Beaton et al., 2008). This includes i.e., participants facing difficulties finding relevant API elements for performing an API task using the enterprise API (Jeong et al., 2009).

2.2.3 API documentation

Although documentation is found to play a critical role in conveying API knowledge, it is often an imperfect resource (Parnin & Treude, 2011; Robillard, 2009; Robillard & DeLine, 2011). Robillard & DeLine (2011) find that inadequate API documentation represents the most severe obstacle developers face in learning a new API. Numerous studies have found incorporating code examples attractive and essential in API documentation (Brandt et al., 2009; Jeong et al., 2009; Meng et al., 2018; Robillard, 2009; Robillard & DeLine, 2011). As suggested by McLellan et al. (1998), “*The code example supported several different learning activities, such as understanding poses of the library, its usage protocols, and its usage contexts.*” (page 83). Other research confirms this observation, arguing that code examples can reduce mistakes and improve the success rate and developers' satisfaction with using the API (Sohan et al., 2017).

Jeong et al. (2009) conducted a study on the usability of the online API documentation for enterprise service-oriented architecture provided by SAP. The study found that familiarity with the terminology used in the documentation, especially acronyms, was important for understanding the documentation. To enhance the usability of online API documentation, the study suggests providing an overall map, using a balance of diagrams and texts, using generally-understood terms or clearly explained terminology, and giving services names users can easily recognize and distinguish from each other. Additionally, multiple studies have recommended making service parameters more prominent, i.e., by the distinction between

optional and required parameter fields (Daughtry et al., 2017; Jeong et al., 2009). Furthermore, several studies recommend providing online service testing for testing the API. This can help developers understand and consume the service, verifying their understanding of its usage and improving their overall experience (Beaton et al., 2008; Danielsen & Jeffrey, 2013; Daughtry et al., 2017; Gao et al., 2020).

Some existing tools can assist in trying out an API without a significant time investment, such as the Swagger UI. In recent years, these tools have been introduced in several large-scale firms, such as Google and SAP, to support developers learning their APIs (Gao et al., 2020). The tools share the common traits of providing a graphical interface for users to enter a request and a button for submitting the request, calling the API with the inserted data, and retrieving a well-formatted response. Studies show that users make heavy use of such tools either instead of documentation or used together with reading documentation (Daughtry et al., 2017; Macvean, 2016). Users of these tools found them helpful, i.e., for finding and evaluating if an API endpoint suits their needs or for learning to invoke an API endpoint without writing code (Daughtry et al., 2017). The explorer was found to be an easy and fast way to test the APIs for users by not having to deal with complexity such as authentication, programming languages, and syntactic issues. It was also found useful for guiding users with feedback on whether they were going in the right direction by having an easily editable request and continuous feedback of changes. Daughtry et al. (2017) paper suggests improvements for these explorers easing the usage of an API explorer by including conditional field requirements, field format guidance, and error response interpretation.

2.3 Chapter summary

Platform ecosystems are made up of a platform providing core functionality and complementary applications extending this functionality using the platform's interfaces. The platform owner must provision resources, referred to as boundary resources. Boundary resources such as APIs enable complementors to engage with the platform and extend its functionality. However, complementors require sufficient knowledge in accessing and extending the platform's boundary resources imposing a knowledge boundary between the platform owner and complementors. These knowledge boundaries are particularly significant for enterprise systems due to their complexity of usage. KBRs are introduced as means to overcome such knowledge boundaries. Current research on supporting developers in learning APIs can be divided into three streams: programmers' experiences learning APIs, designing APIs with learnability in mind, and API documentation. Table 2.3 below summarizes key concepts and theories which is important in related research:

Term/Theory	Description
Platform Ecosystem	A system consisting of a platform providing core functionality and complementary applications that extend the platform's functionality.
Enterprise Platform Ecosystem	Complex platforms that encompass a wide set of functionalities, architectures, and business lines, requiring facilitation of third-party participation for application development.
Boundary Resources	Software tools and regulations that serve as an interface between the platform owner and complementors, enabling them to engage with and extend the platform functionality.
Knowledge Boundaries	Boundaries between platform owner and complementors as complementors need sufficient knowledge for development.
Knowledge Boundary Resources (KBRs)	Objects and activities employed by platform owners to overcome knowledge boundaries and enable effective product development outcomes. These are categorized into scale and scope, including broadcasting, brokering, and bridging KBRs.
API learnability	The ease with which developers can understand, learn, and use an API to build applications.

Table 2.3: Summary of terms and theories

3 Research Approach

This chapter outlines the research approach conducted over a span of 1.5 years, focusing on designing artifacts to support 120 new complementors learning to use the APIs of the enterprise platform DHIS2. First, the chapter begins with a case description that sets the context and provides background knowledge about the research conducted. Second, the research methodology for conducting the research is described. Third, the data collection is discussed, explaining the various stages and methods used to collect data. And finally, the data analysis is presented, which explains how the data was analyzed and ultimately led to the formulation of the design considerations, which will be presented in the chapter Design considerations

3.1 Case description

3.1.1 HISP and DHIS2

The Health Information System Programme (HISP) is a global initiative centered at the University of Oslo (UiO) and founded in 1994. The primary focus of HISP is to promote and strengthen health information systems in low- and middle-income countries. HISP aims to “enable and support countries to strengthen their health systems together with regional HISP groups through increased capacity to govern their Health Information Systems in a sustainable way to improve the management and delivery of health services.” (*HISP UiO Strategy Update 2019-2022*, 2019)

Central to the HISP project is developing an enterprise platform called DHIS2. DHIS2 is a free, open-source software platform for collecting, analyzing, visualizing, and sharing data. The platform is fully customizable to various use cases, making it a flexible solution for diverse data management needs. Primarily used for managing health-related information, DHIS2 is the world’s leading Health Management Information System (HMIS) (*DHIS2 Overview*, n.d.). DHIS2 has seen widespread adoption, with more than 76 countries worldwide utilizing the platform for collecting and analyzing data. This accounts for approximant 3.2 billion people, or 40% of the world’s population, living in countries where DHIS2 is used (*Home*, n.d.).

3.1.2 DHIS2 as platform

The DHIS2 platform serves as a foundation for various health information system applications. As a software platform, DHIS2 serves core functionality, enabling complementors to build custom applications for the platform. Complementors can create DHIS2 applications, either an Android application or a web app, and connect the app to a DHIS2 instance that offers APIs. The API enables complementors to access and manipulate data stored in the DHIS2 instance. DHIS2 also offers a developer portal, serving the essential resources for complementors to building applications for the platform (*Welcome to the DHIS2 Developer Portal | DHIS2 Developer Portal*, n.d.). Once an application is developed, the creator can share the apps with the global DHIS2 community by deploying it to the DHIS2 App hub, as shown in Figure 3.1.

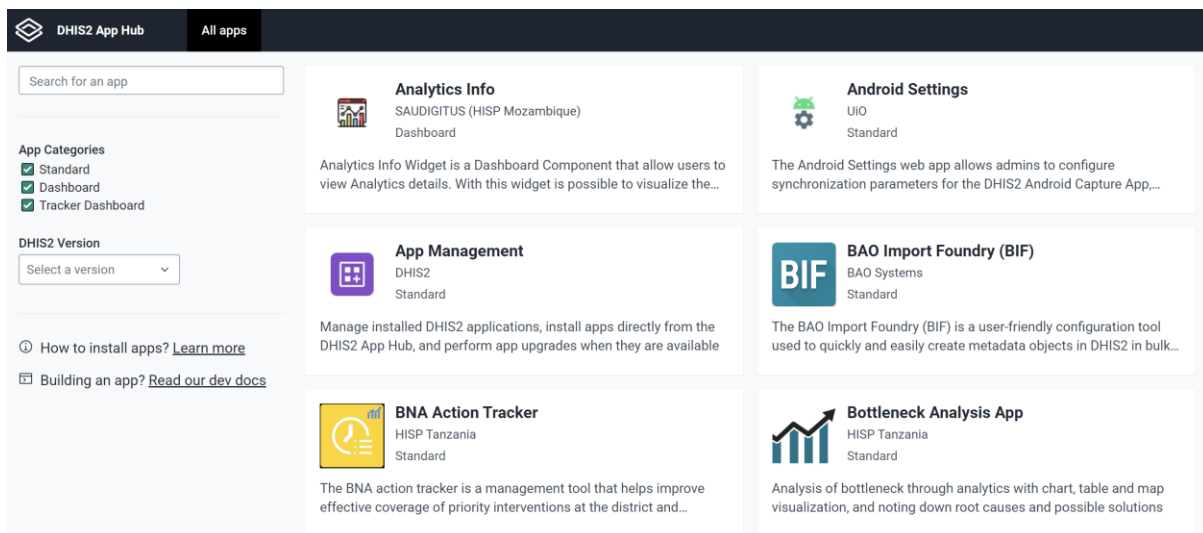


Figure 3.1: The DHIS2 app hub shows some of the available applications.

This thesis employs the DHIS2 platform in understanding how to support complementors in learning the APIs of such enterprise platforms. Exploring the DHIS2 platform and resources contributing to learning the DHIS2 APIs can provide valuable insight into how platform owners can support its complementors in learning the APIs.

3.1.3 DHIS2 Design lab

The DHIS2 Design lab, an initiative based at the Department of Informatics at the University of Oslo, supports and promotes design and innovation within the DHIS2 ecosystem and contributes to broader design, innovation, and digitalization research. The lab consists of researchers and post-graduate students collaborating with the DHIS2 core developers and implementation specialists worldwide (*DHIS2 Design Lab - HISP Centre*, n.d.). This research

project has conducted research as part of the DHIS2 design lab, exploring how resources can be designed for learning to use the DHIS2 platform’s API, making it relevant to the lab’s overall objectives.

The DHIS2 design Lab generates knowledge applicable not only to the DHIS2 but also to research on design, innovation, and digitalization in general. The lab works closely with the UiO master-level course “Development in Platform Ecosystems,” using it as a testing ground for the development of DHIS2 platform resources. Through this course, the design lab explores how to develop capacity-building resources, aiming to apply these findings to enhance application development within the DHIS2 ecosystem. My research project has used the course to explore how resources can be designed to support complementors learning to use a platform’s API. The course will be discussed in the following section.

3.1.4 Development in Platform Ecosystems

Students taking the course masters “Development in platform ecosystems” at the University of Oslo, hereby referred to as “course,” are taught about application design and development principles and practices within platform ecosystems. The course timeline, as shown in Figure 3.2, begins with an introductory lecture before the students take on an online self-paced online course lasting six weeks, hereby referred to as the “DHIS2 app course”. Each second week of this DHIS2 app course, the students must complete mandatory individual assignments, graded to passed or not passed. By the end of the six weeks, students should have a basic understanding of implementing applications for DHIS2. After completing the DHIS2 app course, students should form groups and, throughout a new six-week period, create a DHIS2 application based on a project description.

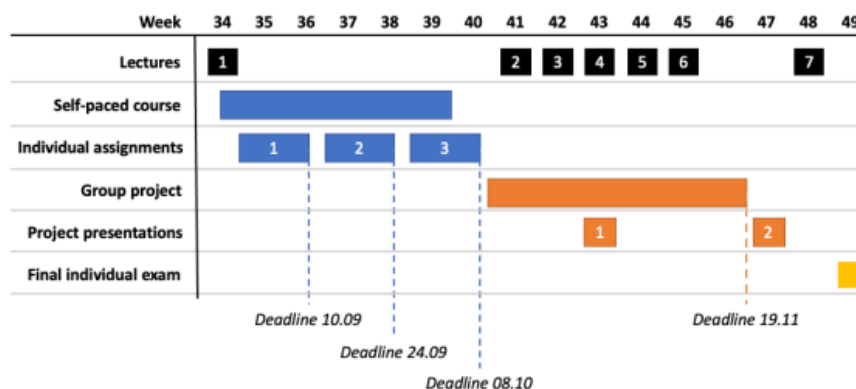


Figure 3.2: Timeline of the course

The DHIS2 app course serves as an important resource for supporting students in learning development for the DHIS2 platform, both during the self-paced period and while completing the group project. The DHIS2 app course consists of six learning modules covering different topics, as shown in Figure 3.3. These modules guide students from basic front-end development skills all the way to creating apps for DHIS2 with JavaScript and React.

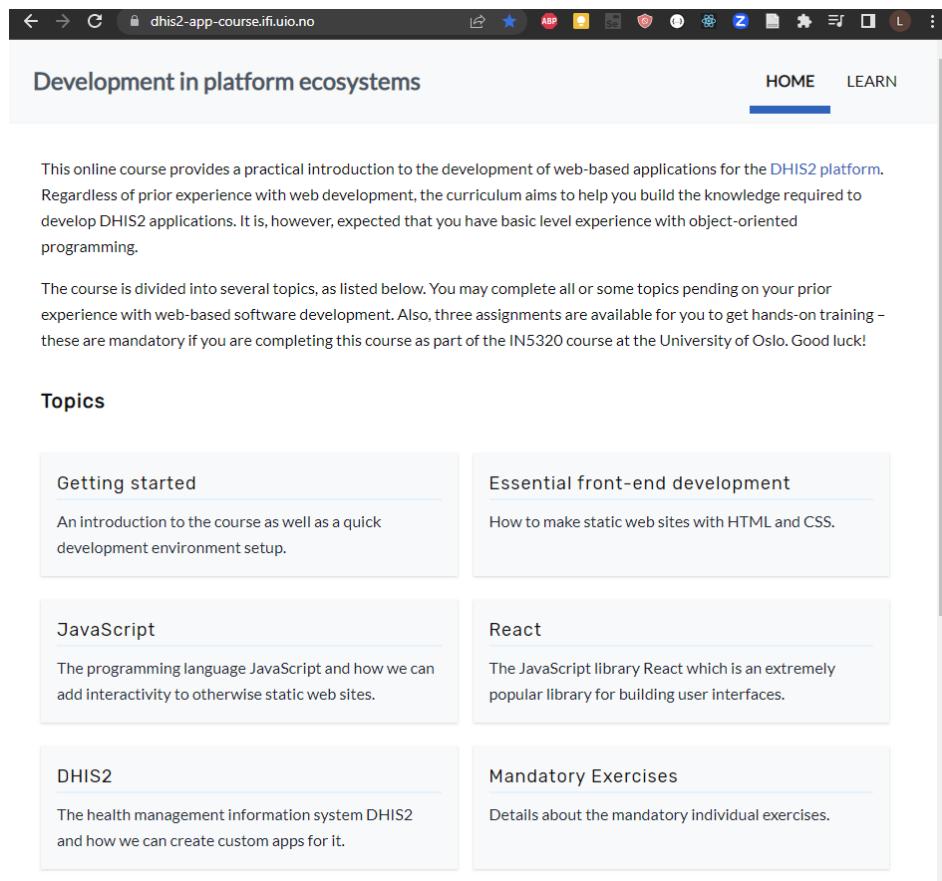


Figure 3.3: The DHIS2 app course front page

3.1.5 DHIS2 Design lab: developing and evaluating learning resources

This research project explored how to design resources that support new complementors learning an enterprise platform’s API. The DHIS2 app course was used as a practical setting for this exploration by making changes to the design of this course. Many students taking this course had no prior experience developing applications for the DHIS2 application, meaning that designing resources in this course could give valuable insight. By tailoring resources in this course, we could better understand how to support new complementors learning a platform API. These findings would also inform the research question, "*How can enterprise platform owners design knowledge boundary resources to support large audiences of new complementors in learning to use the platform APIs?*".

3.2 Research methodology

This thesis explores how platform owners can design knowledge boundary resources for supporting new complementors in learning the platform's APIs. By adopting a qualitative research approach that captures students' experiences as they learn and use the APIs of DHIS2, valuable insight can be gained into designing these resources. Qualitative data sources such as observations and interviews (M. D. Myers, 2020) are used to gain an in-depth understanding of the learning process and inform the design of knowledge boundary resources.

3.2.1 Design science research

The methodology used in this thesis is design science (DS) research. DS involves a rigorous process of creating and evaluating IT artifacts to solve organizational problems (Hevner et al., 2004; Peffers et al., 2007). The artifact can include any designed object targeting to solve an understood research problem, such as instantiations (implemented and prototype systems) or methods (Peffers et al., 2007). Artifacts designed using DS enable researchers to understand and address problems related to developing and successfully implementing information systems for organizations (Hevner et al., 2004). Considering the thesis seeks to address problems complementors encounter when using platform boundary resources, DS can be used by creating an artifact addressing these problems. The development of the artifact should be built based on existing theories and knowledge, coming up with solutions to the defined problems (Peffers et al., 2007).

In conducting the DS for this thesis, the design science research methodology (DSRM) introduced by Peffers et al. (2007) was used throughout the research process. Their methodology builds on top of prior research, targeting to find commonly accepted approaches for carrying out the research rather than focusing on difference views of DS research. By grouping common elements commonly agreed upon from existing literature, they present a process model which consists of six activities for conducting DS in information systems. This model is shown in Figure 3.4 (Peffers et al., 2007).

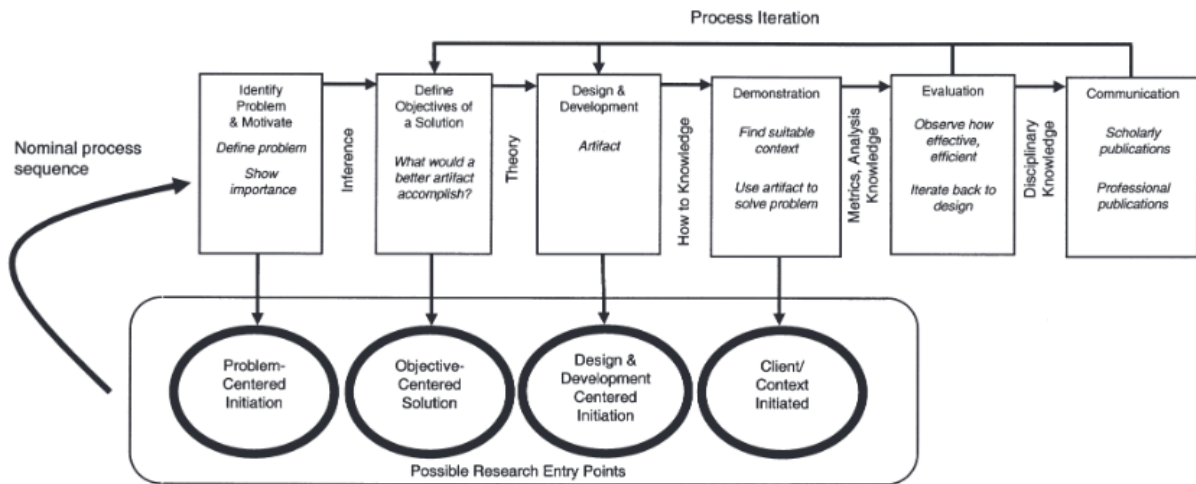


Figure 3.4: Model for conducting DSRM (Peppers et al., 2007, p. 46)

In the following sections, the thesis will go through the six activities outlined in the design science research model by Peppers et al. (2007), explaining what they are and how they were applied to this research project. It's important to note that although the method is explained as a “linear process” in the following section, the reality of conducting the research was different. Throughout the research period, there was a transition back and forth between the activities based on new insights and findings. Before delving into the details of each activity, a timeline is presented in Figure 3.5, showing when the different activities of the DSRM took place during the research period.

