Optimizing the usability of REST API reference documentation

Axel Holmqvist and David Jungermann

DEPARTMENT OF DESIGN SCIENCES
FACULTY OF ENGINEERING LTH | LUND UNIVERSITY 2021

MASTER'S THESIS

homepal
PROPERTY TECH & DATA

Optimizing the usability of REST API reference documentation

Published by
Department of Design Sciences
Faculty of Engineering LTH, Lund University
P.O. Box 118, SE-221 00 Lund, Sweden

Subject: Interaction Design (MAMM01)
Supervisor: Joakim Eriksson
Co-supervisor: Kevin Kimaryo at Homepal AB
Examiner: Mattias Wallergård

Abstract

With an increasingly growing demand for accessible data, the amount of REST API:s worldwide is growing. All REST API:s come with some form of reference documentation, which is crucial for the comprehension and usage of the API. Every API provider should therefore consider the usability of their reference documentation, to facilitate the learning and usage of their product.

This master's thesis was performed in collaboration with Homepal AB, a startup company within the real estate industry. The master's thesis explores what features and properties should be included in a reference documentation order to support the needs of the end users. Furthermore, it aims to investigate how different types of developers use reference documentation differently. These learnings were then applied to a prototype, that was subject to evaluation in several stages.

By using a user-centered design process and an iterative approach, with interviews and and a usability study, combined with literature studies of former research, heuristic analyzes of example documentation, along with insights from the private sector, prototypes of different levels were created and iterated with the goal of creating a final prototype illustrating the learnings, that then could be used as inspiration for improving the reference documentation of Homepal.

The result of the design process, and the final prototype, was mainly evaluated through user observation and feedback, as well as the usage of the System Usability Scale. The prototype was highly and widely appreciated among the users, and received an average SUS score of 95.

Keywords: REST API, reference documentation, usability, user centered design, user experience

Sammanfattning

Med allt större efterfrågan på tillgänglig data ökar antalet REST API:er markant världen över. Varje REST API har någon form av referensdokumentation, vilket är avgörande för förståelsen samt användandet av ett API. Samtliga API-leverantörer bör således överväga optimering av referensdokumentationens användbarhet för att underlätta inlärning och användning av sin produkt.

Denna masteravhandling utfördes i samarbete med Homepal AB, ett startup-företag inom fastighetsbranschen. Mastersavhandlingen ämnar att undersöka vilken funktionalitet och vilka egenskaper som bör inkluderas i en referensdokumentation för att uppfylla slutanvändarnas behov, samt undersöka hur olika typer av utvecklare använder sig utav referensdokumentation på olika tillvägagångssätt. Dessa lärdomar applicerades sedan slutligen på en prototyp som utvärderades i flera steg.

Genom att använda en användarcentrerad designprocess och ett iterativt tillvägagångssätt, med intervjuer och användbarhetsstudier kombinerat med litteraturstudier av tidigare forskning, heuristiska analyser av dokumentationsexempel, samt insikter från den privata sektorn, skapades prototyper på olika nivåer vilka itererades med målet att skapa en slutgiltig prototyp som illustrerar den kunskap som samlats, och som kan fungera som inspiration för att förbättra Homepals referensdokumentation.

Resultatet av designprocessen och prototypen utvärderades huvudsakligen genom användarobservation och feedback, samt genom användning av System Usabilty-skalan. Prototypen uppskattades mycket och brett bland användarna, och fick ett genomsnittlig SUS-resultat på 95.

Nyckelord: REST API, referensdokumentation, användbarhet, användarcentrerad design, användarupplevelse

Acknowledgements

This research paper has been conducted during the Spring of 2021 and is presenting the work of the Master's Thesis by us, Axel Holmqvist and David Jungermann. With its completion, we are finishing our studies in Information & Communication Technologies Engineering at the Department of Design Sciences, Faculty of Engineering LTH, Lund University.

The work has been done at Homepal, in Malmö, Sweden, to which we would like to say a big thank you. The office space successfully made us feel like members of the team, and it has truly been a pleasure writing our Master's Thesis at the company. An extra thank you to our supervisor, Kevin Kimaryo, who has been providing for all of our needs.

We would also like to thank our supervisor at LTH, Joakim Eriksson, who has guided us throughout the thesis with relevant feedback and insightful ideas regarding the design process.

Finally, we would like to express our gratitude to all of the interview participants and user testers, as well as employees being a part of feedback workshops, for taking your time and voluntarily contributing to our work.

Lund, May 2021
Axel Holmqvist and David Jungermann

Contents

1 Introduction **9**
- 1.1 Background . . . 9
  - 1.1.1 Homepal . . . 10
- 1.2 Related Work . . . 10
- 1.3 Scope . . . 11
- 1.4 Purpose and Goals . . . 12

2 Technical Background **13**
- 2.1 Application Programming Interfaces . . . 13
  - 2.1.1 Local API:s . . . 13
  - 2.1.2 Web API:s . . . 14
  - 2.1.3 Open API:s . . . 16
  - 2.1.4 RESTful API:s . . . 17

3 Theoretical Background **19**
- 3.1 User-Centered Design . . . 19
- 3.2 Design Process . . . 20
- 3.3 Usability . . . 20
- 3.4 User Experience . . . 21
- 3.5 Universal Design . . . 22
- 3.6 Heuristic Analysis . . . 22
- 3.7 Interviews . . . 22
- 3.8 Prototyping . . . 23
- 3.9 Usability Testing . . . 23
  - 3.9.1 Planning Usability Studies . . . 24
  - 3.9.2 System Usability Scale . . . 25
- 3.10 API Reference Documentation . . . 25
  - 3.10.1 Supplementary Documentation . . . 26
- 3.11 Types of Developers . . . 27
- 3.12 Concerns with API Usage . . . 28

3.13 API Documentation Guidelines . . . 32
3.13.1 Former Research . . . 32
3.13.2 Insights from the Private Sector . . . 34
3.14 Usage Examples . . . 35

4 Design Process **39**
4.1 Timeline of Key Activities . . . 39
4.2 Understand Phase . . . 41
4.2.1 Literature Study . . . 41
4.2.2 Heuristic Evaluation . . . 41
4.2.3 User Interviews . . . 43
4.2.4 Summarized Results . . . 48
4.3 Explore Phase . . . 48
4.3.1 MidFi Prototyping . . . 50
4.3.2 HiFi Prototyping . . . 55
4.4 Materialize Phase . . . 61
4.4.1 Usability Testing . . . 61
4.4.2 Final Prototype . . . 67

5 Discussion **71**
5.1 Understand Phase . . . 71
5.1.1 Literature Study . . . 71
5.1.2 Heuristic Evaluation . . . 72
5.1.3 User Interviews . . . 72
5.2 Explore Phase . . . 73
5.2.1 Feature Prioritization . . . 73
5.2.2 MidFi Prototyping . . . 73
5.2.3 Developer Team Talk . . . 73
5.2.4 HiFi Prototyping . . . 74
5.3 Materialize Phase . . . 74
5.3.1 Usability Testing . . . 74
5.3.2 Result Evaluation . . . 75
5.3.3 Final Prototype . . . 76
5.4 Future Work . . . 78

6 Conclusion **81**

References **83**

Appendix A User Interviews **89**
A.1 Interview Questions . . . 89
A.2 Form of Consent . . . 91

Appendix B Usability Tests **93**
B.1 User Tasks . . . 93
B.2 Form of Consent . . . 96
B.3 User Feedback Questions . . . 97

Appendix C Implementation **99**
C.1 Selected Implementations 99

Chapter 1
Introduction

This introductory chapter aims to describe the importance of API:s to the modern web, and why documentation plays a tangible role in regards to the adoption and usage of these. Following this, a description of the company which this work is based around, Homepal AB, will follow, with the main goal of describing the context in which this study is performed. Thereafter, this chapter will present the purpose, goals and scope of this thesis work. Lastly, an aggregation of previous related research and findings within the area will act as foundation for further work and considerations made in this thesis.

1.1 Background

Application Programming Interfaces, API:s, are the backbone of the modern web, and their usage is constantly increasing. In 2016, there were around 12 000 open API:s, which is 30 times more compared to 2006 [9], and there are no signs of this growth slowing down. In a study conducted by RapidApi[1], API usage was set to increase in 2020, with 65.5% of the respondents answering that they intend to use more API:s in 2020 [35].

The goal of an API is to provide a service to a consumer, in order to facilitate re-use of previously implemented software. This drastically lowers the complexity of implementing interconnected software, since it is possible to leverage already existing API:s to implement the desired functionality. Web API:s can be based on several different underlying technologies. For the consumer, the underlying technology mainly decides on what format the data should be accessed and returned, but the overall goal is the same for all types of Web API:s. According to the same study conducted by RapidAPI [35], the most prominent technology is RESTful API:s, commonly referred to as REST API:s, with 62.50% using it to implement Web API:s in production.

In order to use third-party software efficiently, the API provider must supply resources to facilitate learning and usage. In studies conducted by Robillard [36], as well as Robillard

[1] https://rapidapi.com/

and DeLine [37], it is concluded that lacking documentation negatively affects the efficiency of learning an API, and is actually the largest obstacle for developers when faced with the task of learning a new API. The documentation in itself can act as an obstacle to overcome. Uddin and Robillard claim that poor documentation may lead to dissatisfied developers, and an inefficient development process, along with potential abandonment of the API [39].

Therefore, as an API provider, putting effort into your documentation may be a well-worth investment, as it can increase the number of satisfied developers and the software that is built on top of your work will be of higher quality. Furthermore, it may increase engagement with your platform, as the developers working with your API are satisfied with the service you provide.

1.1.1 Homepal

Homepal AB[2] is a startup company based in Malmö, Sweden. They are building a platform for organizing and maintaining data for real estate companies. These property companies oftentimes have a large number of intertwined and complex systems for data control. These systems are often based on old technologies, and are difficult to access, which leads to duplicated data with lacking quality. Homepal aims to simplify integration with this data, by providing a REST API for third-party developers, called *Homebase API*, as well as a transparent data model that can be used to structure the complex data generated with the real estate companies. The idea is that instead of writing unique software integrations for every company, you can use Homepal's services for all companies, meaning that software can be reused for all property companies on the Homepal platform.

In order to facilitate this process, Homepal has a developer portal that acts as a starting point for developers that want to use the API and other services, that also holds the documentation along with guides on how to get started using the platform, as well as tools for newer and more experienced developers alike.

1.2 Related Work

There are no doubts that software, and local API documentation is a well-researched subject which should be seen as widely spread and thoroughly examined. However, conducted research that specifically concerns *the usability of REST API reference documentation* is a less explored area when it comes to published work within academia. Despite the growing interest, there is a lack of exhaustive studies concerning individual experiences from developers within this area. After having examined related work for this thesis, two major research groups, from Mersburg University of Applied Sciences and School of Computer Science at McGill University respectively, were found that had a major focus on REST API reference documentation and its usability aspect.

Meng, Steinhardt and Schubert have attempted to understand how developers use API documentation by conducting a study where developers are tasked with solving problems using an API new to them [21]. The results indicate how documentation should support developers when learning a new API, and how documentation can be improved in order to

[2]https://homepal.se/

increase the support for developers. In a subsequent article Meng, Steinhardt and Schubert [22] put the guidelines suggested in the article above to the test. In this study, they use these guidelines to compare developers using optimized documentation and developers using non-optimized documentation. The study showed that the optimized documentation decreased the number of errors, and made the developers more efficient in completing their tasks.

Uddin and Robillard [39] claim that the act of creating functioning documentation for an API is difficult, and that these problems propagate and impact the usage of the API. Ten often occurring documentation problems were selected, and were the focal point of a large study in order to investigate which issues affected developers the most and why. Furthermore, in a study by Robillard [37], developers are asked on the obstacles and challenges with learning a new API. The results show that an API that is easy to use is also easier to learn, and that the overall usability affects how the API is perceived and understood when learning to use it. Furthermore, Robillard claims that: *"[...] the way to foster more efficient API learning experiences is to include more sophisticated means for developers to identify the information and the resources they need-even for well-designed and documented APIs."*. Robillard and DeLine [36] also conducted a study that bring up the difficulties of learning a large API, and how this affects developers and their efficiency. In order to investigate what actually makes API:s hard to learn, a study was conducted in which 440 developers were asked about difficulties and obstacles that may arise when using a new API. Based on this survey, five guidelines for constructing API documentation were constructed; the API documentation should include usage examples, should be unambiguous and complete, support different scenarios for usage, convenient to navigate, and finally, include relevant design elements.

In addition to these two research groups, Sohan, Mauer and Anslow have, together with the aforementioned Robillard, carried out a study [38] on the effectiveness of usage examples in Web API documentation. The authors claim that there is a distinct difference in features between local and Web API:s and that existing research on API documentation usability has primarily focused on *local* API:s, thus the need for research regarding *Web* API documentation usability.

1.3 Scope

This thesis is written within the area of Interaction Design, which consequently will be the main focus. Therefore, the pure textual content of documentation will not be considered in practice, despite it obviously affecting the overall usability of the documentation. In the final stages of this thesis, it will be assumed that the present documentation is factually correct, complete and up-to-date. This delimitation is introduced as a means to prevent this thesis from diving too deep within the field of technical writing, as well as sparing the authors from having to specify the details of the API. This further allows the authors to focus on documentation at a higher abstraction level, where features and concepts can be evaluated from a usability standpoint.

As mentioned in **Section 2.1**, there are considerable differences between local API:s and REST API:s, which warrants differences in how to approach their documentation. Since the collaborating partner, Homepal, is working on a RESTful Web API, this thesis will only focus on the documentation of these API:s. In this thesis, the term *API* will therefore refer to *RESTful Web API* if nothing else is explicitly stated.

Furthermore, the term *documentation* can refer to various types of information, and different abstraction levels. For example, should a "Get Started"-guide be referred to as documentation? Is a StackOverflow thread, or a blog post considered as documentation? This thesis will focus mainly on *reference documentation*, and if nothing else is explicitly stated the term *documentation* will refer to reference documentation for REST API:s. Also, since Homepal's API is not yet widely spread, the authors assume that the documentation is the only source of information on the API, which may not be the case for more established services. This thesis will therefore focus entirely on the documentation provided by Homepal.