Thesis timeline



Figure 3.5: Timeline of DSRM activities during the research period

The first activity of the DSRM is to *Identify the problem & motivate*. This includes defining a research problem and justifying the value of developing a solution (Peppers et al., 2007). As the solution should resolve the research problem, inspecting the problem in more detail is recommended by Peppers et. Al (2007) to ensure the solution covers the problem's full complexity. The activity should state the current knowledge state of the problem and its importance, motivating both the researcher and its audience to pursue the solution and accept its results.

This research project identifies the problem through a collaborative effort with previous course teachers. The usage of APIs was identified as a significant concern, leading to it becoming the focus of this research. A thorough review of related literature further emphasized the importance of addressing this issue (Danielsen & Jeffrey, 2013; B. A. Myers & Stylos, 2016). As a result, the research focused on supporting new complementors learning the platform APIs. By establishing the significance of the problem and understanding its complexity, this activity provides a clear direction and motivation for pursuing the research and developing an artifact for addressing the identified issue.

The second activity was to *define objectives for a solution*. As Peppers et al. (2007) described, this involves inferring a solution's objectives based on the problem definition and the knowledge of what is possible and feasible. The objectives can be quantitative, such as terms in which a desirable solution can be better than the current ones, or qualitative. The objective should also be rationally inferred with the problem specification. The researcher should familiarize themselves with knowledge of the problem and current solutions and their efficiency.

In this research project, the solution's objective was defined based on the identified problem. Given the challenges faced by new complementors in learning to use APIs, it was logical to establish an objective for a solution that supports them in this process. This objective was also reflected in the research question. Together with related research, existing solutions and methods for teaching APIs were explored to identify opportunities for improvement.

The third activity is to *create the artifact*. Such artifacts take various forms, serving to embed a research contribution in the design. This activity involves determining the desired functionality and architecture and then creating it (Peppers et al., 2007).

Over a seven-month period in the research project, web resources were designed that expanded the current DHIS2 app course content. This design process is detailed in the Design and Demonstration chapter. Most of the design of these web resources took place both before the beginning of the course. However, some adjustments also had to be made during the course period.

The fourth activity was the *demonstration*. This involves showing how the artifact can solve one or more instances of the problem (Peffer et al., 2007). To successfully demonstrate the artifact, researchers need to thoroughly understand how to use it to address the problem at hand.

In the context of this research, the demonstration involved conducting the course where students utilized the artifact for learning to use the DHIS2 APIs. Throughout the course, students engaged with the developed artifacts, which provided them with guidance and resources to understand better and work with the API. The demonstration was used to determine how the artifact addressed one or more problems identified with complementors learning platform APIs. This allowed for a more targeted assessment of the solution's effectiveness in overcoming the challenges faced by new complementors.

The fifth activity is *evaluation*. This activity entails observing and measuring how well the artifact supports a solution to the problem (Peffer et al., 2007). The evaluation requires comparing the solution's objectives to the actual observed results from using the artifacts in the demonstration.

In the research project for this thesis, the evaluation process involved collecting data through various means, such as observations, interviews, presentations, assignments, and surveys. This data provided valuable insight into how well the developed artifact supported the students in learning the DHIS2 API. By evaluating this data, the effectiveness of the designed solution could be assessed, and further improvements could be identified. The evaluation allowed a comprehensive understanding of the solution's impact on addressing the identified challenges faced by new complementors when learning platform APIs.

The sixth and final activity is *communication*. This activity involves conveying the problem, its importance, the artifact, its utility and novelty, the rigor of its design, and its effectiveness to research and other relevant audiences, such as practical professionals (Peffer et al., 2007).

In the research project for this thesis, communication is done by presenting both practical and theoretical contributions. The practical contribution includes a set of design considerations that

support new complementors in learning platform APIs. The thesis also contributes to research by extending knowledge on broadcasting KBRs.

In summary, the research project followed the Design Science research approach through six key activities. Next, the thesis will present an overview of how the data was collected throughout the research process.

3.3 Data collection

Throughout the research period, various data collections were used to gather relevant information for the thesis. This data collection subchapter will provide an overview of these methods and describe how they were conducted. The data collection was organized into three stages: *diagnosis*, *design and demonstration*, and *evaluation*. Within each stage, several activities outlined in DSRM were employed to gather data. Before describing each stage in more detail, a brief introduction is provided below.

The *diagnosis* stage encompassed problem identification, providing motivation, and defining objects for a solution (Activities 1 and 2 of DSRM). This stage focuses on the initial research phase, where the problem is acknowledged, and goals are set.

The *Design and demonstration* stage included creating the artifact and demonstrating its use (Activities 3 and 4 of DSRM). This stage emphasized the practical aspects of designing the artifact and showcasing its application in solving the identified problem.

The *Evaluation* stage involves the evaluation activity (Activity 5 of DSRM). This is centered around assessing the effectiveness of the artifact in addressing the problem and meeting the solution objectives.

Please note that communication activity (Activity 6) in DSRM is not discussed in any of these stages. No data collection took place during the communication activity, as it primarily focused on presenting the findings and contributions.

3.3.1 Diagnosis stage

In the diagnosis stage, various methods were taken to identify the challenges faced by students taking the course. One of the methods was to conduct a workshop early in the research, which featured a semi-structured workshop with five group teachers. A group teacher is a student who has previously taken the course and is now teaching new students taking the course. This workshop lasted for 1.5 hours, focusing primarily on students' experiences of learning the DHIS2 APIs, how the current DHIS2 app course resources supported the learning of the APIs, and how these resources could be improved. Additionally, discussions were held with professors teaching the course to identify issues that students experience. This provided valuable insight into the challenges faced by students from an educator's perspective.

Another method employed in the diagnosis process involved reviewing surveys conducted by the previous year's group teachers. Some of these group teachers had researched the

onboarding of app developers in platform ecosystems. During their research, they conducted several surveys to identify issues with the onboarding process. Some of the questions within these surveys were related to API usage, which this thesis used in diagnosing difficulties that students had in learning and using the API.

Lastly, as a student who has taken the course and has worked with many fellow students taking the course, I also considered personal experiences during the diagnosing stage. This included experiences and feedback gathered from others who took the course simultaneously. This approach offered a unique way of a more comprehensive understanding of the problems faced by students when learning to use the APIs within the course.

3.3.2 Design and demonstration stage

Several methods were used in the design and demonstration stage of the data collection. When designing the resources, existing tools such as Postman, Swagger, and Data Query Playground from DHIS2 were used as inspiration when designing the new resources. Additionally, existing research was used as inspiration in designing the resources, and prototypes were created to better understand how the resources should be implemented.

An initial version of the artifact was designed over a period of three prior to the beginning of the course, which fed into data collection as many questions arose during the design process and required ongoing research. This provided valuable insight into how to design resources for the DHIS2 app course.

After designing an artifact based on prototypes, usability testing was conducted to refine the artifact. Five persons were invited for an observation where they were asked to try out the newly designed artifact. Each observation took around 1,5 hours and gave insight into changes required before the course started. When the course began, and students started using the DHIS2 app course, observation played a significant role in data collection. I attended seminars as a group teacher weekly for two hours (in total 24 hours), helping students learn the course content. During these seminars, I discussed with students and gathered feedback.

3.3.3 Evaluation stage

During the evaluation phase of data collection, interviews and focus groups were employed to gather feedback on the learning resources after the course was completed and students had finished their projects. Semi-structured interviews were conducted with nine students who had just completed the project. The interviews lasted approximately 1.5 hours each and covered

two main topics: 1) students' overall experience learning the platform resources and 2) their experience learning DHIS2 API. Additionally, two focus groups were held, where multiple students attended and engaged in an open discussion about the interview questions.

Interviews were conducted until saturation occurred, meaning no significant new information was added to the data collection. I held the interview as the researcher. The interviews were either face-to-face or via the online video service Zoom. The interviews took place over two weeks, and the students selected for interviews were chosen based on their backgrounds. To ensure various experiences and perspectives, most participants were from different groups that were formed when completing the group project.

3.3.4 Summary of data collection

To summarize the data collection process, Table 3.1 is presented. This table outlines the different methods conducted in each stage, the role of the participants involved, the number of attendees, and an estimate of time spent for each method.

Stage	Method	Role of participant	Participants	Amount
Diagnosis	Workshop	Previous group teachers	5	~ 1,5 hour
	Survey	Last years students	13	-
Design and demonstration	Prototyping	Researcher	1	~ 10 hours
	Design artifact	Researcher	1	~ 120 hours
	Usability testing	Developers	6	~ 9 hours
	Observation	Students		~ 24 hours
	Document analysis	Group teachers	~180 assignments	
Evaluation	Interview	Students	9	~ 14 hour
	Focus groups	Students	2	~3 hour

Table 3.1: Summary of data collection

3.4 Data analysis

In this thesis, the data analysis was conducted using thematic analysis, involving searching across the data set to identify repeating patterns of meaning (Braun & Clarke, 2006). The thematic analysis was performed at several phases throughout the research project, eventually leading to design considerations that will be discussed in a later chapter. This subchapter will detail the analysis process for each of these phases. First, we will review the preliminary analysis, where preliminary data was processed to identify problems. Second, we will go through the design analysis describing how the data was processed to inform the design of the artifacts. Thirdly, we will go through the evaluation analysis on how data was processed for evaluating the design. Finally, we will review how the design considerations were constructed based on the previous analysis.

3.4.1 Preliminary analysis

The workshop of previous year’s group teachers facilitated identifying problems and defining solution objectives. Recordings of the workshop were transcribed. These transcriptions were categorized based on subjects, whereas each subject identified a current challenge with the learning resources such as “terminology.” After gathering an overview of the current sections of concern, another iteration of analysis from this material was conducted. Some categories from the first iteration were excluded due to a lack of data. During the second iteration, a table was produced to structure the categorized data based on A) the challenge identified, B) A detailed description of the challenge, and C) empirical quotes supporting this finding. Figure 3.6 shows a part of this produced table

Challenge	Description	Empirical findings
Difficult terminology	Students had issues understanding terminology	<ul style="list-style-type: none"> - En oversikt over terminologi du trenger for å forstå prosjektet vil være nyttig (04:10). F.eks datamutation, dataQuery og meta data osv - Terminologi er viktig å få med seg. Det er ikke tydelig for folk hva det betyr. Platformspesifikke ord (06:35) - En «basic» challenge er syntax forskjeller. Folk bare ser ikke syntax forskjellen (17:15) - Det er utfordrende som student å forstå DHIS2 er som vanlig webutvikling, men med generisk terminologi osv (39:30)
Difficult data model	Students had difficulties understanding the data model	<ul style="list-style-type: none"> - En stor greie var DHIS2 datamodellen 03:50 - Hele greia er generisk, så det er generiske termer. Den abstraksjonsbarrieren gjør det vanskelig (06:45) - Du har datamodellen og hvordan ting er satt opp er en stor utfordring på DHIS2 (4:40)

Figure 3.6: Part of the table showing recognized challenges.

Based on this table, the research identified six categories students encountered while completing the course. These include 1) difficult terminology, 2) difficult data model, 3) high usage complexity, 4) query translator, 5) Document cross reference, and 6) learning by doing. Many of the categories included concerns regarding the usage of the API. A survey conducted by the previous year's student group teachers was used to supplement identifying areas of concern. This survey confirmed that usage of the API had a significant concern, showing that 53,8% of the students claimed using the API was very challenging, and 30,8% claimed it was somewhat challenging of out 13 respondents. Furthermore, personal experiences as a student taking the course and discussions with the professor holding the course confirmed the recognition of API being a challenge in the learning process. Overall, this analysis phase was used as motivation for defining the problem and setting objectives for the solution.

3.4.2 Design analysis

After establishing an initial understanding of the existing challenges, the design analysis phase began. This analysis focused on the design and development stage, which informed the artifacts' design. To create resources that effectively supported students in overcoming the diagnosed challenges, relevant literature and existing artifacts related to conveying API knowledge were used as inspiration for the designing phase.

From the preliminary analysis, it was evident that learn-by-doing was a desired feature. There was already a resource deployed from DHIS2 to support interactivity, mainly a tool named “Data Query Playground” where users can write and run API queries. Further analysis of this tool was done to identify its challenges and analyze how it can support the desired requirement of learn-by-doing. Based on a survey conducted by the previous year's group teachers, some of the shortcomings of this tool were discovered, as shown in Table 3.2.

Challenge	Description
Bugs	Many students reported having difficulties using the playground as there were bugs causing the the cursor and text input field to be inaccurate. When writing text into the playground, it would not be inserted where the users expected it to be.
Not user friendly	Students reported that they did not understand how to use the DataQuery playground. The input format was unknown, causing confusion as to how it should be used.

Table 3.2: Challenges identified in “Data Query Playground”

Based on the challenges associated with the tool, and the nature of it being a DHIS2 app with practical limitations in further development and implementation into existing code bases, it was decided to design a new interactive tool to complement the learn-by-doing.

The design analysis phase spanned seven months. This iterative design analysis involved usability testing, observing participants' experiences, and correcting design flaws based on inadequate design. Throughout usability testing, the researcher analyzed user interactions with the artifact, identifying poor design decisions and gathering feedback to further refine the design. By examining the results through these usability tests, several iterations of design were conducted to support the usage of the resource. Further analysis of the designed artifacts was done throughout the evaluation phase.

3.4.3 Evaluation analysis

During the evaluation phase, interviews and focus groups were conducted on students' experiences using the designed artifacts. These interviews and focus groups were transcribed. Based on the transcriptions, this data was coded as shown in Figure 3.7 using open coding, an interpretive process that allows for a more in-depth understanding of the data by breaking it down into smaller parts, comparing them, and assigning conceptual labels (Corbin & Strauss, n.d.).

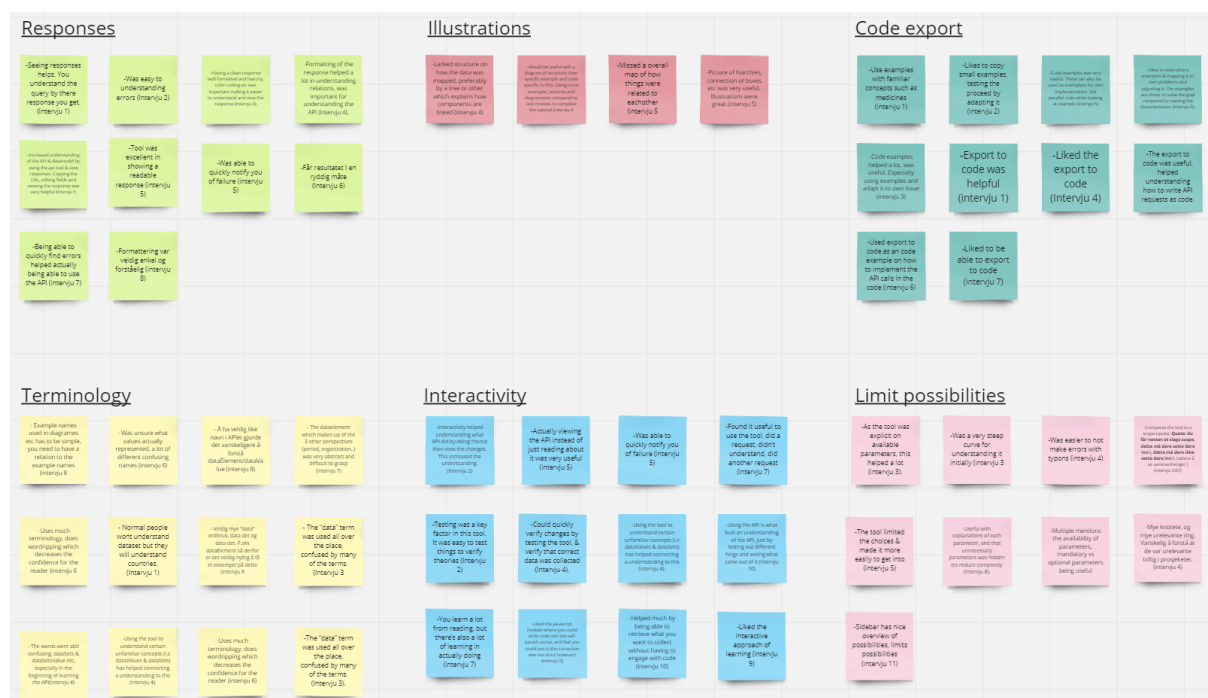


Figure 3.7: Initial coding of the data collected

After the initial coding of the data, these categories were inserted into a table. Figure 3.8 shows a part of this table, including the columns a) a title for the category, b) a longer description of how this category affected API learning, and c) empirical findings supporting the category.

Category	Description	Example of empirical findings
Descriptive terminology Terminology pairing/mapping	<ul style="list-style-type: none"> - Begrepet må være overkommelige, ikke brukes i unødvendig sammenhenger (gå inn på disse diffuse ordene osv). Folk forteller fra intervjuene at organisationUnit gir mening, men ikke de andre. Probably pga «organisation» i navnet?. - Også mange like navn dataElement og dataSet (svært like navn også). DataSets og dataSet. - dataSet <- sets skal heller ikke være i navnet. I tillegg til at ordene er intetsigende i seg selv, blir det et nytt lag av kompleksitet når du har flere ord som er nokså like hverandre. Vanskelig å både forstå betydningen av de, OG å skille de fra hverandre ettersom de er flere intetsigende nokså like ord. Multiple report that by using example names instead of these generic names helps understanding such words. There's also reports that by actually using the API/tool, it makes more sense with these words. The example words should also be relatable for users, such as not <code>oxcontijn</code> but rather antibiotics e.g. 	<ul style="list-style-type: none"> - Normal people <code>won't</code> understand dataset but they will understand countries. (Intervju 1) - The "data" term was used all over the place, confused by many of the terms. (Intervju 3) - The words were <code>abit</code> confusing, <code>dataSets</code> & <code>dataSetValue</code> etc, especially in the beginning of learning the API (intervju 4) - Using the tool to understand certain unfamiliar concepts (i.e. <code>dataValues</code> & <code>dataSets</code>) has helped connecting a understanding to this (intervju 4). - Uses much terminology, does <code>word tripping</code> which decreases the confidence for the reader (intervju 6) - Was unsure what values actually represented, a lot of different confusing names (intervju 6) - The <code>dataElement</code> which makes up of the 3 other perspectives (period, <code>organization</code>...) was very abstract and difficult to grasp (intervju 7) - Veldig mye "data" ordbruk, data det og data det. F.eks <code>dataElement</code> så derfor er det veldig nyttig å få et eksempel på dette (intervju 8) - Å ha veldig like navn i APIet gjorde det vanskeligere å forstå <code>dataElement/dataValue</code> (intervju 8) - Example names used in <code>diagrams</code> etc has to be simple, you need to have a relation to the example names (intervju 8) - Bruke female condoms veldig mye innad i gruppen når de snakket med hverandre i gruppen. De brukte bare ordene som f.eks female condoms i kommunikasjon innad i gruppa. Også brukte de ordet commodity, de kalte det aldri dataSet. Så intern i gruppa så snakket de kun i eksempel format i gruppe når de snakket om disse generelle elementene (intervju 8)
Test assumptions	<ul style="list-style-type: none"> - Interaktivitet (både i api toolet, men andre nevner også denne start modulen hvor du kan teste html+css kode som de også trekker frem som nyttig) -> 1.Lite kompleksitet for å sette opp ting så det er lav terskel for å teste ting og 2. Hyppig feedback/rask feedback. Iterasjonene ser også ut til å skape en bedre forståelse. Så raskere iterasjoner interasjoner = mer forståelse. Og lavere terskel for dette ved å tilrettelegge for 	<ul style="list-style-type: none"> - Interactivity helped understanding what API did by doing thence then view the changes. This increased the understanding. (Intervju 2) - Testing was a key factor in this tool. It was easy to test things to verify theories (Intervju 2) - Could quickly verify changes by testing the <code>tool</code>, & verify that correct data was collected (Intervju 4). - Using the tool to understand certain unfamiliar concepts (i.e. <code>dataValues</code> & <code>dataSets</code>) has helped connecting a understanding to this (intervju 4). - Was able to quickly notify you of failure (intervju 5)

Figure 3.8: Part of the table showing categories affecting API learning.