Lastly, due to the nature of the focus on interaction design, the authors have chosen not to focus and highlight programming contributions at large. The design process of this thesis included programming to a large extent in the implementation phase, but these solutions will only be briefly explained, in order to pertain to the predefined scope.

1.4 Purpose and Goals

The academic purpose is to further elaborate on–and extend–previous work made in regards to REST API reference documentation, by applying theory, as well as insight from the industry and users in order to create a better experience for users of the Homebase API documentation. There are aspects and features that affect the overall usability of the documentation that will be evaluated through various forms of heuristic analysis, interviews, prototyping and usability testing. By doing this, the authors of this thesis hope that it will be made evident what factors positively affect REST API reference documentation, and how different developers, and their respective needs, can be supported when using the Homebase API documentation. In extension, successful approaches and solutions can be applied and implemented for the Homepal Developer Portal. The purpose of this thesis is summarized through the goals below:

Goal 1: Understand what features and properties should be included in an REST API reference documentation to support the needs of the developers.

Goal 2: Understand how different types of developers use documentation differently, and how the documentation accommodates for the needs of different developers.

Goal 3: Apply what was learned in relation to previous goals in order to create a prototype that performs well in evaluation.

Chapter 2
Technical Background

This chapter aims to briefly go over and explain the technical prerequisites for this thesis, in order to provide the reader with the needed context and what technology the work is based upon. The goal is to clearly differentiate different types of API:s and the core technologies they are based upon, as well as briefly explain the relevant technology for this thesis. Worth noting is that all aspects of REST API development cannot be included in this thesis, due to the defined scope. This means that less general terminology will be excluded, and interested readers are encouraged to read up on concepts that are not explained further.

2.1 Application Programming Interfaces

An *Application Programming Interface*, commonly referred to as an *API*, is used to interface two or more pieces of software. API:s define how information should be shared, and how requests to the API should be made, and what is returned from API based on the request. Logically, API:s are often divided into a *client*, the consumer of an API and a *provider*, an entity that maintains and provides the service to the client. API:s come in a multitude of variants, with different purposes and underlying technologies.

2.1.1 Local API:s

A *Local API* is kept locally on the client side, and is often included as a library, framework or package that can be used to access additional functionality locally from the client. An example of a local API is the Java JDK [1]. By linking to a local API, the client can access functions and subroutines from the API, that can then be used as if they were client's own. Local API:s are outside of the scope for this thesis, but it is important to clearly distinguish these from other types of API:s in order to fully understand the context of the work.

[1] https://docs.oracle.com/en/java/javase/15/

2.1.2 Web API:s

Web API:s are made accessible through the internet, and commonly use *HTTP* (see **Section 2.1.2**) to establish communication between a client and a provider, in order to expose functionality that can be used by the client. In Web API terminology, the provider is often referred to as the *server*, since the API is often placed on a server that can be accessed remotely by the client. See **Figure 2.1** for an illustration of the server client infrastructure, alongside a Web API.

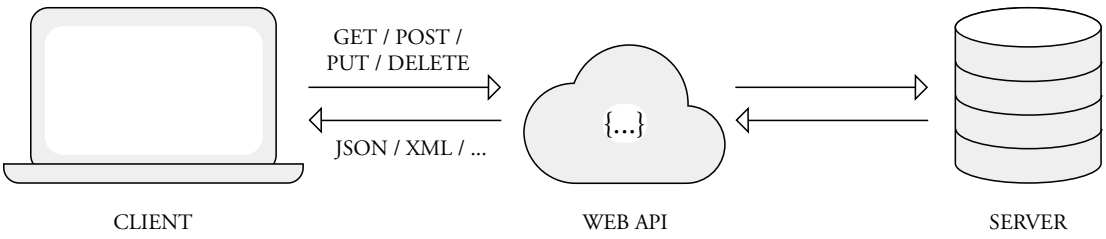


Figure 2.1: Web API Client Server Architecture.

The client can make requests to the server, with the API acting as an intermediary. Depending on what request is made to the API, the server responds with an appropriate object for what was requested. A common scenario is requesting data from the server, through the API, in order to use this data on the client side. Depending on the configuration, the returned object can be in various formats, e.g. *JSON* or *XML*, but as noted in the study conducted by RapidAPI [35], JSON is the most popular data format, and is also the data format used in the Homebase API, which makes it relevant to this thesis. For an explanation of JSON, see **Section 2.1.2**. It is also possible for the client to change the data present on the server with the use of the API. A common property for a clear majority of Web API:s is *CRUD*, which is an abbreviation of Create, Read, Update, Delete, and describes the types of actions that are usually performed on the data held by the server.

Web API:s can have different underlying technologies that define how the communication is structured in detail. However, this thesis is only concerned with REST API:s, which are explained further in **Section 2.1.4**.

HTTP

Hypertext Transfer Protocol is a communication protocol, that is responsible for a majority of the communication handled over internet. For the purpose of this thesis, HTTP will be explained with foundation in its usage related to Web API:s, but the protocol can be used for many purposes outside of the scope of this thesis. One of the fundamental properties of HTTP, that are used heavily in Web API:s are *HTTP Status Codes* [7]. These codes are returned by the server upon handling a request from the client, and depending on the type of request sent from the client and the result of said request, the server returns an appropriate code. An excerpt of the most common codes can be seen in **Table 2.1**.

These status codes are just a small sample of all available codes, but can be used to illustrate the core purpose of them; to provide feedback that clients can base their logic on. If

Table 2.1: HTTP Status Codes.

Status Code	Meaning	Usage
200	Ok	Succeeded request.
201	Created	Successfully created a new resource
404	Not Found	Server could not find the requested resource.
500	Internal Server Error	Internal Error with server.

the server returns an error code, the client knows that something went wrong in a previous request, and can adapt accordingly.

The second relevant core feature of HTTP in regards to this thesis is *Request Methods*. These methods are used, and sent to the server, by the client, and they indicate what action the client wants to perform on the server [7]. Examples of request methods can be seen in **Table 2.2**.

Table 2.2: HTTP Request Methods.

Method	Functionality
GET	Retrieves data
POST	Adds new resource
PUT	Edits resource
DELETE	Removes resource
PATCH	Partially modifies resource

Uniform Resource Identifiers, URI:s, are used to uniquely identify a resource. In practice, these are strings that are formatted in a way that allows them to uniquely point out a resource that the server holds so that the resource can be accessed [7]. In the context of Web API:s, URI:s are often referred to as *Endpoints*.

In the same manner as returning an object from the server to the client, the client can pass objects of different formats to the server. These are referred to as *Message Bodies* in the HTTP specification [7], and can be used to send additional information to the server. If the client wishes to create a new resource, they can supply the parameters for the object they wish to create in the message body.

In order to specify the operation of the HTTP communication, the client can also pass values as *Headers* to the server. Header fields can specify multiple parameters of an HTTP message, for example the data format and length for the message bodies, inclusion of cookies and access control.

Conclusively, HTTP's functions are used to provide the actual functionality of Web API:s. If a server holds information about *Users*, and each user has a unique social security number, this number can be sent via HTTP to the server to retrieve other information about this user. An HTTP request, from the client sent to the server, in order to retrieve information about a single user, could therefore look like this:

```
GET    http://example-api.com/users/960318-9001
```

Since the client uses the GET request method, nothing needs to be supplied in the message body of the request. The social security number is passed as a parameter in the URI, and can be parsed by the server. If everything has gone according to plan, the server will respond with something similar to this:

```
HTTP 1.1 200 OK
<Headers>
...
{
   ssn: "960318-9001",
   name: "Kent Andersson"
}
```

The status of the completed request is sent back to the client, along with the requested resource passed in the message body. Headers are omitted in the example.

JSON

JavaScript Object Notation is a syntax and open standard file format for storage and transfer of data. As previously mentioned, it is used heavily within Web API:s, and is also in use at Homepal. Essentially, a JSON object is a key-value store, where an object has a unique key that is providing a value [4]. JSON is easily parsed and created by computers, while also being easy for humans to read and comprehend. The syntax is simple, yet still offers appropriate amounts of complexity and expressivity. For example, you can create arrays and nested objects in order to accurately represent complex data structures. JSON is not dependent on any language, and can be used with almost all programming languages. Below follows an example of a JSON object.

```
{
   id: 1,
   name: "Kent Andersson",
   ssn: "960318-9001",
   children: ["Lisa Andersson",
              "Filip Andersson"]
}
```

2.1.3 Open API:s

Open API:s are API:s that are publicly available, with few restrictions on access. These are often based on public data, or by authorization and authentication to the data you wish to access. An example of an Open API is Spotify[2]. By using this Web API, you can access data available to all people, but you can also authenticate yourself in order to access data specific to your user account. The Homebase API works in a similar way in regards to access. Developers who wish to integrate with the API request access via the Developer Portal are

[2]https://developer.spotify.com/documentation/web-api/

subsequently provided with means of authorization via *OAuth*[3], and are then granted access to the appropriate data. Furthermore, third-party developers can use the sandbox along with the mock data, without authentication. This allows developers to develop their application without actually authenticating to access the real data.

2.1.4 RESTful API:s

A *RESTful API* is an architectural software style, that can be applied to Web API:s. REST stands for *Representational State Transfer*, and was coined by Roy Fielding in 2000 [10]. Fielding defines REST by applying various constraints in sequence to a blank canvas project. There are several constraints that must be met for a service to be considered entirely RESTful. However, as noted by Pluskiewicz, there are common misconceptions around REST and its usage [32]. He claims that it is hard to pertain to all constraints as presented by Fielding, and that in reality most "REST" API:s are technically not RESTful at all, which has subsequently coined the terms *RESTish* and *REST-like*, to represent API:s that aim to be RESTful, but breaks some properties of the definition. This thesis will however adhere to a more relaxed approach to the RESTfulness, and will also consider API:s that are not entirely RESTful by definition.

This thesis will not bring up all the properties and constraints that define a REST API in detail, since it lies outside the scope for this thesis. But, knowing the core concept of the architecture will be beneficial for understanding what should be included and considered when constructing documentation. There are three main legs that make up the foundation of a REST API, according to Pluskiewicz [32].

1. Representation
2. State
3. Transfer

Representation is a depiction of a resource on the server, meaning that a client does not directly modify the resource on the server, but rather modifies a representation of the resource that then can be used as a blueprint to modify the actual resource. *State* refers to the internal state of the *client* application. This state is created from one or more resource representations. Finally, *Transfer* refers to the act where a client can transfer between different states by retrieving and altering resources placed on the server. In HTTP, this is done by using the aforementioned request methods, and using the response from the server to update the internal client state. A memory rule, brought up by Pluskiewicz [32], is that *"Each endpoint REpresents a State Transfer"*. A prerequisite for a REST service is the Client-Server architecture explained in **Section 2.1.2**. It is also important that the server does not hold any form of history on the communication with the client. This is due to the *Statelessness* property defined by Fielding. Essentially, this means that each request sent to the server should be atomic, and should contain all information needed for the server to fulfill the request. Furthermore, all resources should be uniquely identified with the use of a URI [10].

[3]https://oauth.net/2/

Chapter 3
Theoretical Background

This chapter aims to bring up the theoretical background for this thesis, both in relation to REST API reference documentation, but also in regards to interaction design as well as the design process and methodology that has been applied and used during this thesis work.

3.1 User-Centered Design

Preece, Rogers and Sharp describe the concept of *User-Centered Design* as involving users throughout the design process [34]. They claim that this stems from the fact that real users always have a different perception of what is being evaluated than a person that has worked on the product, or is involved with the process in some manner. You, as a product developer, are already somewhat aware of the possibilities and limitations of the product, which colors your perception of how the product should be used. Opening up to feedback from users without prior experience with the product can provide valuable insights, as well as new outlooks on the subject product, which in turn can yield a better, more usable end result. In Norman's *The design of everyday things* [30], the author brings up three aspects to consider, in order to maintain focus on the user throughout the design process:

1. Early focus on the users.
2. Empirical measurements.
3. Iterative design process.

There are several methods that can be used to incorporate users, and their feedback into the design process – for example, by usability testing and evaluation, surveys or interviews. Norman states that the goal of the user-centered design philosophy is to understand the needs of the users, and how to accommodate for these needs by representing them in your design.

3.2 Design Process

A concept within interaction design that is tightly coupled with user-centered design is the *Design Process*. Preece, Rogers and Sharp define the process as an iterative four step procedure [34].

1. Identify requirements.
2. Develop design alternatives.
3. Develop prototypes.
4. Evaluate.

The goal is that these activities are dependent on each other, and should be performed multiple times, iteratively. In coherence with the user-centered approach, the user's needs should be considered on every activity. The purpose of this workflow is to continuously innovate and improve on previous work, so that the product constantly evolves, and the end result is the best possible product for the specified requirements. These requirements should therefore be grounded in a rigid, well-researched understanding of the end user. There are multiple conceptual models of this process, but most are based around the same key activities. This thesis will follow a process based on the model described by Gibbons [11]. An illustration of this process can be seen in **Figure 3.1**.

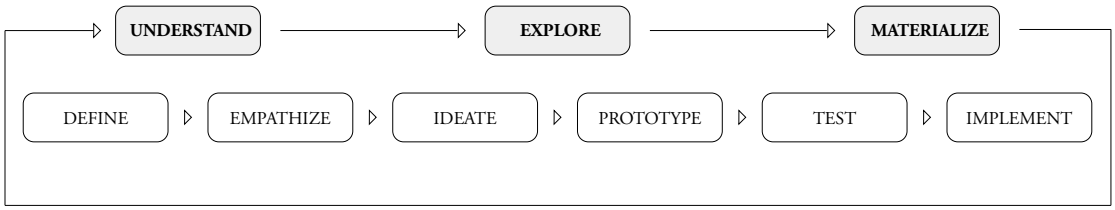


Figure 3.1: Re-illustrated Design Process from Gibbons [11].