The table was continuously analyzed by comparing the different categories and reviewing each in more detail. This process resulted in some of the categories being merged while new ones surfaced. This categorizing process provided a better understanding of the factors that impacted the student's API learning.

3.4.4 Constructing contribution.

Drawing from the analysis of earlier phases, a set of categories that affected API learning emerged through the evaluation. Initially, these categories were specific to the DHIS2 environment. Through iterative analysis and refinement, these categories were adapted into more general design considerations, applicable not only to DHIS2 but also to other platform owners. Figure 3.9 shows one of the first iterations of constructing the design considerations.

Consideration	Description
Termonology mapping	Participants had difficulties grasping & distinguish terminology in the interface that were platform-specific. It's expressed that using these words in practice & having samples the generic words increase the interpretation of such interface terminology.
Limit possibilities	Many participants expressed a introduction of the API which had demanding requirements of knowledge. Having a limit of possibilities, only giving a scope of what the user should be aware of was retrieved as useful by the users.
Test service	This service gives an option of quickly being able to very changes by running frequent tests. There's few takes from the interviews that many reports: <ul style="list-style-type: none">- Multiple report that it's fast to iterate the stages of<ul style="list-style-type: none">* Edit* View & verify changes- Multiple reports many iterations

Figure 3.9: Considerations evolved from an early iteration of the analysis

By carefully analyzing the data and refining the table of design considerations, these considerations contribute to insight for platform owners seeking to support new complementors in learning to use the platform APIs. The final design considerations are presented in Chapter 7 Design considerations.

3.5 Ethical considerations

Throughout the study, ethical considerations were taken to ensure the participants' privacy. During the observation process, I was transparent about my role as a researcher and the purpose of my observation. Participants were asked for consent if I would use the observation for my study, allowing them to make informed decisions about their involvement. Before conducting interviews, participants were made aware of the recording process and their right to withdraw their consent at any time. All data collected throughout the study was anonymized to protect participants' privacy. This approach ensured that individuals could not be identified, and their personal information remained confidential.

3.6 Chapter Summary

The chapter opened with a case description centered around DHIS2, a widely used open-source platform. The case description elaborated on how this research engaged in the DHIS2 design lab, which aims to expand knowledge applicable not only to DHIS2 but also to research on design, innovation, and digitalization in general. The research contributes to this knowledge by designing resources for students taking the “Development in platform ecosystems” course at the University of Oslo. Specifically, these resources were designed in the DHIS2 app course to support the students learning of the DHIS2 platform APIs. The research follows the Design Science Research Methodology (DSRM), conducting the activities involved in this methodology. The data collection included three stages: diagnosis, design and demonstration, and evaluation. Techniques such as workshops, surveys, interviews, and focus groups were employed, including participants such as students and teachers. Finally, the data analysis highlighted the usage of thematic analysis across different phases of the study. This process helped identify patterns and led to the formation of design considerations serving as valuable guidance for guide platform owners in designing broadcasting KBRs that supports complementors in learning the platform APIs.

4 Findings from Diagnosis

This chapter presents the findings from the diagnosis stage of the data collection. This stage was critical for identifying student challenges when taking the DHIS2 app course. Specifically, the chapter identifies the difficulties that last year's students taking the course experienced while learning the APIs. The data sources used in this stage included a workshop with group teachers who taught the course, discussions with professors holding the course, and personal experiences gained while taking the course myself as a student. Based on the findings, four challenges were identified, namely a) complex data model, b) complicated terminology, c) limited opportunities for learning by doing, and d) no query-to-code translation. These challenges are presented in greater detail in the following subchapters. The challenges served as a foundation for designing and demonstrating new resources supporting complementors learning the APIs, which will be covered in the next chapter.

4.1.1 Complex data model

The data model was a key challenge for students learning to use the API. Understanding this data model is essential for using the DHIS2 APIs. The data model is designed to be highly flexible, whereas each element of the model has a generic name and accommodate any data type. As shown in Figure 4.1, the data model is centered around the concept of a “DataValue” which can be recorded for any “DataElement” (representing the specific item, occurrence, or phenomenon being captured, “Period” (representing the time dimension), and “Source” (meaning the spatial dimension such as an organization unit within a hierarchy) (4.5 *The Data Model*, n.d.).

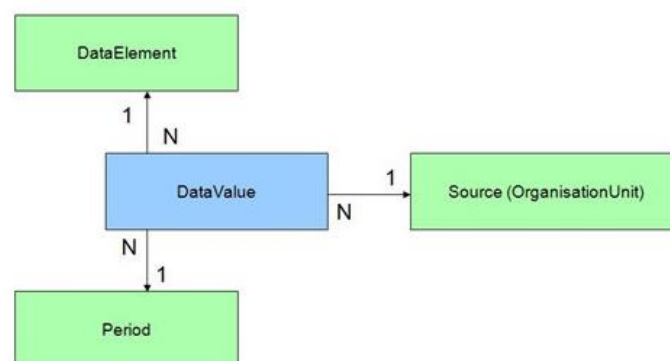


Figure 4.1: Representation of the data model

Students expressed difficulties due to the various generic names used in the data model, making it difficult to determine each element's purpose and how to use them. One group teacher explained, *“The data model and how things are set up in DHIS2 is a big challenge for users due to its complexity. [...] It contains many generic terms, and this abstraction barrier makes it much more difficult to learn.”* Additionally, each element has one or more relationships to other elements, adding further complexity. Another group teacher elaborates on this point, explaining:

“The biggest issue in understanding the API was the data model's complexity of the data relations, including relationships between these elements. [...] It was particularly difficult to retrieve elements with an only subset of their values, connecting such values to other elements, filtering and sorting based on the connected elements, and so on. It's a lot to get your head around.”

The difficulties of understanding the individual elements and their interrelations with others caused a learning barrier for students trying to use the API.

4.1.2 Complicated terminology

Terminology was one of the leading reasons students had difficulty grasping this data model concept. A group teacher expressed this in the workshop: *“Terminology in itself is very important for users to understand such as the data model. “dataset,” “dataValue,” “dataElement.” It's not clear to users what these platform-specific words mean. It's a big challenge.”*

The current DHIS2 app course did not sufficiently provide students with sufficient material to understand and apply the data model and terminology. This especially applied to platform-specific words where students had no prior knowledge about the terms and concepts. One of the difficulties which group teachers recognized in students learning new terminology was that similar terms would easily be mixed up and cause confusion. i.e., the DHIS2 app course introduces two types of DHIS2-specific code functions that should be used when interacting with the API: “useDataQuery” and “useDataMutation.” It was reported that many students had difficulties separating these terms. One group teacher said: *“We got so much feedback about people not seeing the syntax differences.”*

The DHIS2 app course lacked apparent differences between very similar syntaxes, making it difficult to distinguish the terminology. The group teachers highlighted this learning barrier and underscored the need for resources to address this issue. One of the group teachers suggested a possible solution: *"I think it's best to create a task-centred tutorial that would give many possibilities. [...] I think this is a good approach for especially new students to learn the usage of DHIS2."*

4.1.3 Limited opportunities for learning by doing

During the workshop, the learn-by-doing approach was frequently mentioned as an effective strategy for learning as one group teacher stated: *"Interactivity and learning by doing are large factors for people to learn anything. It doesn't help just viewing something. For people to learn, they have actually to try it out themselves."* The DHIS2 app course included an interactive learning resource that focused on teaching JavaScript, which received positive student feedback. However, no interactive resources were designed to facilitate learning the data model, terminology, or the API. Another group teacher suggested adding support:

"Tutorial-based learning resources where you complete tasks and learn to do things yourself is useful [...] In regards to the API, I think this is a good approach. Especially for new students. A sandbox environment would be useful if possible."

When students were observed exploring the API, they were found to be using a "brute forcing" strategy, meaning that they would repeatedly make minor changes to the API request before testing such changes. Producing a "sandbox" environment to support this activity where students could quickly apply and verify changes was emphasized as positive for the learning effects. The DHIS2 team had already developed an interactive tool called "Data Query playground," which supports this exploring activity. Students were recommended to try out this tool throughout the DHIS2 app course. However, it was not widely used by students due to lacking a user-friendly interface. One of the group teachers expressed this: *"Some students used DataQueryPlayground. It's some errors in it in general and not the most UX-friendly tool."*

While I completed the course as a student, I shared the same experiences as other students having difficulties using this tool. The difficulties were mainly due to two reasons: first, the DataQueryPlayground had bugs making it very difficult for users to use the tool. Whenever writing input to the tool, the characters would not occur at the expected position making it

difficult to provide input. Secondly, as the group teachers expressed, it's not a UX-friendly tool. It required the user to insert DHIS2-specific syntax for sending a request. As a result, the tool does not facilitate new developers with no knowledge of this syntax.

Besides the Data Query Playground tool, various other approaches were reported to be used by students to explore the DHIS2 API. One group teacher noted that some students experimented by simply writing queries in the web browser: *“Some students said it was easier to just write queries in the web browser [...]. This could be due to the errors they experienced in Data Query playground.”* However, using the web browser has limitations, such as formatting of the response, no syntax checks, and not supporting all HTTP verbs.

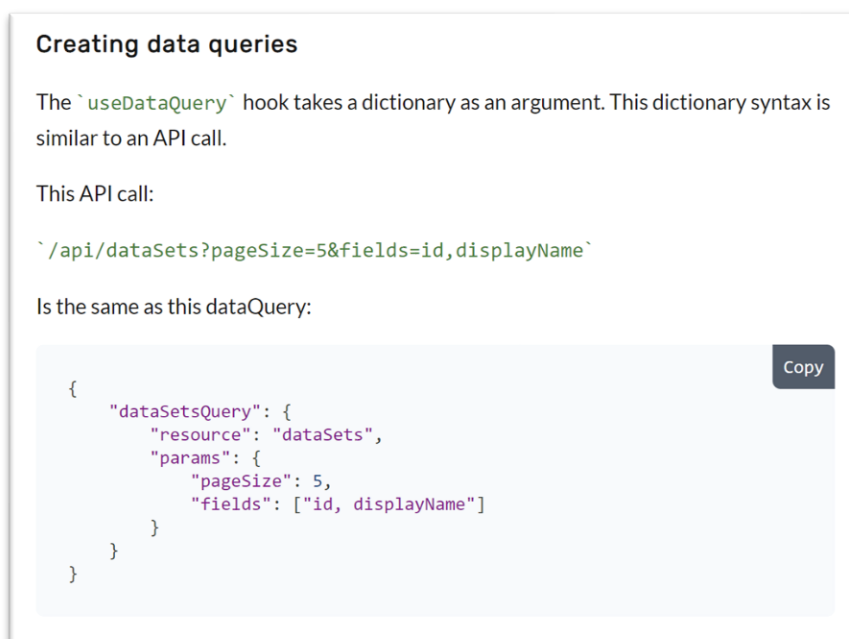
Other students were reported to be using the API software “Postman” to explore the API, but these students were generally more advanced users. A group teacher commented, *“One student group used postman, yet many other groups never heard of postman. [...] Experienced users find ways to solve problems, yet new developers have difficulties. [...] You have to support competence at the lower level.”*

Finally, some students were found to be exploring the API by editing code in their application for sending an API request, then building the application, and finally testing it, which was a time-consuming activity. A group teacher suggested, *“It's a good idea to let users the ability to easily adapt and test things instead of having to do this through their app, then editing, then running, then editing, and so on. It's much better through a playground.”*

The variety of approaches students use to explore the DHIS2 API indicates a lack of a cohesive strategy for exploring the API. The Data Query Playground, for example, has been found to have bugs and lacks a user-friendly interface. Thus, the student uses other approaches to explore the API rather than using the tool explicitly provided for this purpose. This indicated the need for further development of the Data Query Playground or other tools to better support the users' needs in exploring the DHIS2 API. Experienced students tend to find ways to solve their issues, indicating that the design of tools for exploring the API should emphasize supporting students with less experience.

4.1.4 No query-to-code translation

As previously mentioned, many students used a web browser to send API requests (referred to as “query”). Once the student had queried the API and the API returned the desired result, they encountered the additional challenge of applying this query to their code. To do so, they had to convert the query into a DHIS2-specific syntax for sending the queries by code. Figure 4.2 shows a description used in the DHIS2 app course of how to convert the query. As shown in the figure, the green line of text is how the query is sent using a browser, and the purple text is the same query using the DHIS2-specific syntax required for sending the request by code. Several students experienced difficulties converting to this syntax.



Creating data queries

The `useDataQuery` hook takes a dictionary as an argument. This dictionary syntax is similar to an API call.

This API call:

```
/api/dataSets?pageSize=5&fields=id,displayName`
```

Is the same as this dataQuery:

```
{
  "dataSetsQuery": {
    "resource": "dataSets",
    "params": {
      "pageSize": 5,
      "fields": ["id, displayName"]
    }
  }
}
```

Copy

Figure 4.2: Description in the DHIS2 app course of how to convert a query

Similarly, many students faced difficulties using query examples in the DHIS2 API documentation, as these also had to be converted. Overall, students struggled with writing and converting queries due to being unfamiliar with the DHIS2-specific syntax. One of the group teachers highlighted this challenge during the workshop: *“It should’ve been more code examples on converting API queries to appropriate format. [...] It doesn’t help to get raw string queries as they need to be translated.”*

To address this issue, group teachers proposed a helpful feature for the students – a converter that would translate queries to the DHIS2-specific syntax, which can be used by code. As one group teacher suggested: *“Another function which student requested was a query converter where you basically provide and query and retrieve back a code snippet.”* With the help of

this tool, students could focus on writing queries and exploring the API without worrying about appropriately converting the request such that it was applicable in code.

4.2 Chapter Summary

This chapter diagnosed the challenges students had when completing the DHIS2 app course.

Table 4.1 summarizes the main challenges identified in the diagnosis.

Challenge	Description
Complex data model	The data model is designed to be flexible, accommodating any data type, but its generic names and numerous relationships between elements made it difficult for students to understand each element’s purpose and use
Use of complex terminology	The platform-specific terms were a learning barrier for students. The current DHIS2 app course does not provide sufficient material for students to understand and apply the data model and terminology.
Limited opportunities for learning by doing	A lack of a learn-by-doing approach made it difficult for students to understand the data model and terminology. Task-centered tutorials and a sandbox environment supporting learning by doing was recommended as practical strategies for overcoming this challenge. The sandbox needs to support new complementors.
No query-to-code translation	Converting query strings to DHIS2-specific syntax such that these could be used in code was a significant challenge for the students. Group teachers proposed a converting tool to translate queries to the DHIS2-specific syntax to address this challenge.

Table 4.1: Summary of challenges identified in the diagnosis

5 Findings from Design and Demonstration

This chapter will delve into the findings from the next data collection stage, which was to design and demonstrate artifacts that addressed the challenges identified in the diagnosis stage. This stage was important for exploring how to design artifacts to address the identified challenges and support new complementors learning the APIs. As a response to the diagnosed challenges, two artifacts were designed. Firstly, the API testing tool was designed to address the issues of "limited opportunities for learn-by-doing" and "no query-to-code translation." Secondly, an introduction tutorial to the DHIS2 API was designed to address the "limited opportunities for learn-by-doing" challenge and complexities related to the "complex data model" and "complicated terminology" challenges. The artifacts and the challenges they address are summarized in Table 5.1. This chapter will begin by presenting the design of the API testing tool, followed by presenting the design of the introductory tutorial to the DHIS2 API.

Artifact	Responding to challenge
API testing tool	<ul style="list-style-type: none">• Limited opportunities for learn-by-doing• no query-to-code translation
Introduction tutorial to the DHIS2 API	<ul style="list-style-type: none">• limited opportunities for learn-by-doing• complex data model• complicated terminology

Table 5.1: Summary of artifacts responding to challenges.

5.1.1 API testing tool

Figure 5.1 shows the front page of the DHIS2 app course as described in subchapter 3.1.4. The DHIS2 app course featured a range of modules with different topics that users could select to access the respective tutorials. As part of this research project, an “API testing tool” was designed and added to the DHIS2 app course, which can be seen at the bottom of Figure 5.1. This newly integrated resource aimed to support the diagnosed challenge of “Limited opportunities for learning by doing.”

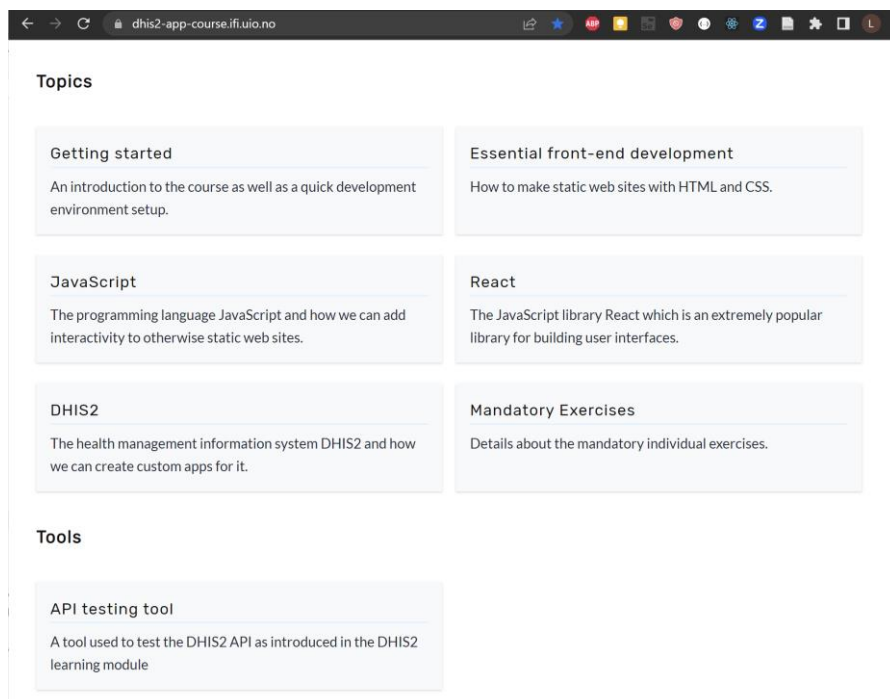


Figure 5.1: DHIS2 app course frontpage, including the API testing tool

By clicking the API testing tool module on the front page, the user is redirected to a page featuring the tool. As illustrated in Figure 5.2, this page provided an environment for users to experiment with the DHIS2 API using the testing tool. The tool offers predefined API endpoints (an endpoint being a specific URL where the API can be accessed) displayed on the left side of the page, as shown in Figure 5.2. By only offering users the relevant endpoints for taking the course, this aimed to reduce the complexity associated with introducing the API. This predefined set of available endpoints also ruled out the possibility of requesting invalid endpoints. These design decisions were taken to prevent students, and especially those with limited experience, from being overwhelmed by the vast amount of endpoints the API provides.

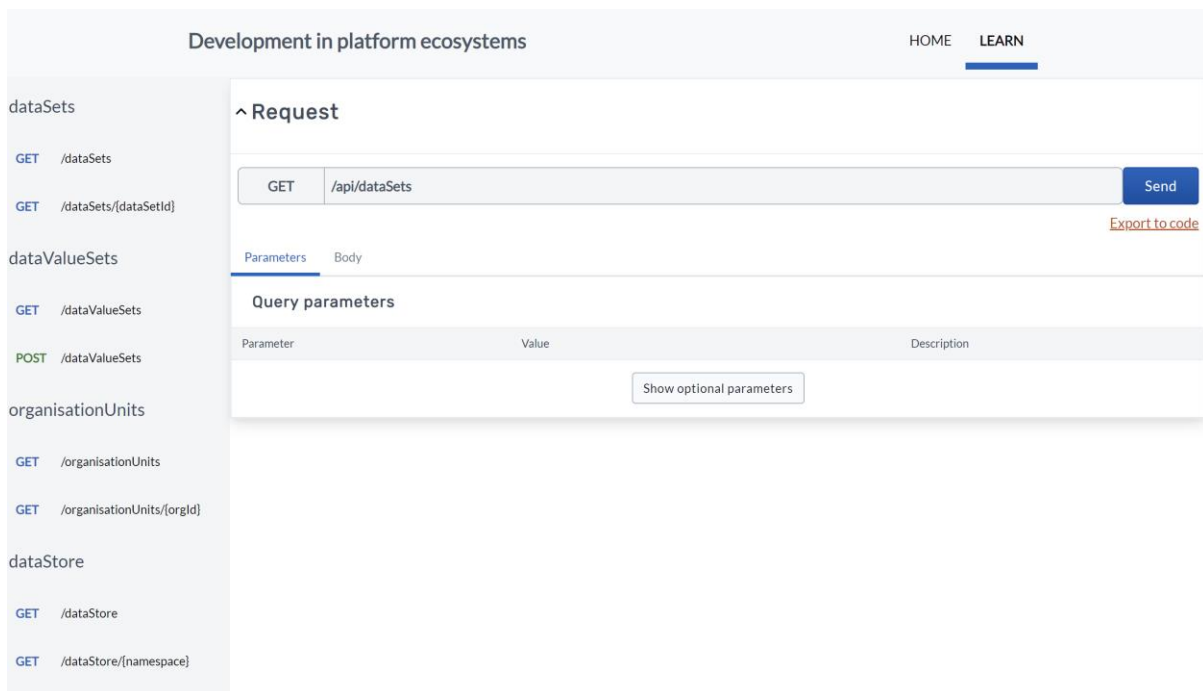


Figure 5.2: The API testing tool page

Once the user selected an API endpoint, they could proceed to fill out a form with parameters that the chosen endpoint supported, as shown in Figure 5.3. The tool only displays parameters supported by the endpoint, guiding the user and preventing input of invalid parameters. Each listed parameter describes its primitive type (for example, number), an example of input, and a more detailed description of the parameter. This detailed description includes a “more” link which redirects the user to a more detailed description in the DHIS2 documentation.

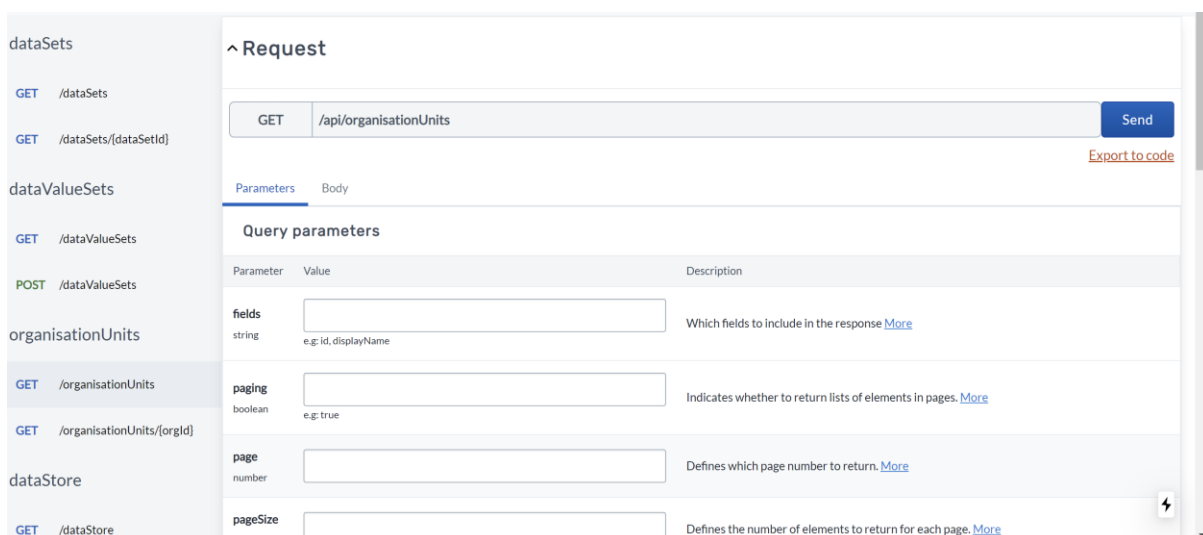


Figure 5.3: API testing tool showing available parameters.

The tool only displays the required parameters for an API endpoint by default, as these are necessary for the primary usage of the API. Optional parameters, typically needed only for advanced usage, are listed below the required ones but hidden by default. As shown in Figure 5.4, the user can view the optional parameters by clicking the “Show optional parameters” button. This design choice of optional parameters by hiding optional parameters underscored the usage of the fundamental aspects of using the API, making it more accessible for new users.

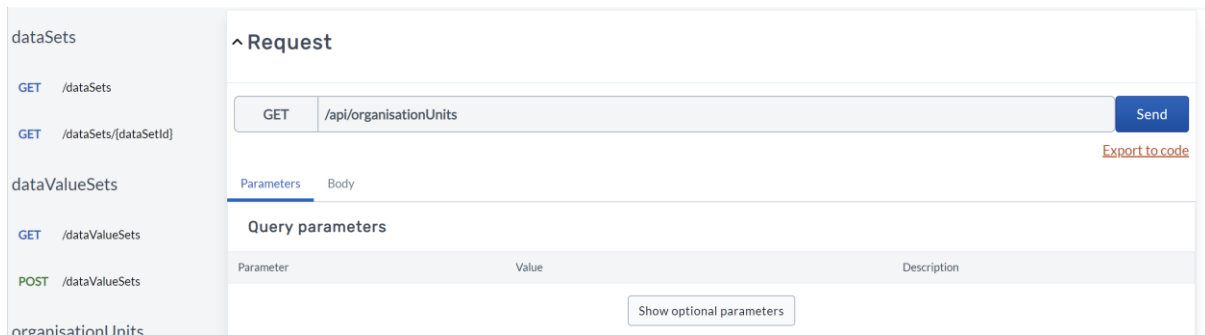


Figure 5.4: Option in API testing tool to show optional parameters.

The diagnosis revealed that the students encountered challenges when using DHIS2’s “Data Query Playground” for testing the API. Issues such as badly formatted input or missing parameters were difficult for the users to recognize and solve—the API testing tool aimed to support these challenges by providing better error handling and reporting. To help users detect errors, the tool displays error messages in a red box, as shown in Figure 5.5. This error message made it easy for users to see when something was incorrect. The tool conducts basic checks, such as detecting missing parameters, and highlights these errors to the user.

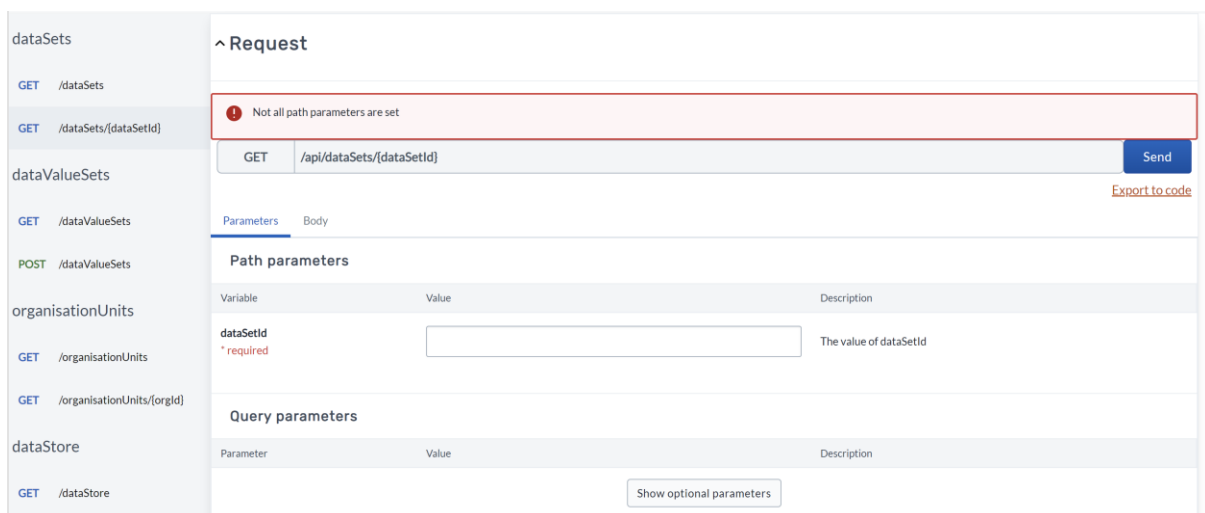


Figure 5.5: Example of an API error in the API testing tool

The API testing tool also supports attaching a body to the HTTP request. Some API requests required including a ‘body’ – additionally data sent along with the request. In the case of the DHIS2 API, this body needed to be in JSON format. To facilitate this requirement, the tool assists users in writing correctly formatted JSON messages. As shown in Figure 5.6, the tool provides an input field with syntax highlighting and syntax checks where the user can easily detect if the JSON body is incorrectly formatted. The input field with syntax formatting and syntax checks assisted users in creating valid API requests and provided specific error feedback.

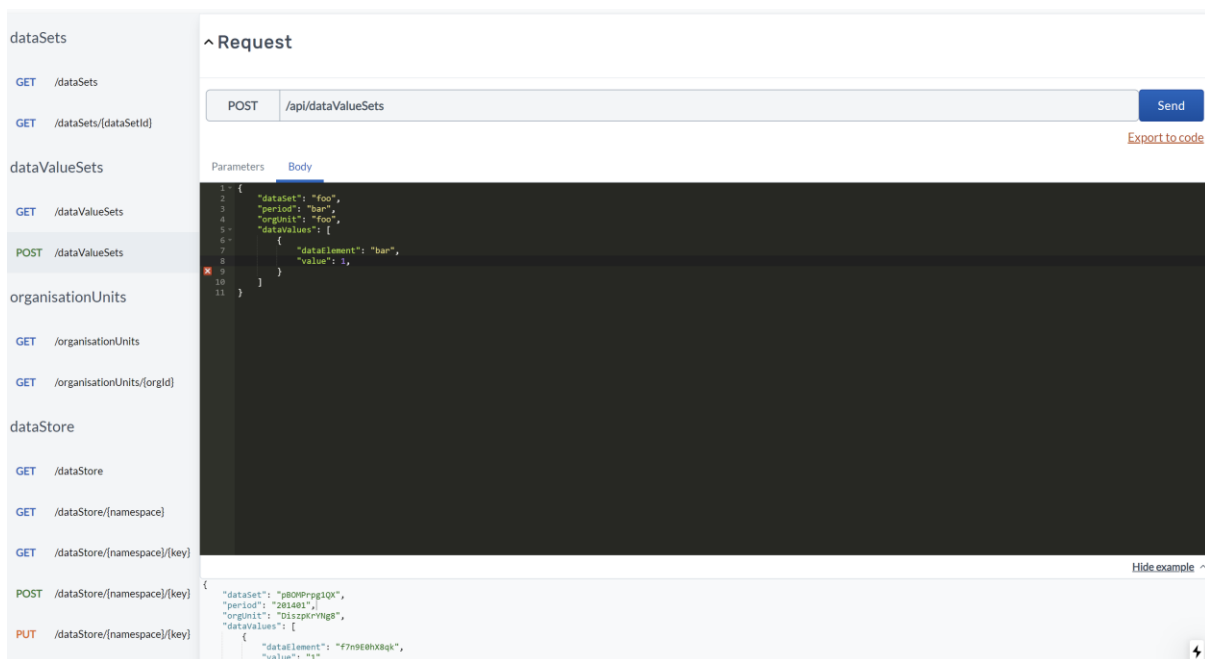


Figure 5.6: Input field in the API testing tool with syntax highlighting and syntax checks

When the user was ready to test the API request, they could initiate it by clicking the “Send” button. This button converts all the populated parameters and the body content into a query sent to the DHIS2 API. Subsequently, a “response” tab is displayed beneath the response, displaying the response from the API, as shown in Figure 5.7. The response tab displays the converted request sent to the API, an HTTP response code indicating whether the request succeeded, and the response body. The response body has color-coded syntax and indentation, simplifying the process of reading the response for the user.

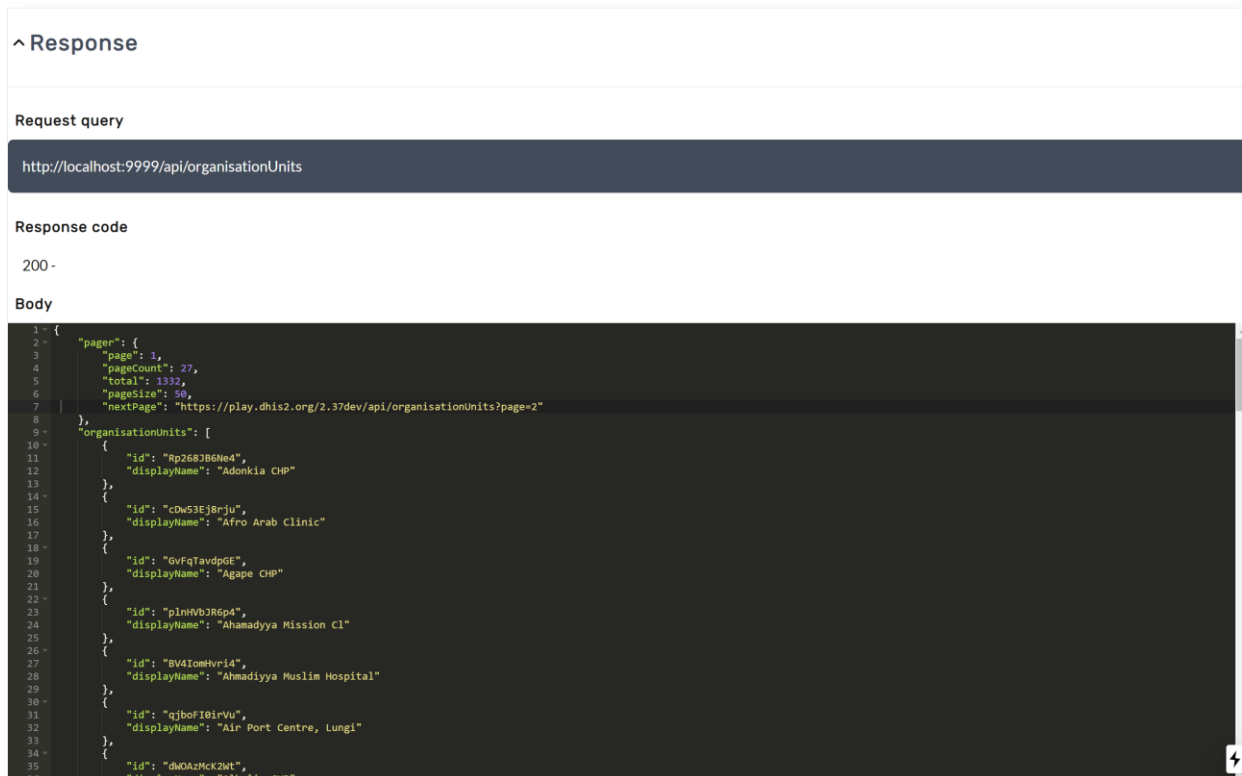


Figure 5.7: API testing tool showing the response from the DHIS2 API

When the user had sent a request, and the API responded with the expected results, the user could turn this API request into code by clicking the “Export to code” button. This feature aimed to address the diagnosed challenge of “No query-to-code translation,” where students wanted functionality for transforming an API request to code. When clicking the “export to code” button, the API testing tool transforms all inputs from the tool, such as parameters, into a code snippet. The user could easily copy this code snippet and use it in their code project. As shown in Figure 5.8, an API request is converted to a code snippet, including explanatory comments supporting how to use this snippet.