3.3 Usability

Preece, Rogers and Sharp define *Usability* as a quality attribute that assesses the degree to which interactive products are easy to use and learn [34]. This includes optimization of the interaction that users have with the interactive product. Preece, Rogers and Sharp has broken down the concept into six goals:

1. Effective
2. Efficient
3. Utility
4. Learnable

5. Memorable

6. Safe

These goals can be considered when evaluating the usability of an interactive product, usually by asking questions regarding a product, related to a certain goal. Preece, Rogers and Sharp continue by explaining that problems or issues with a design can be discovered early, if these usability goals are considered by the designer.

3.4 User Experience

User Experience (UX) is a broad concept, and should be seen as the total experience with a system. Nielsen and Norman [29] define it as *"[...] all aspects of the end-user's interaction with the company, its services, and its products."*. Preece, Rogers and Sharp [34] formulate it as a measurement of how humans feel for a product and their satisfaction or contentment of using it. Furthermore, they state that it is not possible to design a user experience, only design *for* a user experience. Unfortunately, there is no magical formula, unifying theory or framework for this, only methods and guidelines. McCarthy and Wright have, however, made an effort in creating some kind of framework by defining what they call "The Four Threads of Experience" [19]. The threads should not be seen as fundamental elements of experience, *"[...] rather ideas to help us think more clearly about technology as experience."* [19]. The four threads include:

The Sensual Thread: represents the sensual engagement and immersive part of an experience, formerly the interplay, interaction and connection to the system in question.

The Emotional Thread: represents how emotions are connected to the situation and the experience. For example the joy of solving a problem, or the frustration of unresponsive functionality.

The Compositional Thread: represents the narrative part of an experience. What's this about? Where am I? What has happened? What will happen if...? In other words inner thoughts and questions that are raised in parallel to our experiences.

The Spatio-Temporal Thread: represents the place and the time where the experience is taking place, and how this effect the overall experience.

When thinking about user experience these four threads have to be merged to get a weaving instead of singular threads, in order to capture the often inter-weaved reality of the user's experience.

Lastly, Nielsen and Norman state that it is important to distinguish UX from both User Interface (UI) and Usability. UI is merely a part of the design, where the UX can be bad, due to other parameters, even though the UI is well designed. Usability is solely a quality attribute of UI covering whether a system is easy to learn, if it is efficient to use, how pleasant it is, and so forth [29].

3.5 Universal Design

Another staple concept in the field of interaction design is *Universal Design*. People have different physical and psychological attributes, that may affect how they use a product. Norman [30] brings up different physical factors, such as height, that affect how a user interacts with a design. In the context of this thesis, factors concerning prior API knowledge and overall software development skills are more applicable to the concept of universal design. The goal is to design a product that can be useful for all users, but this is not an easy task – sometimes there must be compromises. Norman claims that some things must be different for different people, but also for different times and contexts. Norman continues by explaining that a single product, often can not fit the needs of all users, but it can be adapted and optimized to everyone's benefit. By providing flexibility, you allow the user to decide how they want their experience, which in turn allows for a user base with a broader spectrum of defining psychological and physical attributes.

3.6 Heuristic Analysis

Heuristic Analysis is a form of evaluation, where predefined rules, commonly referred to as *heuristics* are used to locate issues with a product or interface, as stated by Nielsen [26]. Traditionally, heuristic evaluation is used to find usability issues, where the heuristics defined are commonly various usability principles. Nielsen further states that a small group of evaluators should find issues together, since a single evaluator may miss certain issues in the interface. However, the law of diminishing returns applies in this case, and Nielsen therefore claims that the number of evaluators can be kept quite low, while still finding the clear majority of issues. Nielsen recommends a number of evaluators from three to five.

While this thesis is very interested in the general usability aspect of the final product, the focus lies within how well the documentation is structured. Therefore, the set of heuristics used is different from more traditional usability heuristics. The heuristic analysis is described further in **Section 3.13.1** and **4.2.2**.

3.7 Interviews

Preece, Rogers and Sharp describe four main categories for conducting interviews: *Unstructured, semi-structured, structured* and *group interviews* [34]. The names stem from the amount of control that the interviewer has over the conversation, where a structured interview is the most rigid in its approach. The suitability of a given interview type depends on what results are sought after. If the interviewer is looking for *quantitative* data, a structured interview is a better choice, since the closed questions can be used to draw general conclusions, when aggregated with answers from interviews with other participants. On the other side of the spectrum, an unstructured interview, with more open-ended questions, can provide more *qualitative* data, since the answers can be more elaborate and detailed [34].

However, in order to get the most out of your interview, it must be meticulously planned in advance. As stated by Nielsen, there are several pitfalls to conducting interviews with users [28]. One of these pitfalls is asking users to either remember previous use of a product,

or speculating on future use of said product. Therefore, Nielsen claims that you should only focus on the present, and ask questions regarding their current usage, and their opinions on what is placed directly in front of them.

On the contrary, Nielsen states that an interview can be a great tool for exploring the general attitude of users in regards to a product or a problem. These opinions can then be translated into concrete design properties by the designers. However, as an interviewer, you should avoid coercing users into giving their opinions, which Nielsen refers to as the *Query Effect*. When asked, users tend to give their opinion, regardless if its well-formed or not, and may not even reflect their actual preferences. This can lead to malformed requirements down the line, due to making decisions based on ill-formed opinions. Rather, regard unsolicited opinions higher, as they are more likely to be something that actually affects the end users. Lastly, using only interviews as a means of obtaining user input may not be the best approach. However, by combining it with other forms of user evaluation, through *Data Triangulation*, one can make the end result greater than the sum of its parts [28].

3.8 Prototyping

Prototyping is a very useful tool for a number of reasons. Preece, Rogers and Sharp claim that it can be used as a means of facilitating communication between stakeholders at various stages in the design process [34]. Furthermore, it is an effective, quick and cheap canvas in order to discover, test and evaluate designs. Rather than blindly creating the product, and subsequently waiting for user feedback, you can develop a prototype, and evaluate it quickly. This allows for quicker iterations, and hopefully, a more complete and well-functioning design in the end.

As the the design process progresses, the amount of resources spent on prototypes is generally increased. A more refined prototype will yield better results, as it will be more similar to the end product, therefore providing feedback that in theory should be more applicable. However, on the other hand, it takes longer to develop, which renders them less efficient. Early on, as Preece, Rogers and Sharp note, a *Low Fidelity Prototype* is oftentimes preferred. These prototypes, often abbreviated *LoFi prototypes*, are usually not similar to the final product, and represent little-to-none of the functionality. They are used more to represent ideas, than for actual evaluation, and are quick and easy to iterate on. As the project progresses, designers tend to construct *Middle-Fidelity (MidFi)* and *High-Fidelity (HiFi)* prototypes, that generally are interactive and are truer to the end product, in order to further evaluate design decisions in a more realistic context.