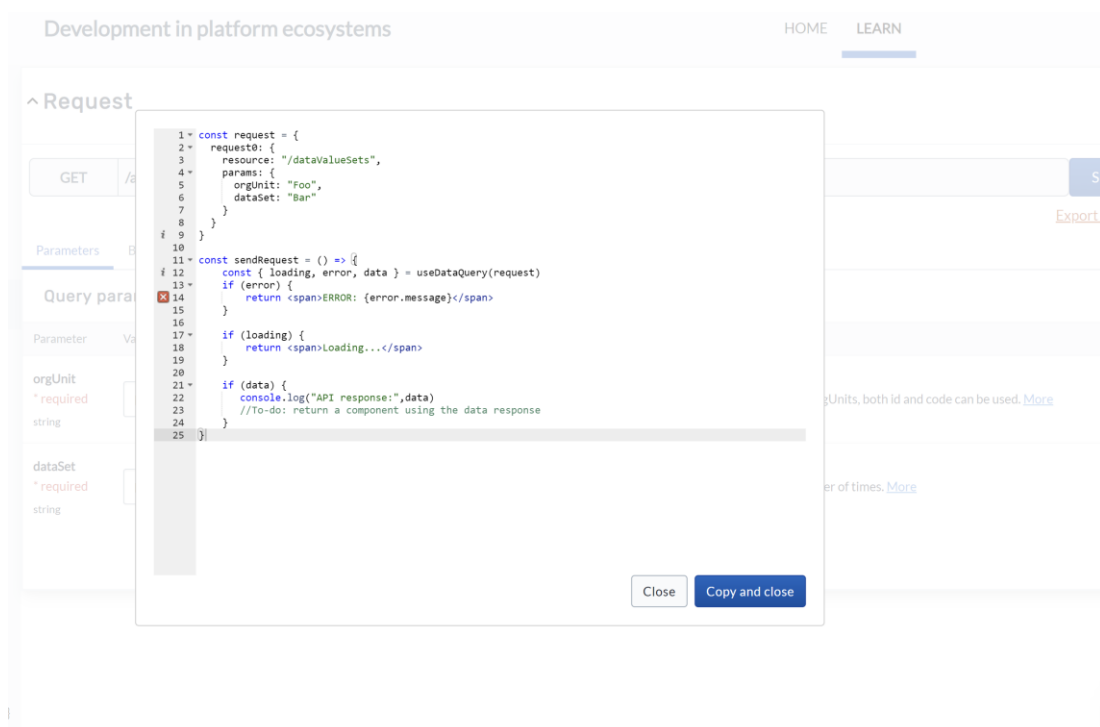


Figure 5.8: API request converted to a code snippet

In summary, the API testing tool supported the diagnosed challenges of “Limited opportunities for learn-by-doing” and “No query-to-code translation.” It supports the first challenge by allowing users to experiment with the API, enabling them to alter and send API requests without rebuilding their project or using external programs. The API testing tool simplified the usage of the API by offering guidance in available endpoints and parameters and minimizing potential user errors, thus reducing usage complexity. This approach could be particularly useful for new users of the API. To address the “No query-to-code translation,” the tool featured an “Export to code” function. This function transforms the API request into a code snippet that can easily be copied and incorporated in own code projects.

5.1.2 Introduction tutorial to the DHIS2 API

In response to the diagnosed challenges of the “Complex data model” and “Complicated terminology” associated with learning the DHIS2 API, the DHIS2 app course content was adapted. Particularly, the course module “DHIS2,” as shown in Figure 5.9, was adapted. This module teaches students how to develop applications for the DHIS2 platform, including an introduction to the APIs. Modifications were made to the content of this module, addressing the diagnosed challenges. Additionally, the module was adapted to be interactive, which addressed the challenge of “limited opportunities for learning by doing.”

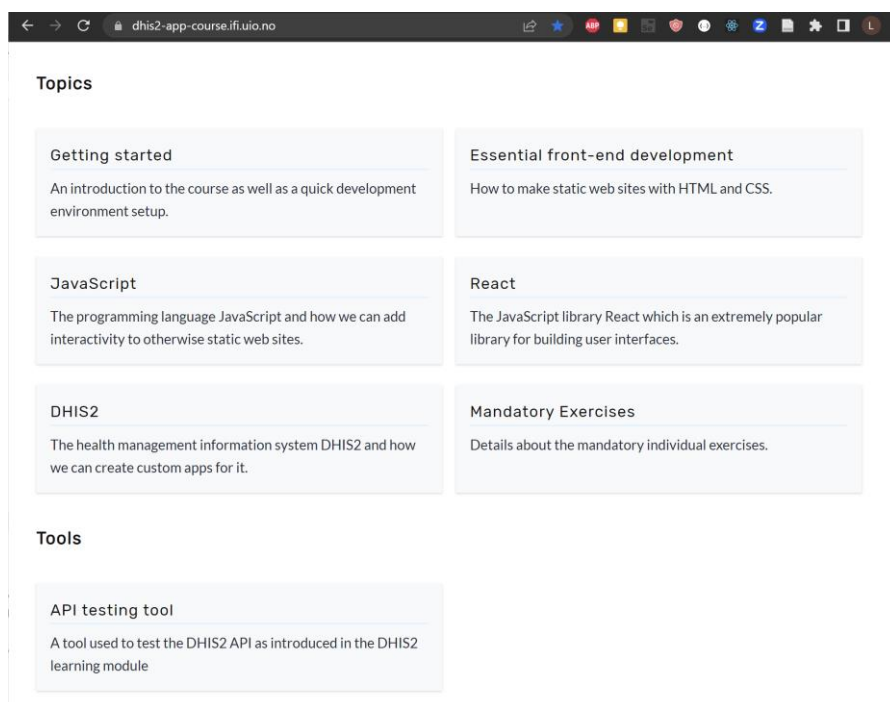


Figure 5.9: DHIS2 app course front page, including the DHIS2 module

Within the DHIS2 module, students were first introduced to the DHIS2 data model before delving into API usage. Understanding the data model and how different elements in the data model are related is crucial for comprehending the API’s architecture and using it. To facilitate learning of the data model, the new design of this module introduced a scenario involving a person administering antibiotics. After introducing the scenario, the data model was presented, with each element being mapped to the antibiotic’s scenario. Throughout the introduction of the data model, illustrations, such as shown in Figure 5.10, and text was emphasized to support the understanding of how the data model related to real-life examples.

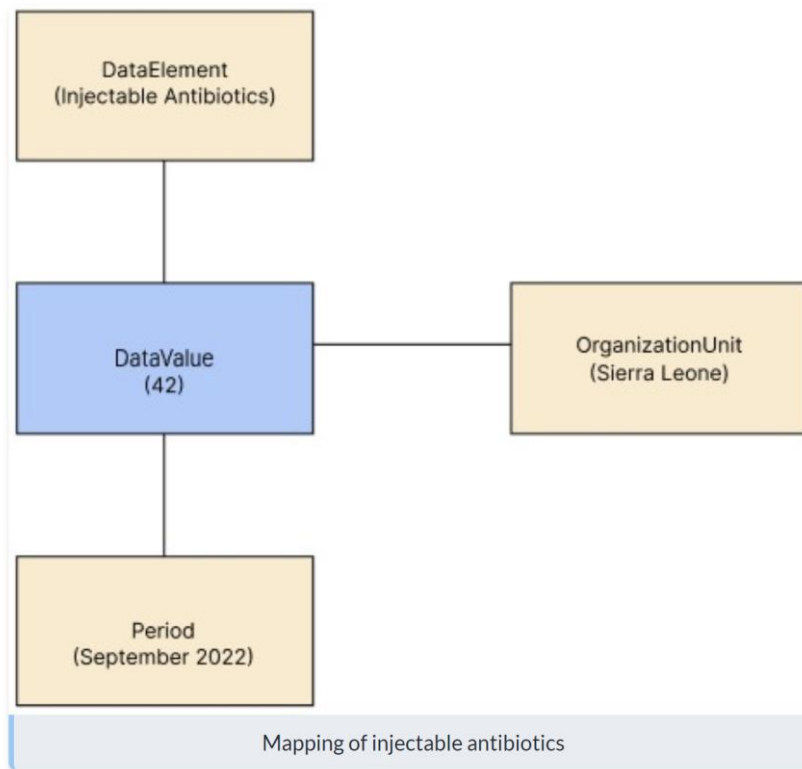


Figure 5.10: The data model is mapped to a real-life scenario

After learning about the data model, the module introduces students to how to use the DHIS2 API. This introduction was adapted to be interactive. The new design of the introduction mainly consisted of three segments: first, an *explanation* was given about the usage of the API, followed by the API testing tool as introduced earlier, where students could *try out* the API, and eventually, a *task* where students had to use the tool to complete the task. Figure 5.11 shows a part of the designed tutorial for introducing the DHIS2 API, including how the three segments apply to the design. This interactive approach aimed to support the diagnosed challenge of “limited opportunities for learning by doing” by enabling students to learn while making API requests.

Test query 2 (Collecting dataElement)

Let's run our second API query which should access the 'Life-saving Commodities' dataSet and find it's dataElements.

In test query 2 shown below, you will see that `{dataSetId}` has been added to the path. This is a variable which you will set by entering the dataSetId value in **Path parameters** below.

^ Request

GET
/api/dataSets/{dataSetId}
Send

[Export to code](#)

Parameters
Body

Path parameters

Variable	Value	Description
dataSetId <small>* required</small>	<input style="width: 80%;" type="text"/>	The value of dataSetId

Query parameters

Parameter	Value	Description
<div style="border: 1px solid #ccc; padding: 5px; display: inline-block;">Show optional parameters</div>		

Test query 2

Task

Paste the Life-saving Commodities id you copied from Test query 1 into the dataSetId path parameter of Test query 2. Then click send.

Explanation

Try out

Task

Figure 5.11: A part of the introduction to the DHIS2 API

5.2 Chapter Summary

This chapter addressed the design of two artifacts, an API testing tool and an introduction tutorial to the DHIS2 API, aiming to address the identified API learning challenges. The API testing tool addresses the "limited opportunities for learn-by-doing" and "no query-to-code translation" challenges by providing a simple, user-friendly environment to experiment with the API and transform the request into code snippets. The introduction tutorial to the DHIS2 API addresses the challenges of the "complex data model" and "complicated terminology" by offering an interactive, scenario-based learning experience to familiarize with the DHIS2 data model, the terminology in this data model, and API usage. Additionally, the tutorial addressed the "Limited opportunities for learn-by-doing" challenge as the tutorial was interactive.

6 Findings from Evaluation

This chapter will go through the evaluation phase of the data collection to evaluate how the designed artifacts responded to the diagnosed challenges in the previous chapters. Evaluating the designed artifacts can determine how the artifacts support complementors in learning the DHIS2, which in return can contribute to answering the research question of how to design resources for supporting complementors. This chapter will begin by presenting an evaluation of the API testing tool, followed by an evaluation of the introductory tutorial to the DHIS2 API.

6.1 API testing tool

6.1.1 Diagnosed challenge: Limited opportunities for learning by doing

The diagnosed challenge of “limited opportunities for learning by doing” was addressed by designing the API testing tool and using it in the introduction tutorial to the DHIS2 API. This facilitates hands-on learning by allowing users to quickly test and validate their API understanding. The learn-by-doing approach offers an alternative strategy to just reading about the API, which multiple students found helpful when learning to use the API. Several students empathized how the learn-by-doing approach and the API testing tool supported learning the DHIS2 API. As one student expressed: *“Interactivity is basically how I learn. [...] To see the changes in practice without just having to read a text is very useful. I like seeing myself how change to a parameter changes the response.”*

In the diagnosis phase, students were observed employing a brute-forcing strategy for learning the API. This involved repeatedly making minor changes and then testing them. The API testing tool was designed to support and facilitate this iterative learning approach by providing an environment where students could easily experiment with sending requests and examining the response. Several students found this trial-and-error method helpful, as one remarked: *«If I had difficulties with a query, I just did more testing. When things got more difficult, it only required you to do more testing”*.

The API testing tool supported the learning process for beginners by offering predefined endpoints and parameters. This was useful for students without experience working with the API, as searching for available parameters and endpoints was unnecessary. Additionally, the

tool supported students by preventing them from making common errors, a feature that many students found beneficial. One of the students detailed the experience as follows:

“Most importantly, the tool helps understanding what the API does. The tool is easy to navigate. You have one explicit field for each value. [...] Whenever I tried exploring through the web browser, I often made mistakes and retrieved incorrect results. In addition, I could not remember how exactly to write query strings. So having specific fields (in the API testing tool), assisting you in what to write, helped me.”

During the diagnosis stage, it was observed that students used a variety of approaches for exploring the API. Notably, some students tested the API by writing, editing, and running code in their applications. This method was shown to be inefficient. Thus, a suggestion was to enable students to easily adapt and test without doing this by code. The API testing tool responds to this suggestion, providing a more efficient way for students to interact directly with the API. One of the students expressed how this was useful: *“It was much more efficient being able to retrieve something from the API before digging into how to do it in the code.”*

The clarity and readability of responses provided by the API testing tool were recognized as important for students’ understanding of the DHIS2 API. As one student expressed, *“First and foremost, the response which helped me understand the API. [...] The response has a very nice overview, comes in a hierarchy and is colored.”* Another student underlines this: *“The API testing tool gives a good overview to quickly let you retrieve data and view that you have retrieved correct information.”* Given the numerous test of the API conducted by students, the ability to quickly interpret these responses for evaluating the API’s response was identified as important.

6.1.2 Diagnosed challenge: No query-to-code translation

One of the main difficulties students encountered was converting an API request into a DHIS2-specific syntax so that it could be used in code. The API testing tool incorporated a “convert to code” functionality in response to this challenge. This allows user to easily export their API queries as code snippets that can be directly integrated into code. By offering this functionality, the tool eliminates some of the complexity and confusion associated with the manual conversion of API requests. Many students expressed this export to code as a helpful feature. One student expressed, *“I like the export to code function. It helped me understand how to use*

it in my code”. As a result, students could experiment with the API using the tool, and when they got the desired result, apply the API request to their code.

Some students also found the export-to-code functionality helpful as it offers a code example of how API requests were converted to the DHIS2-specific syntax. Instead of copying and pasting the exported code into the code base, some students used it as a code example to learn how to write an API request by code. One of the students describes this during the interview: *«Export to code shows how you code it basically. Based on this code, I could conclude how I can do it with other queries. So you could you just swap the values from this example code. It helped me a lot to understand how to code the query»*

In summary, the API testing tool efficiently addressed the challenge of converting API queries through code, which was no longer a widespread issue among students. Through observations and conversations with students, it was discovered that many students found this helpful. There were no reports of difficulties related to using API requests in code.

6.2 Introduction tutorial to the DHIS2 API

6.2.1 Diagnosed challenges: Complicated terminology and complex data model

A challenge identified in the diagnosis was that students encountered difficulties comprehending the data model due to its abstraction layers and terminology. This thesis found similar challenges: there's a high usage of words likely not to be understood by new complementors. The DHIS2 app course was designed to support the challenge of complicated terminology by exemplifying a scenario and relating the data model to real-world examples. Using illustrations to map this scenario to the data model and terminology used in the data model was meant to support the students' understanding of terminology. However, based on student feedback, the terminology was still a concern. Terminology being used, such as "dataElement," had low meaning for the students, meaning it was difficult to grasp the concept of what a dataElement is and what it should be used for. One of the students expressed this concern: *"For instance, dataset and dataSetValue were confusing. In the beginning, I did not have a clue what a dataElement could be"*.

Many of the presented words had many similarities, such as "DataElement," "DataValue," "DataSets," "DataSet," and "DataValueSets." With the students being presented with various confusing terms, an additional challenge was that these presented terms had similarities, making it difficult to distinguish one from another. One student articulated this confusion by saying, *"DataElement, dataValue, dataModel, and DataSet, they all look similar to me."* Additionally, all the terms shared the common trait of containing the word "Data," which offers little insight into their specific meanings. Another student expressed confusion regarding this data terminology by saying, *"DataSet and Datavalue. There's a lot of 'data data.'" These were factors that confused students understanding and differentiating the terminology.*

The terminology students mentioned as difficult was often associated with the terminology used in the data model. Not understanding this terminology was a concern as it's crucial for using the API. There were relationships between the different elements in the data model, which made it important to understand these associations and their relevance to each element. As one student expressed, *"I was worried about being able to understand the data model initially. It was a lot of different names and elements to be handled simultaneously."*

Overall, learning the data model presented several challenges: firstly, understanding the different terms was an obstacle. Secondly, the students were found to have difficulties

differentiating between similar terms. And finally, as the data model had many relationships between the elements, it was essential to understand how the different terminology was related.

In response to overcoming the challenge of complicated terminology and complex data model, students suggested several strategies for adapting the DHIS2 app course content to assist in building a better understanding. This included adding more visual aids such as diagrams, illustrations, and maps to explain terminology and the data model. Students in the interviews expressed that a clear structure and visualization of how different components are linked and related is very helpful. These illustrations should ideally contain relatable examples of what the element can represent. I.e., the DHIS2 app course used the medication “oxycontin” as an example, yet some students had difficulties understanding the example due to being unfamiliar with the medication. Figure 6.1 shows an illustration in the tutorial, which students did not find helpful as it lacked a relatable example. Figure 6.2 shows a tutorial illustration with a relatable example that was more helpful for students.

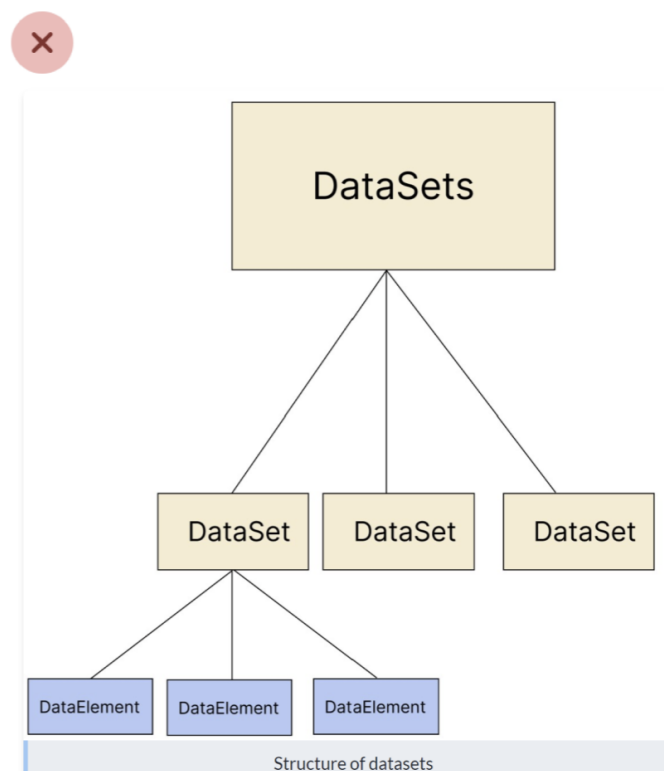


Figure 6.1: Illustration in tutorial with no relatable example

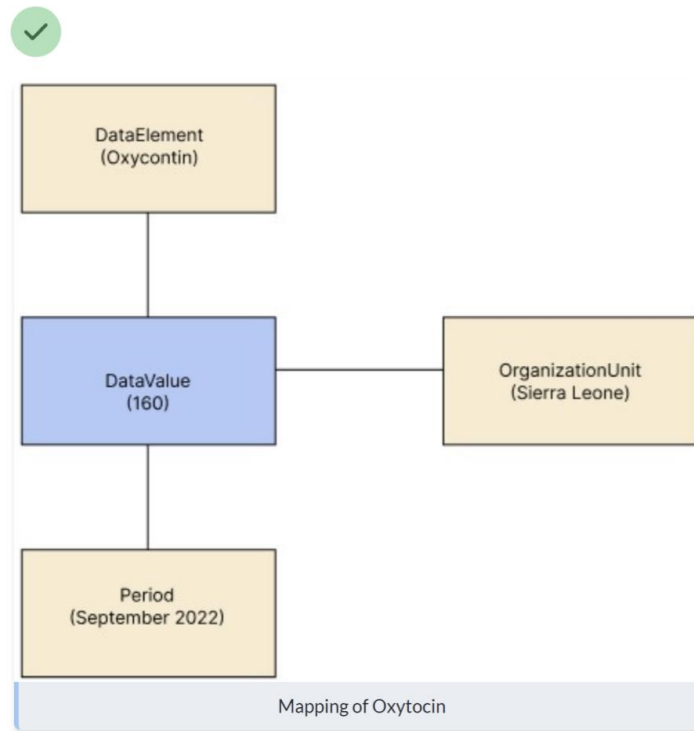


Figure 6.2: Illustration in tutorial with a relatable example

Besides maps and illustrations for learning the terminology and data model, several students reported that practicing using these terms helped build a better understanding of them. The API testing tool offered this functionality, allowing students to run API queries meanwhile learning the terminology and its interrelations in the data model. As one student mentioned, *“I learned the data model by using the API testing. Just editing fields and viewing how it affected the results helped.”* Being able to immediately test and verify understanding of the terminology and data model was found to be a helpful strategy for several students

Students liked the step-by-step tutorial provided for learning the API. This tutorial, which included tasks, gave students a specific purpose for exploring the API. Furthermore, the tutorials included answers to these tasks allowing students to compare their work to the expected result. As one student noted, *“The walk-through with the API testing tool where you had to complete a task and use the result in completing the next task gave you a better understanding of the terminology”*. This interactive approach made the DHIS2 app course more engaging and hands-on, supporting the learning experience.

6.3 Chapter Summary

The evaluation of the designed artifacts that addressed the challenges identified in the diagnosis was presented in this chapter. Specifically, the API testing tool and the introductory tutorial to the DHIS2 API were evaluated.

Firstly, the API testing tool was evaluated. It did address the challenge of ‘limited opportunities for learning by doing’ by offering an environment for hands-on exploration and experimentation, which many students found useful. Moreover, it supported the challenge of ‘no query-to-code translation’ by incorporating a feature that allows users to easily convert their API requests into code snippets.

Second, the introduction tutorial to the DHIS2 API was evaluated. This evaluation showed mixed results. The tutorial was designed to address the ‘complicated terminology’ and ‘complex data model’ challenges. While it did provide some clarity through examples, illustrations, and interactivity, students still experienced confusion around the terminology and the data model. This tutorial also addressed the issue of ‘limited opportunities for learning by doing’ as it was interactive, enabling students to experiment with the API while learning it. Many students found this interactive approach helpful in learning the API.

7 Design considerations

This chapter will outline four design considerations based on findings from the diagnosis, design and demonstration, and evaluation. These design considerations can serve as valuable guidance for platform owners when designing broadcasting KBRs to support new complementors learning the platform’s API. Table 7.1 summarizes these design considerations with a short description to provide a clear overview. Following the table, each design consideration will be discussed in greater detail, highlighting their importance and offering practical guidance for their implementation.

Design consideration	Description
Offering a simple testing tool	The platform owner can provide a tool that allows complementors to easily test, verify, and explore the platform’s APIs, enabling them to interact with the API directly without writing code or using third-party tools. This testing tool supports the learning process of new developers by facilitating practical usage of the API, enabling fast iterations of testing the API. To facilitate supporting of new complementors, the tool should ensure simplicity by reducing the complexity of usage, such as having predefined parameters and endpoints.
Facilitate exporting to code	The platform owner can complement the testing tool with functionality for converting API requests made in the testing tool into a code snippet usable in the platform’s supported programming languages. This functionality aims to bridge the gap between testing the API and its practical applications, making it easier for complementors to implement the API requests within their code.
Provide interactive tutorials	The platform owner can offer tutorials combined with hands-on activities that enable complementors to explore the API and apply knowledge while learning about the API. These tutorials introduce learners to the API's practical use, where immediate feedback from interactive resources supports the learning process. Platform owners making these interactive tutorials should combine tutorials with hands-on activities, including problem-specific descriptions, and include tools where learners can test out what is described in the tutorial.
Support esoteric terminology comprehension	The platform owner can provide resources supporting complementors to better understand the esoteric terminology used in the platform’s API. Esoteric poses a significant learning barrier for new complementors, leading to confusion and difficulties using the APIs. Thus, platform owners need to design resources supporting the comprehension of these terms. These resources can include such as tutorials mapping esoteric terms to familiar terms and providing illustrations explaining the terms.

Table 7.1: Design considerations

7.1 Offering a simple testing tool

The platform owner can provide a tool that allows complementors to easily test, verify, and explore the platform’s APIs, enabling them to interact with the API directly without writing code or using third-party tools. The findings of this thesis indicate that deploying a simple testing tool supported complementors learning to use the platform’s API. Complementors have assumptions about how the API works based on tutorials and other documentation related to the API. By offering an option where complementors can easily test and verify assumptions about the API, platform owners can facilitate the learning process of new developers. A testing tool enables a different approach to learning the API through interaction and exploring its content, complementing other learning resources. Such a tool is particularly efficient when combined with API documentation due to enabling testing of assumptions and mapping documentation to practical usage. Figure 7.1 illustrates an example of a testing tool as designed in this research project.

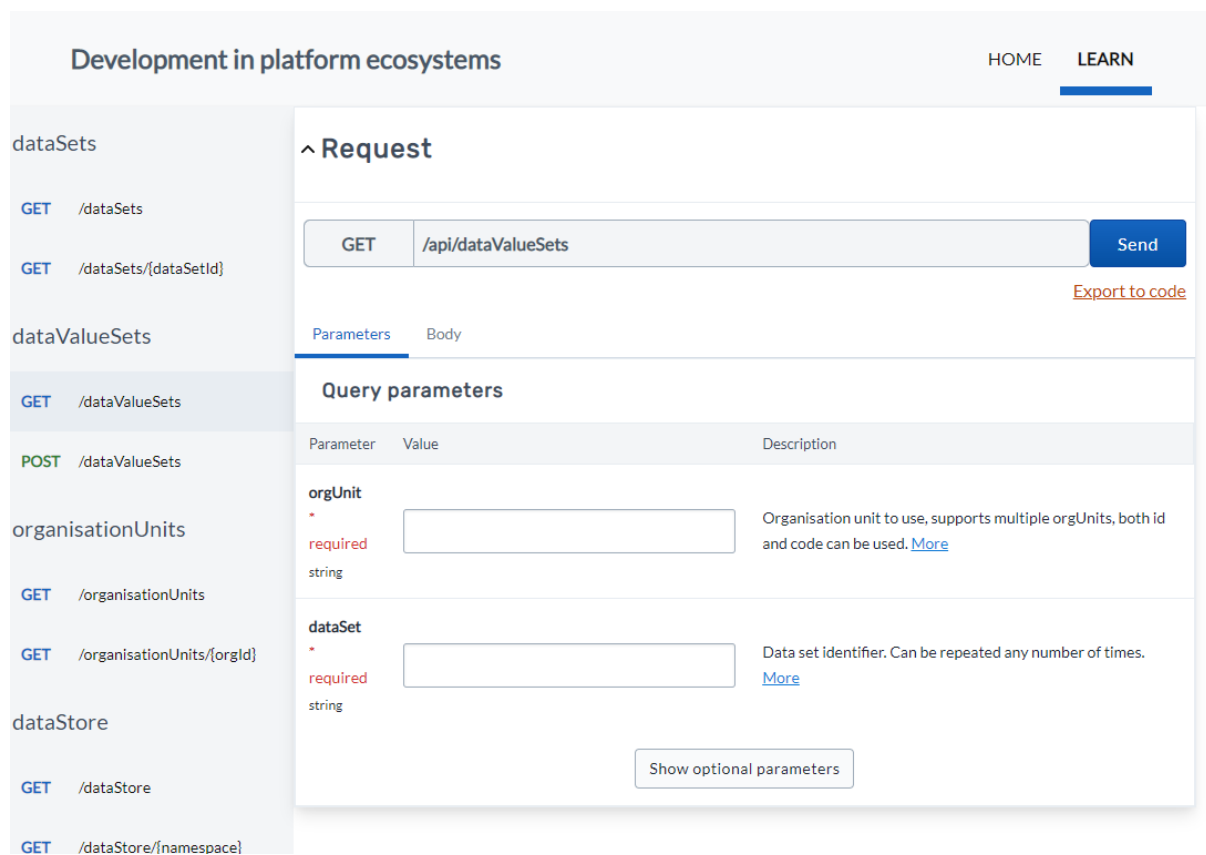


Figure 7.1: Example of a simple testing tool as designed in this research project

Complementors use the testing tool to explore the API by editing requests, sending them, and receiving responses. Through this procedure, complementors become familiar with various aspects of learning the API, such as how to use it, the structure of the API, relationships

between different parts, and relevant terminology. By deploying a testing tool, the platform owner can facilitate complementors engaging in this procedure iteratively, allowing them to repeatedly edit, test and receive feedback on API requests.

Providing a testing tool enables platform owners to decide functionality supported and tailor it to meet the needs of their complementors. For onboarding new complementors, it's essential to consider their needs when designing the tool. To support their learning of using the API, the tool should be designed with simplicity in mind. This can be achieved by reducing its complexity, such as only displaying available parameters and presenting only relevant endpoints. As found in this thesis, offering complementors a limited subset of API endpoints is more suitable for not overwhelming the complementors with all supported functionality of the API. To assist complementors in making the API requests, input fields for parameters should include an example and only accept the required format, i.e., a number. In the event of an error, the testing tool must display clear and easily understandable error messages.

The response section of the testing tool requires particular attention as complementors rely on understanding its contents to benefit from the tool. Platform owners should emphasize designing this well-formatted, including indenting and color-coding of syntax to ease the complementors interpretation of result messages. Figure 7.2 illustrates an API request made using a web browser, as some students used. The response is displayed below the search bar, and it is apparent that the response is just a bundle of unformatted text, making it difficult for users to interpret the response.

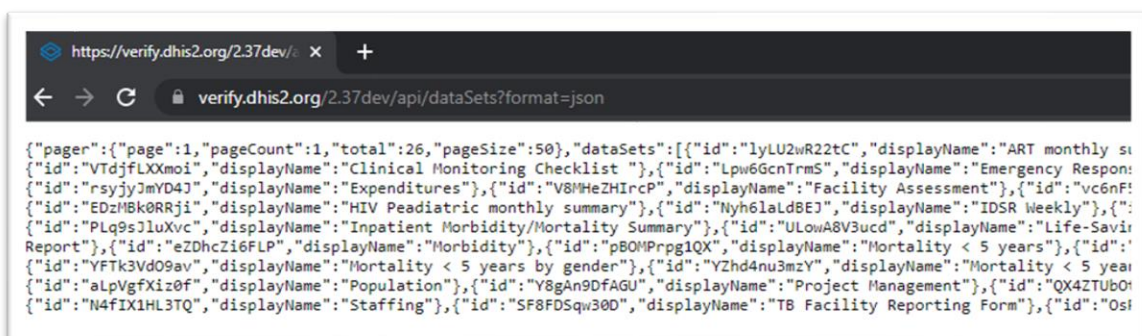


Figure 7.2: An API request where response is not formatted.

As shown in Figure 7.3, the testing tool represents a more user-friendly display of the API request results. The syntax is highlighted with colors, such as green for property names, and the text is indented. This representation makes it easier for complementors to interpret the information, which is essential for understanding the API.

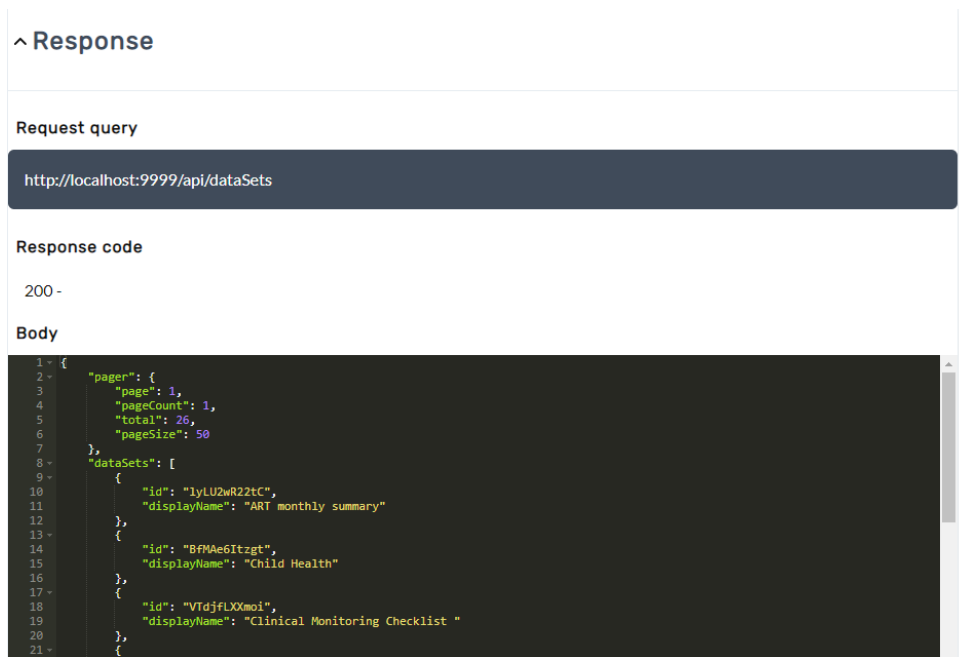


Figure 7.3: An API request where the response is well formatted

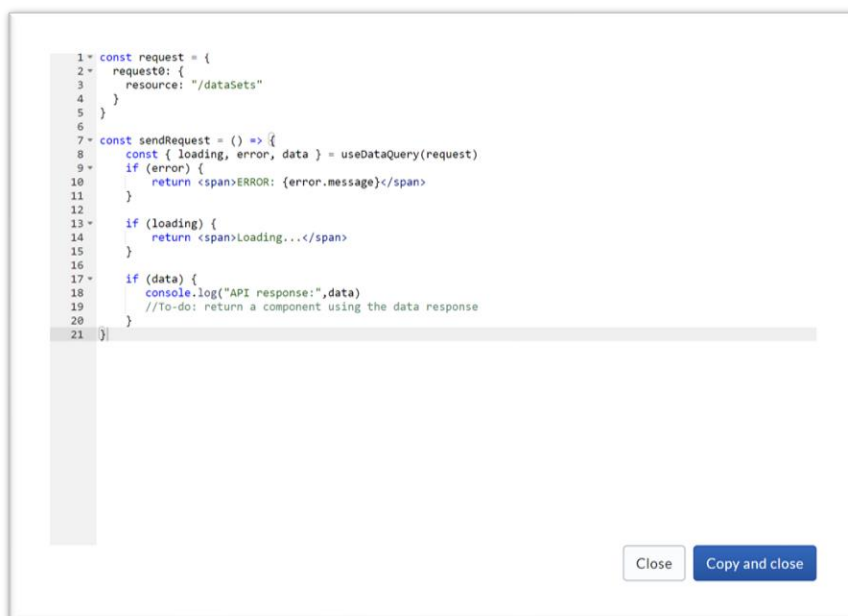
Another benefit of provisioning a testing tool is that it allows the complementors to engage and explore the API directly without writing code or using a third-party tool to interact with it. This lowers the boundary for testing and exploring the API, especially for new complementors who may not be familiar with other tools for interacting with the API. Additionally, not having to write code for testing the API significantly increases the speed for complementors to test and explore the API. Lastly, providing a testing tool prevents the risk of coding errors when exploring the API. Not having to write code ensures complementors that occurred errors are related to the API requests, not coding errors.

In summary, a testing tool supplements the broadcasting KBR for supporting new complementors in learning to use the API. A testing tool allows complementors to verify their assumptions about the API, facilitating their learning process. The tool grants possibilities of fast iterations of testing, easy-to-understand responses, and an interactive learning environment that supports the desired learn-by-doing strategy. Platform owners providing such tools can tailor them to meet the complementors' needs, such as simplicity for supporting new complementors. The tool enables complementors to engage and explore the API directly without writing code or using third-party tools, reducing barriers to testing the API and potential errors using other tools.

7.2 Facilitate exporting to code

The platform owner can complement the testing tool with functionality for converting API requests made in the testing tool into a code snippet usable in the platform's supported programming languages. Findings in this thesis indicate that platform owners should include this export-to-code functionality when designing a testing tool. This allows complementors to directly apply the knowledge they acquired during the exploration of the API to a code snippet that can be used in their code. Figure 7.4 shows an example of exporting to code using the testing tool. Exporting the queries to code bridges the gap between learning to use the API and its practical applications.

Platforms may require a platform-specific approach for sending queries to the API. For example, this thesis found that DHIS2 libraries required a custom syntax for sending queries to the API. As complementors had difficulties writing such DHIS2-specific syntax, it was beneficial to have an export-to-code functionality to support practical usage of the API request. Complementors used this export code by applying it directly to the code. However, as found in this thesis, several complementors also used the export-to-code functionality as a code example for learning how to write API requests by code.



```
1 const request = {
2   request: {
3     resource: "/dataSets"
4   }
5 }
6
7 const sendRequest = () => {
8   const { loading, error, data } = useDataQuery(request)
9   if (error) {
10    return <span>ERROR: {error.message}</span>
11  }
12
13  if (loading) {
14    return <span>Loading...</span>
15  }
16
17  if (data) {
18    console.log("API response:", data)
19    //To-do: return a component using the data response
20  }
21 }
```

Figure 7.4: Example of a code example produced using the export-to-code function

7.3 Provide interactive tutorials

The platform owner can offer tutorials combined with hands-on activities that enable complementors to explore the API and apply knowledge while learning about the API. The findings of this thesis show that interactive tutorials support the understanding of complementors learning to use the platform API. The interactive tutorials combine tutorials with interactive activities of sending API requests enabling complementors to explore the API and apply their knowledge during the tutorial. This type of tutorial promotes an engaging learning approach where complementors learn new API concepts before interacting with the API and seeing how it applies in practice, promoting the understanding of API structure. To support new complementors in understanding the API, platform owners should provide some tutorials with limited complexity. Figure 7.5 illustrates an example of an interactive tutorial as designed in this research project.

Test query 2 (Collecting dataElement)

Let's run our second API query which should access the 'Life-saving Commodities' dataSet and find it's dataElements.

In test query 2 shown below, you will see that `{dataSetId}` has been added to the path. This is a variable which you will set by entering the dataSetId value in **Path parameters** below.