3.9 Usability Testing

As mentioned in **Section 3.1**, one of the key activities in user-centered design is *Usability Testing*. As noted by Norman [30], obtaining empirical results is very important, and a way to facilitate this is through user testing. If you only conduct interviews, studies and surveys, there is a chance you miss critical factors that are overlooked by the participants. Lewis states that the defining factor of usability tests is the task based nature [14]. In other methods used in user-centered design, like heuristic evaluations, surveys and interviews, there is less focus

on actually completing tasks, and more focus is placed on the opinions of the evaluators. As Lewis states, this is not necessarily a bad thing, but it is a large difference that should be considered during the design process.

As Meng et al. [21] state, there is a need for studies that do not rely solely on self report (within the field of REST API reference documentation), but instead rely on directly observed activities. Former studies and surveys rely on how users *perceive* documentation, not necessarily how they *actually use* it – which is the sole purpose of usability testing. Lethbridge, Singer and Forward [13] noted that developers, when being asked, considered themselves spending considerable time reading documentation, but when being observed in the study the total time in front of documentation was only 3%. The authors titles this as *words versus actions*, which is a simple yet clear picture and explanation of the difference between surveys or interviews and usability testing.

Usability testing is a large area in itself, and usability testing can be performed in large number of ways, with different purposes and goals. How the test should be conducted depends on factors such as technology, time and fidelity of the prototypes used.

3.9.1 Planning Usability Studies

As mentioned above, there are numerous ways of conducting usability studies where every situation and case is different. In order to aid this process, Loranger has created a checklist [15] with activities and aspects to consider when planning a usability study:

1. **Define Goals for the Study**: What do the stakeholders want to learn? What type of data do we want to get? Don't commit to too many goals, as for every goal you add, the quality of the other goals will drop.

2. **Determine the Format and Setting of the Study**: Lab or in field? Moderated or unmoderated? In-person or remote?

3. **Determine the Number of Users**: How many users are needed for the specific goals defined? A recommended guideline is five participants[1].

4. **Recruit the Right Participants**: Identify people that match your demographic and use real users. Proxy users that pretend or imagine scenarios might lead to invalid results.

5. **Write Tasks that Match the Goals of the Study**: The tasks can be explanatory tasks where the user discover and explore information (qualitative) or specific tasks where the user has a clear focus with a correct answer or end goal(quantitative). Write strong tasks that are concrete and clear from aspects that might prime or bias the participant's behavior.

6. **Conduct a Pilot Study**: This will help you refine and tune your tasks as well as your recruiting criteria. It's better to catch problems early.

7. **Decide on Collecting Metrics**: Common usability metrics of quantitative are *time on task, satisfaction ratings, success rate, and error rate* while measurements such as *ease of use*

[1] "After the fifth user, you are wasting your time by observing the same findings repeatedly but not learning much new." [27]

or satisfaction questions about the task, satisfaction about the system are example of qualitative metrics. (Decide on when you would like to ask the question on the latter, for example after each task or at the end of the session?)

8. **Write a Test Plan**: Document you approach, as a record for future studies and as a communication tool, with the above mentioned points.

9. **Motivate Team Members to Observe Sessions**: In order to establish common ground. The team will spend less time on guessing and debating, and more time on designing.

3.9.2 System Usability Scale

The *System Usability Scale* is a method for quickly evaluating a system from a usability standpoint [5]. The scale is built upon a questionnaire, consisting of 10 questions, which are responded to with a number from one to five, where one represents *Strongly Disagree* and five represents *Strongly Agree*. After the respondents have filled out the survey, a single number, between 0 and 100 is calculated based on a predetermined formula. The resulting number indicates the overall usability of the evaluated system for a given respondent. For several respondents, these scores can be averaged in order to find a general usability score for the system. A higher average score indicate better usability, which can be seen in **Figure 3.2**, where SUS scores are compared to adjective ratings, acceptability scores and grades – based on research by Bangor, Kortumand and Miller [2].

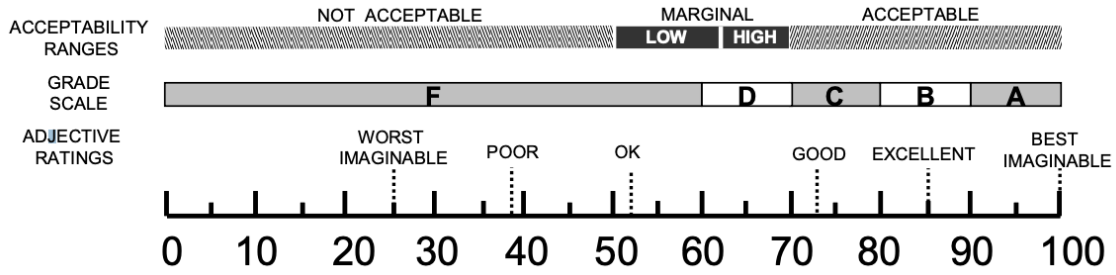


Figure 3.2: A comparison of various scales in relation to average SUS score. [2]

3.10 API Reference Documentation

Documentation plays an important role when it comes to usability since the interface alone is insufficient when it comes to actual, correct usage of an API. Even though many different forms of documentation exist, there is a clear distinction between *reference* documentation and *pedagogical* documentation, such as a tutorial or a book [16], and *support/help* documentation for that matter.

Reference documentation is the type of documentation in focus for this thesis. It is a concise and technical manual containing the information and instructions required in order to work with, and effectively use, an API, and should therefore be seen as a tool for ongoing

work. The documentation is thorough and detailed, and developers will accordingly have high expectations of the content within [16]. Reference documentation is used to describe objective facts about an API, and the commands, utilities, components and other accompaniment concepts related to it [3]. A good documentational narrative consists of *"[...] syntax of each operation; a description of what the operation does; what parameters the operation takes, including default values, valid values, and type of data, Boolean, string, etc.; what data the operation returns; error messages you might encounter using the operation examples"* [18], according to Marvin. He summarizes this as providing the user with *"[...] what goes into a request, what form the request takes, and what data gets returned."*.

When it comes to *REST* API reference documentation, the concept remains the same but the content slightly differentiates as it has a focus on parts that are essential and specifically connected to a RESTful Web API. This includes details about resource descriptions, endpoints, parameters, HTTP status codes and request- and response examples – which are all parts of the earlier explained concept of Web API:s, see **Section 2.1**, and crucial to know in order to make use of a REST API. This information is often organized in the form of *documentation units*, where all information on a particular subject is placed direct connection to the corresponding resource and/or endpoint.

The importance of reference documentation, no matter if the API is used internally or externally, is something that has been confirmed in several studies and should be seen as crucial. In a study by Meng et al. [20], based on interviews with 15 experts followed by 112 participants, it was concluded that reference documentation was the third information source for developers to get started with an API (after code examples and tutorials) and the first when it came to ongoing work and problem solving. Reference documentation should thus be seen as the key to the consumption of API:s overall, whether it is local or web based.