Request

GET /api/dataSets/{dataSetId} [Send](#) [Export to code](#)

Parameters Body

Path parameters

Variable	Value	Description
dataSetId *required	<input type="text"/>	The value of dataSetId

Query parameters

Parameter	Value	Description
-----------	-------	-------------

[Show optional parameters](#)

Test query 2

Task

Paste the Life-saving Commodities id you copied from Test query 1 into the dataSetId path parameter of Test query 2. Then click send.

Figure 7.5: A part of the interactive tutorial as designed in this research project

A benefit of interactive tutorials is the ability to provide descriptions and guidance within the context of hands-on activities. As suggested in the design consideration of provisioning a test tool, such a tool is particularly efficient when combined with API documentation. The tutorial introduces new concepts where a testing tool enables immediate testing of such described concepts. Platform owners can use such tutorials to demonstrate how the API should be used. Descriptions in the tutorials should be problem-specific and elaborate on how to use the API for solving practical tasks, together with a detailed description of the expected results of the specific task. Overall, using interactive tutorials balances the text-heavy nature of traditional tutorials with practical and experimental learning, which is more engaging and helpful for users.

The interactive tutorials should preferably have an embedded tool within the learning environment, allowing the complementors to use this tool instead of relying on external tools. Such a tool can be adapted to align with the tutorial content by, i.e., validating if the request is correct according to the tutorial task. If the tutorial requires setting up an external API testing software such as “Postman,” this may cause a barrier for new complementors unfamiliar with such tools. Providing an embedded, easy-to-use tool lowers the threshold for new complementors to explore and test the API.

7.4 Support esoteric terminology comprehension

The platform owner can provide resources supporting complementors to better understand esoteric terminology being used in the platform's API. While designing broadcasting KBRs for an enterprise API, it's essential to consider that esoteric terminology such as "dataElements" or "dataSets" as shown in the findings of this thesis, represents a significant learning barrier for new complementors. Whenever introducing such terminology, documentation should dedicate additional resources to explaining these terms. This may include a mixture of text, illustrations, and visual representations carefully explaining the terms and their intents. The documentation should map the esoteric term to language familiar to the complementors. Avoid presenting multiple esoteric terms on the same documentation page, as this can easily confuse the various terms; instead, dedicate separate resources for different esoteric terminology. If esoteric terminology is related to other esoteric terminology, include an illustration demonstrating this. When possible, avoid using esoteric terminology in the documentation. Instead, platform owners should consider using language that is easy to understand and free of jargon. This approach can help reduce confusion and make the documentation more accessible to a broader range of complementors, including new complementors.

Enabling complementors to test the API using the terminology early in the learning process positively affected the understanding of esoteric terminology. Deploying a testing tool, as suggested in the first design consideration, can support complementors with understanding esoteric terminology. Platform owners should consider designing broadcasting KBRs allowing contributors to quickly access and try out terminology being introduced, helping contributors verify understanding and theories which they may have about the platform's terminology.

This design consideration also raises awareness of the confusion related to esoteric terminology, which platform owners should consider when designing the platform APIs. When designing the APIs and terminology, platform owners should ensure that esoteric terms are distinguishable, meaning their names are not very similar. The platform owner should strive to use names which is easy to understand, making it easier for complementors to understand and use the terminology correctly. This may reduce the provisioned KBRs related to terminology, as clear naming would cause less confusion in learning the platform APIs. Requirements for a high amount of provisioning of KBRs related to terminology could indicate that the terminology is esoteric and confusing for new complementors.

8 Discussion and Contributions

The thesis will now discuss how the findings of this thesis contribute to research about KBRs and their role in addressing the knowledge boundaries, specifically for new complementors. By presenting three contributions to research and a set of practical contributions, this thesis extends current knowledge on how platform owners can support large audiences of new complementors in learning to use the platform API. Before delving into the contributions, Table 8.1 is included to summarize them.

Contribution to	Contribution	Summary of contributions
Research	Extended knowledge of broadcasting API KBRs	In this research, resources were designed to support large audiences of new complementors learning to use platform APIs. These resources align with Foerderer et al. (2019)'s definition of broadcasting KBRs. The resources contribute to existing research on broadcasting KBRs by exploring how various types of KBRs can be designed to support new complementors in learning enterprise platform APIs. The thesis also addresses research calling out for interactive learning by deploying such sources (Cummaudo et al., 2022; Gao et al., 2020; Macvean, 2016)
	Describes hosted developer sandbox as a broadcasting KBR	Foerderer et al. (2019) propose several broadcasting KBRs, including a "hosted developer sandbox," however, there is no description of such broadcasting KBR. A "testing tool" was developed throughout this thesis, which can be considered a hosted developer sandbox. Using the testing tool as a foundation, this thesis extends Foerderer et al. (2019)'s research on KBRs by offering a description of a hosted developer sandbox, including what it is and how it supports complementors.
	Extends understanding of various approaches addressing knowledge boundaries	Drawing upon existing literature and empirical findings, this thesis highlights different levels of design in platforms for addressing knowledge boundaries, including <i>platform-level</i> design and <i>resource-level</i> design. This contributes to understanding how knowledge boundaries can be addressed in various ways (C. Baldwin & Woodard, 2008; Foerderer et al., 2019; Ghazawneh & Henfridsson, 2013).
Practice	Design considerations for platform owners	The thesis contributes to practice by providing four design considerations that can serve as valuable guidance for platform owners when designing broadcasting KBRs to support new complementors learning the platform's API, including (1) Offering a simple testing tool, (2) Facilitate exporting to code, (3) Provide interactive tutorials, and (4) Support esoteric terminology comprehension
	Extended resources	The thesis also contributes to practice by extending resources used by complementors of DHIS2 and students taking the course Development in Platform Ecosystems at the University of Oslo

Table 8.1: Summary of contributions

8.1 Contribution to research

The thesis contributes to the field of platform ecosystems (Tiwana, 2014) by exploring how platform owners can design resources supporting complementors learning to use the platform's boundary resources. Furthermore, it contributes to research on boundary resource usage (Ghazawneh & Henfridsson, 2013), particularly in the context of platform APIs as a boundary resource. Building upon Foerderer et al. (2019)'s research on KBRs, this thesis explored how platform owners can support new complementors in learning enterprise platform's APIs by designing broadcasting KBRs. This exploration incorporates approaches previously identified as useful in facilitating API learning, such as interactivity, extending knowledge on how such resources support API learning (i.e., Beaton et al., 2008; Gao et al., 2020; Jeong et al., 2009). Finally, it contributes to understanding how platform owners can address knowledge boundaries in various ways (C. Baldwin & Woodard, 2008; Foerderer et al., 2019; Ghazawneh & Henfridsson, 2013).

The following discussions will elaborate on how this thesis contributes to research, including a) a discussion of designing broadcasting API KBRs, b) examining a hosted developer sandbox as a KBR, and c) a platform and resource design level discussion about how platform owners addressing knowledge boundaries for supporting complementors in learning the APIs

8.1.1 Broadcasting API KBRs

Using Foerderer et al. (2019) definition of broadcasting KBRs, both the API testing tool and the interactive tutorial to the DHIS2 API can be classified as such. These resources aim to reach a wide range of complementors and offer a standardized approach to knowledge transfer without requiring direct interaction with the platform owner. Throughout this thesis, it has been explored how these broadcasting KBRs can be designed to address the identified challenges associated with API learnability. By evaluating the designed resources, the findings contribute to existing research on how broadcasting API KBRs can be designed to support new complementors in learning APIs.

Prior research about API learnability has been oriented toward the challenges complementors have in learning to use APIs and how API documentation can be structured to sufficiently onboard complementors (Jeong et al., 2009; Meng et al., 2018). Although technical documentation is relevant in conveying knowledge about how the API should be used, several researchers find this an imperfect resource for conveying the knowledge (Parnin & Treude, 2011; Robillard, 2009; Robillard & DeLine, 2011). Low research attention has been received on other types of documentation artifacts, representing a high value for complementors, including how to produce documentation, how they should be communicated, and the most efficient means for developers to consume them (Cummaudo et al., 2022). An interactive tutorial and an API testing tool for exploring the API were designed as part of this research. These serve as complementary resources to the technical documentation and can contribute to the lack of research on supplementary resources.

An interactive tutorial supplements technical documentation in supporting API learnability. The content of such tutorials differs from the technical documentation, offering a starting point and step-by-step tutorial for complementors to understand the API and its practical usage. The tutorials can support complementors with common difficulties experienced when learning a new API, such as mapping API documentation to practical usage and integrating multiple API elements when solving a task (Duala-Ekoko & Robillard, 2012; Robillard, 2009; Robillard & DeLine, 2011). The findings of this thesis indicate that complementors find such tutorials useful, using a step-by-step approach to learning the API. By providing an interactive tutorial containing practical scenarios, showing best usage practices, and engaging the complementors in solving practical tasks, platform owners can more efficiently address some of the knowledge boundaries complementors experience in learning the APIs. The tutorials can introduce and

explain concepts in a more accessible way without having to include all the technical details that should be included in the technical documentation.

Furthermore, these tutorials can introduce subsets of the API functionality. In this introduction, a more in-depth description can be included using illustrations and examples shown during this thesis to be valuable for complementors. Introducing only a subset of the functionality with in-depth descriptions can limit the overwhelming complexity typically associated with using enterprise platforms that offer a broad range of functionalities (Beaton et al., 2008; Foerderer et al., 2019; Jeong et al., 2009). In return, this makes it more manageable for new complementors to learn the APIs of these enterprise platforms.

Additionally, an API testing tool was designed in this research project. Previous research found that such interactive approaches were known to affect the user's experience learning the API (Gao et al., 2020; Robillard & DeLine, 2011). This thesis confirms this observation, finding that incorporating interactive resources supports the API learning outcome of complementors. Other studies related to such interactive tools show that users make heavy use of these tools either instead of reading documentation or used together with reading documentation (Daughtry et al., 2017; Macvean, 2016). Through implementing such an interactive tool in the online course, this research finds the same pattern: combining descriptions with an interactive approach where complementors can test the API and verify their understanding of the descriptions was helpful in API learnability.

In summary, this thesis has addressed supporting complementors learning APIs by designing supplementary broadcasting KBRs, including an interactive tutorial and an API testing tool. These resources extend Foerderer et al. (2019)'s research on KBRs by exploring how such broadcasting KBRs can be designed to support new complementors in learning enterprise platform APIs. Although this thesis only researched two types of broadcasting KBRs, further avenues for research can explore how other types of broadcasting KBRs support different expertise levels of complementors.

8.1.2 Hosted developer sandbox

As discussed in the related literature, KBRs play a critical role in overcoming knowledge boundaries that can arise between platform owners and complementors. Foerderer et al. (2019) suggest several empirical examples of a broadcasting resource, including technical documentation and sample code, which have received high research attention (Parnin & Treude, 2011; Robillard, 2009; Robillard & DeLine, 2011). Alongside other broadcasting KBRs, a “hosted developer sandbox” is listed as an empirical example of a broadcasting KBR, yet neither Foerderer et al. (2019) nor other research delve into the details of this specific type of KBR. Given the absence of a detailed description of hosted developer sandbox, this thesis interprets such resources as one provided by platform owners, enabling developers to experiment with the platform’s capabilities through a sandbox environment. These sandbox environments allow users to interact within an isolated system with sample data which does not harm the company’s data (Klaus & Changchit, 2020). In this thesis, a testing tool was hosted for complementors to use the APIs, which can be considered a hosted developer sandbox. Using the testing tool as a foundation, this thesis contributes knowledge of broadcasting KBRs by offering a description of a “hosted developer sandbox” as a broadcasting KBR including what it is, how it supports complementors and a design consideration for implementation.

A hosted developer sandbox is a resource hosted by the platform owner, providing complementors with a sandbox environment where they can test code, explore new features, and experiment with the boundary resources of the platform without worrying about negatively impacting the production environment. The hosted developer sandbox allows developers to work more efficiently and confidently using the platform’s boundary resources, as they can test their code in a safe and controlled environment. By facilitating an iterative loop of testing, feedback, and adjustments, the hosted developer sandbox enables complementors to quickly test assumptions about how to use the platform’s boundary resources.

As found in this thesis, a hosted developer sandbox supports complementors using the APIs. It enables complementors to experiment with platform capabilities without setting up a new code project or modifying existing code. The developer sandbox was particularly useful for new complementors with limited knowledge about existing tools facilitating testing and exploration. For instance, an API testing tool used to explore the API was designed as part of this research project. Although existing tools such as “Postman” leverage several of the same capabilities as the API testing tool, it was found that many new complementors were not

familiar with these external tools. Provisioning a hosted developer sandbox as part of the platform's broadcasting KBRs supports new complementors' option to test and explore through sandbox environments.

Through the findings, it has been suggested that a hosted sandbox is particularly efficient when used in combination with other broadcasting resources, such as tutorials. For example, this thesis found that an interactive tutorial combining text, illustrations, and examples with a hosted developer sandbox was particularly useful for learning to use the API. Thus, platform owners who want to provision such sandboxes should consider supplementing with additional resources in collaboration with the sandbox.

As a broadcasting KBR, the hosted developer sandbox can enable complementors to address challenges before requiring technical assistance by brokering and bridging activities. This saves costs for platform owners as such activities are generally more costly and time-consuming. If complementors do, however, require such assistance, the hosted developer sandbox can support such bridging and brokering activities. Throughout the conduction of this thesis, the hosted developer sandbox was often used when assisting students with concerns related to learning the API. Demonstrating usage through a sandbox environment provided practical knowledge on how to leverage the platform's capabilities.

If platform owners choose to design a hosted developer sandbox, a recommendation is to include functionality allowing users to easily convert the obtained results in the sandbox to a code example that is usable in code projects (design consideration 2). As found in this thesis, the export-to-code function was a valuable feature, allowing users to generate code that they can use either as a code example or directly apply in the code project. Jeong et al. (2009) proposed that it could be helpful with a feature enabling users to generate code when using these testing tools. Such functionality was introduced as part of this research, and it was confirmed that a significant number of contributors did find it helpful. Additionally, this thesis has demonstrated one approach for implementing this type of functionality.

In summary, this thesis extends Foerderer et al. (2019)'s research on KBRs by offering a description of a hosted developer sandbox, including what it is and how it supports complementors. The findings revealed that complementors, especially new complementors, greatly benefited from using hosted developer sandboxes when learning to use APIs. As a result, platform owners designing broadcasting KBRs should consider deploying such hosted developer sandboxes to support learning. Combining a hosted developer sandbox with other

complementary resources, such as an interactive tutorial and documentation, further enhances the learning experience for complementors.

Further avenues for research could explore how a hosted developer sandbox can be designed to support other areas of learning the platform's resources, such as its UI libraries. Another avenue to such developer sandboxes is to make them shareable and examine how this supports conveying knowledge. Google et al. (2017) found that collaborations with other developers are one reason for using API explorer tools. Similarly, throughout this research period, the hosted developer sandbox was helpful when assisting complementors who encountered difficulties. However, due to limitations of not being capable of sharing its state, the hosted developer sandbox was not possible to use for assisting remotely. This motivates further exploring how sharing the state of the hosted developer sandbox can support learnability.

8.1.3 Platform and Resource level design

API learnability is a crucial aspect of any platform, referring to the ease with which developers can understand, learn and use an API to build applications (Meng et al., 2018; Tello-Rodríguez et al., 2020). Throughout the diagnosis in this thesis, several challenges were identified in existing resources related to API learnability. Based on the diagnosis, new broadcasting KBRs were provisioned to support complementors using the API. However, the evaluation phase found that the designed resources have limited impact in addressing some diagnosed challenges, such as difficulties in understanding the terminology and the data model. This raises questions about the effectiveness of addressing knowledge boundaries by providing KBRs. Although the conducted approach of supplying KBRs should be considered an important strategy for overcoming knowledge boundaries in platform ecosystems (Foerderer et al., 2019), it does not capture the complete picture of supporting complementors in learning to use the APIs.

The platform literature suggests that overcoming knowledge boundaries is possible by shaping the boundary resources to support its complementors (Eaton et al., 2015; Ghazawneh & Henfridsson, 2013) and that knowledge boundaries result from platform design (C. Y. Baldwin & Woodard, 2008). On the other hand, Foerderer et al. (2019) argue the importance of providing KBRs for addressing knowledge boundaries. Drawing from the empirical findings and existing literature, an interesting understanding emerges regarding different approaches to how platform owners can address knowledge boundaries for its complementors. This understanding suggests that platform owners can focus on two levels of design for addressing knowledge boundaries in platform ecosystems: *resource-level design* and *platform-level design*:

Resource-level design refers to design efforts aiming to improve the KBRs for supporting complementors in using the boundary resources. As Foerderer et al. (2019) argue, the platform owner must provide KBRs to overcome knowledge boundaries. Throughout this research project, resource-level design was emphasized in addressing knowledge boundaries by designing various broadcasting KBRs, such as a testing tool and an interactive tutorial to support complementors learning to use the DHIS2 API.

Platform-level design refers to design efforts aiming to improve the usability of the platform's boundary resources themselves, not just the knowledge resources that accompany them. It's argued that knowledge boundaries are a consequence of the platform's design (C. Baldwin &

Woodard, 2008). This implies that designing the platform’s boundary resources addresses the knowledge boundaries that emerge when complementors engage with these, and hence, also the need for KBRs. For instance, complementors experienced challenges learning the DHIS2 API throughout the research project due to its complicated terminology and complex data model. The DHIS2 Platform owner could address these challenges by designing the core functionality to inherit less complexity or creating interfaces that abstract more of this complexity.

Figure 8.1, which expands on Ghazawneh and Henfridsson's (2013) model of boundary resource, illustrates the two levels of design, highlighting which elements of the platform ecosystem each design level encompasses. The platform-level design involves the platform and the design of the platform’s boundary resource. On the other hand, the resource-level design involves the KBRs, which support complementors developing third-party applications in using the boundary resources.