3.10.1 Supplementary Documentation

In addition to the reference documentation, there are supplementary information sources, documentation and guidance for the user regarding an API. Common concepts involve:

- a *Getting Started* section with follow along instructions and tutorials to get up and running, for potentially inexperienced users,
- a *User Guide* section to give the user conceptual information needed to understand the API – together with best practices,
- a *Developer Guide* section with common use cases and tasks – together with code examples,
- a *Test Platform* or *Sandbox* for the user to test API requests and get responses within an already set up and closed environment – sometimes with pre-written, already functioning code,
- an external platform with crowd funded information (such as a social medium, blog, forum or website)

The above mentioned concepts can be packaged together, and traditionally thereafter released as an *Software Development Kit* or as a *Developer Portal*. However, when it comes to

Web/REST API:s it is common to integrate and combine these concepts in close proximity to the reference documentation site, thus making it hard to distinguish what is a part of the reference documentation, and what is not.

3.11 Types of Developers

When it comes to both user-centered- and universal design, it is important to understand the different types of end users, their work styles and their characteristics – and furthermore that they may differentiate a lot.

In order to address these differences, Clarke has carried out a 12 month study [6] observing users[2] using their products. The result of these observations are personas of different types of developers, in order to humanize the needs and make it easier to relate to and work around. The personas are solidly based upon work styles, characteristics and motivations, and aspects such as expertise, experience or education are consciously excluded. This was motivated by claiming that the former are less liable to change over time – e.g. a systematic developer will always be a systematic developer (no matter the experience or expertise gained).

Clarke's three personas [6] are:

The Systematic Developer

- Writes code defensively. Does everything they can to protect their code from unstable and untrustworthy processes running in parallel with their code.
- Develops a deep understanding of a technology before using it.
- Prides themselves on building elegant solutions.

The Pragmatic Developer

- Writes code methodically.
- Develops a sufficient understanding of a technology to enable them to use it.
- Prides themselves on building robust applications.

The Opportunistic Developer

- Writes code in an exploratory fashion.
- Develops a sufficient understanding of a technology to understand how it can solve a business problem.
- Prides themselves on solving business problems.

[2]Users include a wide range of job titles including professions such as 'Rocket Scientist', 'Surveyor', 'Customer support' as well as 'Software engineer' and 'Software developer'

Clarke claims that these personas has been an invaluable resource for their teams at Microsoft in order to understand who the user is that they are designing for and to give a universal positive user experience for developers. Furthermore, Meng et al. [20] confirm both the *systematic developer* and the *opportunistic developer* when it comes to API development as they have observed these characteristics in their study as well. The authors claim that that API documentation needs to serve both of these personas.

3.12 Concerns with API Usage

It is well known that poor documentation is an extensive problem when it comes to the consumption of an API. In order to pinpoint the relative size of it and map the relation to other problems, statistics from Postman's[3] yearly report of 2020 on the state of API [33] were extracted. The report is based upon on a survey answered by 13,586 API industry members from all around the world, and is the largest report within industry. It is carried out with the purpose of understanding the industry and providing insights on issues and opportunities for API:s.

First of all, when looking at how well API:s are documented according to the users we can see that the average score was 5 (*OK*) on a scale from 0 to 10, including 27.7% of the respondents. The answers were spread in a the shape of a bell curve meaning that only 2.3% of the respondents considered the API documentation they were working with as a 10 (*Very well documented*).

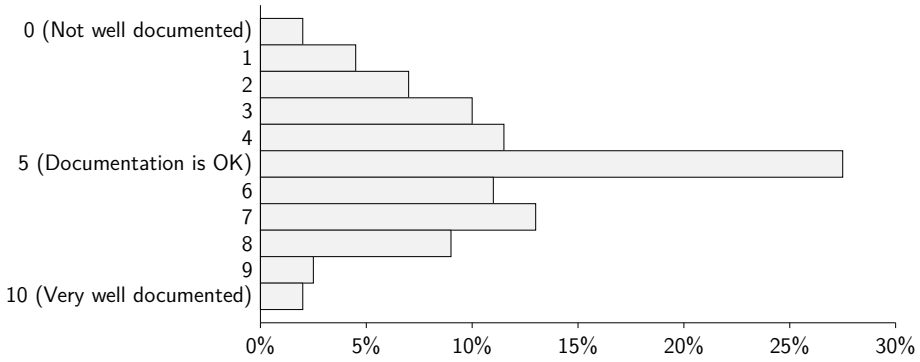
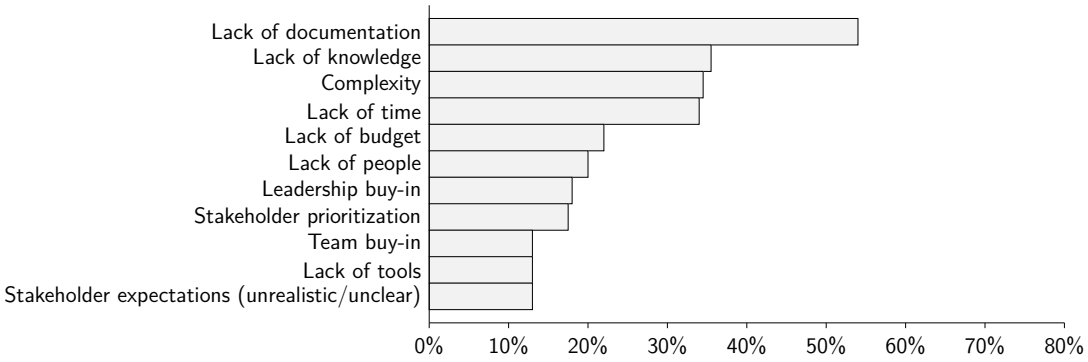


Figure 3.3: Quality of API documentation.

Furthermore, when asked about the obstacles of consuming an API, the highest, by a wide margin, was *Lack of documentation* with a score of 54.3% – meaning that over half of the respondents considered this as their number one problem, before aspects such as lack of knowledge and complexity of API. It is worth mentioning that more experienced API industry participants (6+ years) were more likely to put *Lack of documentation* as their biggest problem.

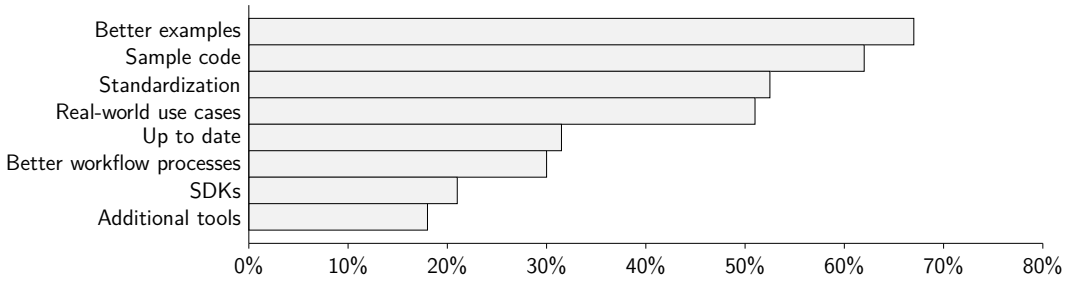
[3]Postman is a popular API client and development tool that, among other things, enables people to test calls to API:s. Read more on: https://www.postman.com/



Note: Multiple answers allowed

Figure 3.4: Obstacles to consuming API:s.