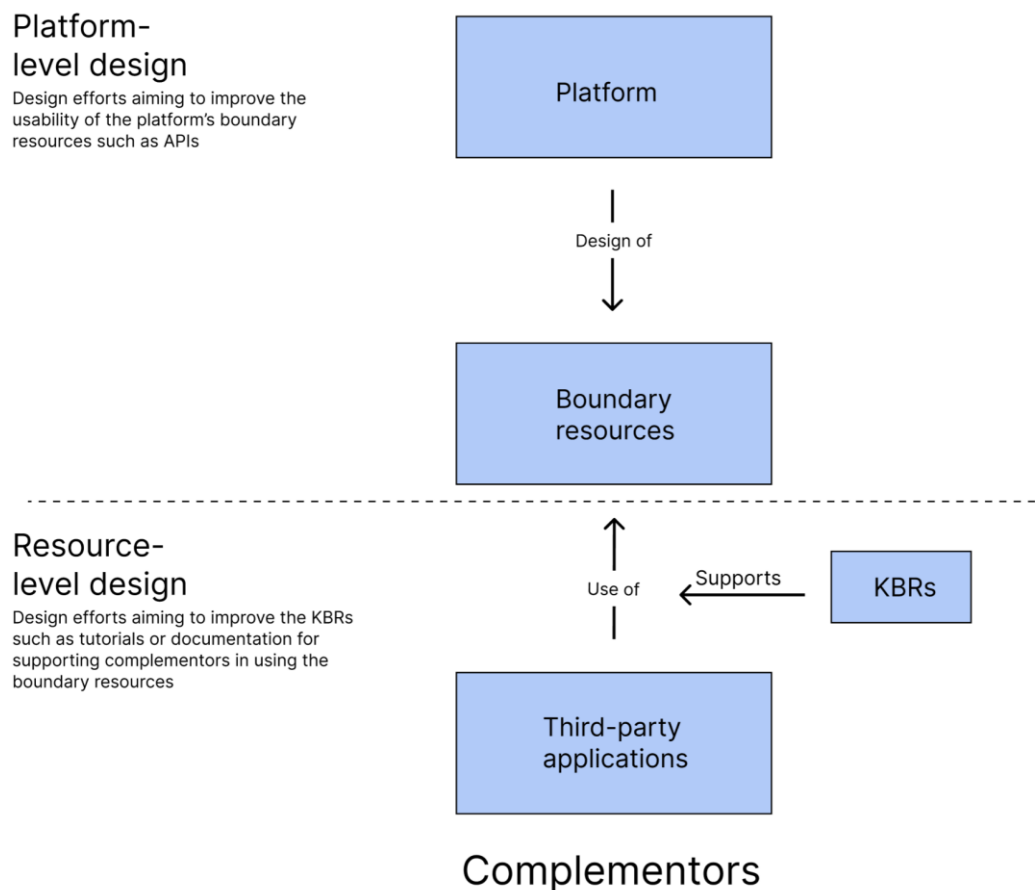


Figure 8.1: Design levels for supporting complementors

The levels of design should be considered jointly, as they can influence each other. As demonstrated in this thesis, providing additional KBRs for overcoming knowledge boundaries had a limited impact on some of the challenges associated with learning the API. One explanation is that the provisioned KBRs were inadequate in addressing the identified challenges. Another explanation is that this points to lacking usability in the boundary resources, where complexity leaks from the platform core and increases knowledge boundaries. In this case, provisioning KBRs to compensate may not be as effective in addressing the concerns, implying a *saturation* state where investment in improving the KBRs has a limited impact on learnability. In such a case, it could be more appropriate for platform owners to instead emphasize platform-level design by improving the usability of the platform's APIs.

However, emphasizing platform-level design may not be a straightforward task. Designing the boundary resources for enterprise platforms poses unique challenges due to the platform's large and complex nature. Enterprise platforms target a large group of complementors, each with a varying level of expertise (Jeong et al., 2009). The boundary resources should be designed to be user-friendly by abstracting complex core logic facilitating the diversity of complementors using them (Tiwana, 2014). However, serving the primary connectors to the platform core, it's also important that the boundary resources remain stable over time to ensure reliability for complementary applications that rely on these (C. Baldwin & Woodard, 2008; Tiwana, 2014). This stability requirement imposes a challenge by making it difficult for platform owners to design boundary resources that should be stable yet requires adoption. However, by not addressing the usability of the boundary resources concerns, this may result in broadening knowledge boundaries.

In response to these challenges, platform owners may turn to resource-level design, provisioning KBRs to support complementors in learning to use the boundary resources. Provisioning these types of resources is an important strategy for addressing knowledge boundaries (Foerderer et al., 2019). Additionally, supplying more KBRs can be considered an attractive approach as it does not require adjustments to the platform's design and thus won't break any functionality of the platform or its complementary applications. However, the extent and quality of KBRs should be evaluated, as insufficient or inadequate KBRs may not be the leading cause of the problem. If there is an excessive number of KBRs to explain a phenomena and additional KBRs do not efficiently address the knowledge boundary as demonstrated in this thesis, it may indicate saturation of KBRs. In this case, platform-level design may be more efficient for addressing the knowledge boundaries.

While platform owners may encounter the challenging choice of deciding which level of design – platform-level or resource-level to prioritize, it's important to consider the implications of each. On the one hand, platform-level design may be difficult due to the necessity of maintaining stable interfaces. On the other hand, resource-level design by supporting complementors with KBRs may not efficiently address the underlying usability issue, especially when there's a saturation of KBRs. This situation underscores the importance of evaluating both levels of design in addressing knowledge boundaries, looking for indicators such as saturation. However, saturation is likely not the only indicator. There are presumably more indicators that could guide platform owners in determining which level of design would be most effective in addressing the knowledge boundaries. Thus, additional research is required to identify and explore these potential indicators.

8.2 Contributions to practice

In addition to contributing to research, this thesis also seeks to provide practical knowledge on designing broadcasting KBRs for new complementors. One of the key contributions is the development of the four design considerations (1) Offering a simple testing tool, (2) Facilitate exporting to code, (3) Provide interactive tutorials, and (4) Support esoteric terminology comprehension that platform owners can consider when creating KBRs. These four design considerations can serve as valuable guidance for platform owners in designing broadcasting KBRs supporting complementors in learning the platform APIs. Furthermore, the thesis demonstrates how these design considerations can be designed.

Additionally, this thesis contributes to the learning content of the Master Course “Development in Platform Ecosystems” at the University of Oslo, teaching students about platform ecosystems and application development for such ecosystems. Specifically, students taking this course go through the self-paced DHIS2 app course, where they are taught how to develop applications for the DHIS2 platform. This research project has designed an API tool and an interactive tutorial within this DHIS2 app course, supporting students taking this course.

Inspections of the network traffic to the DHIS2 app course found that most visitors were not students of the Development in Platform Ecosystems course but other complementors interested in learning development for the DHIS2 platform. This broadens the scope of this research project's practical contributions by supporting students and other complementors of DHIS2 who visit the DHIS2 app course.

8.3 Limitations

In this thesis, several limitations should be acknowledged. First, only students were observed, and no experts were included in the research, as the focus was on new complementors. The students had a relatively homogenous starting point, as they all came from the University of Oslo. This homogeneity may have influenced the data collection by not capturing a broader range of expertise of complementors. The incentives for students were also different. Unlike complementors who may be motivated by developing a new app for the DHIS2 to become a complementor, students were primarily focused on completing the course.

Another limitation may be social desirability bias and the “demand effect.” The concept behind the demand effect is that subjects respond to a question as they believe the researcher desires (Grimm, 2010). Several students taking the course were familiar with my study, including that I had designed the API testing tool and the interactive tutorial, which may have influenced the data collection. To handle this possible bias, recommended strategies for minimizing social desirability were utilized, such as carefully designing the interviews to avoid leading questions (Grimm, 2010).

The study’s reliance on a single case where students learned development for the DHIS2 enterprise platform also presents a limitation, as the findings could be more generalizable if other enterprise platforms, such as SAP, were included.

Based on the findings of this thesis, several design considerations were proposed. It’s important not to view these considerations as strict rules for designing broadcasting KBRs supporting learning the platform APIs; instead, These design considerations can serve as valuable guidance for platform owners when designing broadcasting KBRs to support new complementors to learning the platform APIs.

Lastly, although this research project focused on enterprise platforms, the applicability of the design considerations might extend to non-enterprise platforms. However, it’s important to emphasize that the insights and recommendations made in this study are particularly tailored to enterprise platforms, reflecting their unique characteristics.

9 Conclusion

In conclusion, this thesis has addressed the research question, “*How can enterprise platform owners design knowledge boundary resources to support large audiences of new complementors in learning to use the platform APIs?*”. To address this question, the study participated in the “Development in Platform Ecosystems” course at the University of Oslo. In this course, 120 students learn how to develop applications in platform ecosystems, and as part of the curriculum, they are taught to use the APIs of the enterprise platform DHIS2. Throughout the course, this thesis collected data by diagnosing, designing, demonstrating, and evaluating resources that should support students in learning to use these platform APIs. The findings of this data collection resulted in a practical contribution of four design considerations, including (1) Offering a simple testing tool, (2) Facilitate exporting to code, (3) Provide interactive tutorials, and (4) Support esoteric terminology comprehension. These design considerations can serve as valuable guidance for platform owners when designing broadcasting KBRs to support new complementors learning the platform APIs.

Furthermore, the thesis contributes to existing research on boundary resource usage (Ghazawneh & Henfridsson, 2013), particularly in the context of platform APIs as a boundary resource. The thesis extends Foerderer et al. (2019)’s research on broadcasting KBRs in three ways: Firstly, it explores how various types of *broadcasting API KBRs* can be designed to support new complementors in learning enterprise platform APIs. Secondly, it explores the *hosted developer sandbox*, including a description of what it is and how it supports complementors. Third, it discusses *platform and resource level design*, highlighting different levels of design in platforms for addressing knowledge which contributes to understanding how knowledge boundaries can be addressed in various ways (C. Baldwin & Woodard, 2008; Foerderer et al., 2019; Ghazawneh & Henfridsson, 2013).

In conclusion, these contributions to research and practice extend the knowledge in platform ecosystems by providing insight into how to support new complementors learning the platform’s APIs. The platform strategy is endangered if not adequate resources are provided for supporting complementors in using the platform’s boundary resources, such as APIs (Foerderer et al., 2019). The thesis has addressed this challenge and contributes to the field of platform ecosystems (Tiwana, 2014) by exploring how platform owners can design these resources, supporting complementors learning to use the platform’s boundary resources. This is crucial for onboarding complementors and adapting to a successful platform strategy. (Bergvall-Kåreborn & Howcroft, 2014; Foerderer et al., 2019; Tiwana, 2014)

10 References

- 4.5 *The Data Model*. (n.d.). Retrieved June 1, 2022, from https://docs.dhis2.org/archive/en/2.30/developer/html/techarch_data_model.html
- Baldwin, C., & Woodard, C. J. (2008). The Architecture of Platforms: A Unified View. *Platforms, Markets and Innovation*. <https://doi.org/10.2139/ssrn.1265155>
- Baldwin, C. Y., & Woodard, C. J. (2008). The Architecture of Platforms: A Unified View. *SSRN Electronic Journal*. <https://doi.org/10.2139/ssrn.1265155>
- Beaton, J. K., Myers, B. A., Stylos, J., Jeong, S. Y. (Sophie), & Xie, Y. (Clare). (2008). Usability evaluation for enterprise SOA APIs. *Proceedings of the 2nd International Workshop on Systems Development in SOA Environments - SDSOA '08*, 29. <https://doi.org/10.1145/1370916.1370924>
- Bergvall-Kåreborn, B., & Howcroft, D. (2014). Persistent problems and practices in information systems development: A study of mobile applications development and distribution: Mobile applications development and distribution. *Information Systems Journal*, 24(5), 425–444. <https://doi.org/10.1111/isj.12036>
- Bianco, V. D., Myllarniemi, V., Komssi, M., & Raatikainen, M. (2014). The Role of Platform Boundary Resources in Software Ecosystems: A Case Study. *2014 IEEE/IFIP Conference on Software Architecture*, 11–20. <https://doi.org/10.1109/WICSA.2014.41>
- Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., & Klemmer, S. R. (2009). Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1589–1598. <https://doi.org/10.1145/1518701.1518944>
- Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2), 77–101. <https://doi.org/10.1191/1478088706qp063oa>
- Carlile, P. R. (2004). Transferring, Translating, and Transforming: An Integrative Framework for Managing Knowledge Across Boundaries. *Organization Science*, 15(5), 555–568. <https://doi.org/10.1287/orsc.1040.0094>
- Corbin, J., & Strauss, A. (n.d.). *Grounded theory research: Procedures, canons, and evaluative criteria*.
- Cummaudo, A., Vasa, R., Grundy, J., & Abdelrazek, M. (2022). Requirements of API Documentation: A Case Study into Computer Vision Services. *IEEE Transactions on Software Engineering*, 48(6), 2010–2027. <https://doi.org/10.1109/TSE.2020.3047088>
- Cusumano, M., Yoffie, D., & Gawer, A. (2020). The Future of Platforms. *MIT Sloan Management Review*.
- Danielsen, P. J., & Jeffrey, A. (2013). Validation and Interactivity of Web API Documentation. *2013 IEEE 20th International Conference on Web Services*, 523–530. <https://doi.org/10.1109/ICWS.2013.76>

- Daughtry, J., Macvean, A., & Luke, C. (2017). The Uses of Interactive Explorers for Web APIs. *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security - CCS '13*, 49–60. <https://doi.org/10.1145/2508859.2516655>
- DHIS2 Design Lab—HISP Centre*. (n.d.). Retrieved April 20, 2023, from <https://www.mn.uio.no/hisp/english/dhis2-design-lab/index.html>
- DHIS2 Overview*. (n.d.). DHIS2. Retrieved April 20, 2023, from <https://dhis2.org/overview/>
- Duala-Ekoko, E., & Robillard, M. P. (2012). Asking and answering questions about unfamiliar APIs: An exploratory study. *2012 34th International Conference on Software Engineering (ICSE)*, 266–276. <https://doi.org/10.1109/ICSE.2012.6227187>
- Eaton, B., Elaluf-Calderwood, S., Sørensen, C., & Yoo, Y. (2015). Distributed Tuning of Boundary Resources: The Case of Apple’s iOS Service System. *MIS Quarterly*, 39(1), 217–243. <https://doi.org/10.25300/MISQ/2015/39.1.10>
- Foerderer, J., Kude, T., Schuetz, S. W., & Heinzl, A. (2019). Knowledge boundaries in enterprise software platform development: Antecedents and consequences for platform governance. *Information Systems Journal*, 29(1), 119–144. <https://doi.org/10.1111/isj.12186>
- Gao, G., Voichick, F., Ichinco, M., & Kelleher, C. (2020). Exploring Programmers’ API Learning Processes: Collecting Web Resources as External Memory. *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 1–10. <https://doi.org/10.1109/VL/HCC50065.2020.9127274>
- Gat, I., Remencius, T., Sillitti, A., Succi, G., & Vlasenko, J. (2013). *The API Economy: Playing the Devil’s Advocate*. 26(9).
- Ghazawneh, A., & Henfridsson, O. (2013). Balancing platform control and external contribution in third-party development: The boundary resources model. *Information Systems Journal*, 23(2), 173–192. <https://doi.org/10.1111/j.1365-2575.2012.00406.x>
- Grimm, P. (2010). Social Desirability Bias. In J. Sheth & N. Malhotra (Eds.), *Wiley International Encyclopedia of Marketing* (p. wiem02057). John Wiley & Sons, Ltd. <https://doi.org/10.1002/9781444316568.wiem02057>
- Hevner, March, Park, & Ram. (2004). Design Science in Information Systems Research. *MIS Quarterly*, 28(1), 75. <https://doi.org/10.2307/25148625>
- HISP UiO Strategy Update 2019-2022*. (2019).
- Home*. (n.d.). DHIS2. Retrieved April 20, 2023, from <https://dhis2.org/>
- Jeong, S. Y., Xie, Y., Beaton, J., Myers, B. A., Stylos, J., Ehret, R., Karstens, J., Efeoglu, A., & Busse, D. K. (2009). Improving Documentation for eSOA APIs through User Studies. In V. Pipek, M. B. Rosson, B. de Ruyter, & V. Wulf (Eds.), *End-User Development* (Vol. 5435, pp. 86–105). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-00427-8_6
- Kauschinger, M., Schrieck, M., Boehm, M., & Krcmar, H. (2021). *Knowledge Sharing in Digital Platform Ecosystems – A Textual Analysis of SAP’s Developer Community*. 18.

- Klaus, T., & Changchit, C. (2020). Sandbox Environments in an ERP System Context: Examining User Attitude and Satisfaction. *Electronic Journal of Information Systems Evaluation*, 23(1). <https://doi.org/10.34190/EJISE.20.23.1.003>
- Macvean, A. (2016). *API Usability at Scale*.
- Meng, M., Steinhardt, S., & Schubert, A. (2018). Application Programming Interface Documentation: What Do Software Developers Want? *Journal of Technical Writing and Communication*, 48(3), 295–330. <https://doi.org/10.1177/0047281617721853>
- Myers, B. A., & Stylos, J. (2016). Improving API usability. *Communications of the ACM*, 59(6), 62–69. <https://doi.org/10.1145/2896587>
- Myers, M. D. (2020). *General References on Qualitative Research | Qualitative Research in Information Systems*. https://www.qual.auckland.ac.nz/general/#Qualitative_Research_in_Business_&_Management.
- Parnin, C., & Treude, C. (2011). Measuring API documentation on the web. *Proceeding of the 2nd International Workshop on Web 2.0 for Software Engineering - Web2SE '11*, 25–30. <https://doi.org/10.1145/1984701.1984706>
- Peppers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3), 45–77. <https://doi.org/10.2753/MIS0742-1222240302>
- Piccioni, M., Furia, C. A., & Meyer, B. (2013). An Empirical Study of API Usability. *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 5–14. <https://doi.org/10.1109/ESEM.2013.14>
- Pollock, N., Williams, R., & Procter, R. (2003). Fitting Standard Software Packages to Non-standard Organizations: The ‘Biography’ of an Enterprise-wide System. *Technology Analysis & Strategic Management*, 15(3), 317–332. <https://doi.org/10.1080/09537320310001601504>
- REST | APIs | SAP Business Accelerator Hub*. (n.d.). Retrieved May 13, 2023, from <https://api.sap.com/content-type/API/apis/REST>
- Robillard, M. P. (2009). What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software*, 26(6), 27–34. <https://doi.org/10.1109/MS.2009.193>
- 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>
- Sohan, S. M., Maurer, F., Anslow, C., & Robillard, M. P. (2017). A study of the effectiveness of usage examples in REST API documentation. *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 53–61. <https://doi.org/10.1109/VLHCC.2017.8103450>
- Strong & Volkoff. (2010). Understanding Organization—Enterprise System Fit: A Path to Theorizing the Information Technology Artifact. *MIS Quarterly*, 34(4), 731. <https://doi.org/10.2307/25750703>

Tan, W., Fan, Y., Ghoneim, A., Hossain, M. A., & Dustdar, S. (2016). From the Service-Oriented Architecture to the Web API Economy. *IEEE Internet Computing*, 20(4), 64–68. <https://doi.org/10.1109/MIC.2016.74>

Tello-Rodríguez, M., Ocharán-Hernández, J. O., Pérez-Arriaga, J. C., Limón, X., & Sánchez-García, Á. J. (2020). A Design Guide for Usable Web APIs. *Programming and Computer Software*, 46(8), 584–593. <https://doi.org/10.1134/S0361768820080241>

Tiwana, A. (2014). *Platform Ecosystems: Aligning architecture, Governance, and Strategy*.

Welcome to the DHIS2 Developer Portal | DHIS2 Developer Portal. (n.d.). Retrieved April 20, 2023, from <https://dhis2.github.io/>