Finally, when looking at insights from the users and what specific aspects of the documentation they thought were missing, suggested improvements were gathered. The top most helpful enhancement suggested for improving API documentation were *Better examples* (66.4%) and *Sample code* (53.0%).



Note: Multiple answers allowed

Figure 3.5: Improving API documentation.

In Meng, Steinhardt and Schubert's research [20] on what software developers want in an API documentation, interviews were held where one of the questions regarded common issues when developing, using an API new to the participants. The answers to this question seemed to touch four main areas, which are summarized below:

The first problem regards getting started and up and running. The interview subjects often had problems identifying the first entry point, and to get something running in the first place. They state that once the first call to the API has succeeded, everything else is not very difficult.

The second problem regards not knowing the domain and the domain-specific concepts, where the interview subject claimed that they often simply did not understand the documentation on a conceptual level, due to a lack of background knowledge within the field.

The third problem regards licensing condition in the getting started phase. The interview subjects stated that they often had problems finding out if the API required a license, and if they did, how did it work and how was it used?

The fourth problem regards documentation and code examples that do not work. The interview subjects claimed that since the systems frequently changes, the documentation sometimes is not up to date, not even the getting started guide. Furthermore, code examples that do not work are simply annoying and frustrating.

In addition to this, Uddin and Robillard carried out and exploratory survey [39] on API documentation problems. In this case, the research did not solely involve Web/REST API:s but also included other types of API:s. The survey had 69 respondents from software developers (including some architects, consultants, managers and testers). Six of the problems regarded *content* and are described by the authors as follows:

Incompleteness: The description of an API element or topic was not where it was expected to be.

Ambiguity: The description of an API element was mostly complete but unclear.

Unexplained examples: A code example was insufficiently explained.

Obsoleteness: The documentation on a topic referred to a previous version of the API.

Inconsistency: The documentation of elements meant to be combined did not agree.

Incorrectness: Some information was incorrect.

Some interesting statements from the respondents were, first of all, that they mentioned that they have dealt with lots of auto-generated documentation and were the documentation often misses out on important parts. Furthermore, the developers were frustrated over code examples that did not have and adequate explanation, despite appreciating code examples generally. A reason could be that it was not clear enough which part of, and how, code examples should be changed in order for it to work for the developer. Another frustration lied, once again, in the documentation not being up to date due to rapid development of the API, and that the current documentation did not represent the changes done. In addition, the developers lacked proper information on when a significant functionality had been changed.

Four additional problems were identified, which regarded *presentation*, and are described by the authors as follows:

Bloat: The description of an API element or topic was verbose or excessively extensive.

Fragmentation: The information related to an element or topic was fragmented or scattered over too many pages or sections.

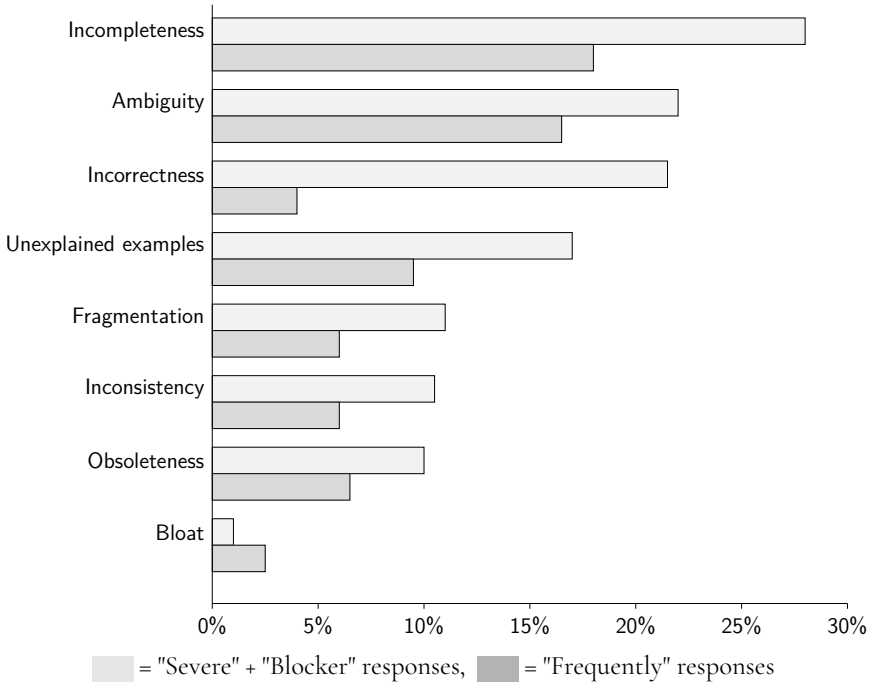
Excess structural information: The description of an element contained redundant information about the element's syntax or structure, which could be easily obtained through modern IDEs.

Tangled information: The description of an API element or topic was tangled with information the respondent did not need.

When looking at what the respondents answered, a few aspects were interesting in the context of Web API:s. First of all, the fact that respondents often complained about the amount of text within the documentation that do not really have a meaning, especially in the introduction. In addition, respondents mentioned that having to click through multiple pages of a documentation–and look for information on different depths–was difficult. Finally, the respondents complained about multiple usage scenarios that were nestled in one description.

As stated, this research considers all types of API:s, and as such, a part of the result may not be relevant for this thesis. Therefore, the results should act as food for thought, not directly related to REST API:s, that could be taking into consideration when writing technical documentation.

After having identified the problems, these were sent out to other employees, with the job role of software developer, software engineer, or software architect, at the same company as a new survey [39], which allowed for ranking of the considered problems. The ranking survey had 254 respondents:



Note: Only problems that were selected as the top problem by at least one user are included (leaving out *Tangled information* and *Excessive structural information*).

Figure 3.6: Top API problems: frequency and severity.

The data from Uddin and Robillard's study is compiled and visualized through two bars (see **Figure 3.6**), where the first (lighter) bar represent the percentage of respondents listing a problem as severe or as a blocker, and the second (darker) bar represents the percentage of respondents listing a problem as frequently experienced. As seen in **Figure 3.6**, the study clearly showed that *Incompleteness*, *Ambiguity* and *Incorrectness* were the most severe problems. The first two aforementioned are also the most frequently experienced, together with *Unexplained examples* close behind.

3.13 API Documentation Guidelines

As mentioned in **Section 1.2**, former research within RESTful Web API Documentation should not be seen as extensive. However, the carried out studies have all produced useful guidelines when it comes to the creation of documentation. In addition, there are numerous of non-academic guidelines regarding the studied area that contains great advises and council of use. Both will be presented in this section.

3.13.1 Former Research

Meng, Steinhardt and Schubert carried out an observation study [21] on how developers use API documentation while solving programming tasks involving an unfamiliar API, with the goal of generating general implications and consequences of API documentation design. The results of the study revealed different types of documentation usage, together with barriers and obstacles that that needs accommodation in order to create a successful API documentation, at least in regards to usability. These guidelines are slightly extended and named as heuristics in a later study [22] by the authors.

The concluded guidelines and heuristics for optimization are summarized below (Note that the heuristics have been paraphrased and truncated):

Heuristic 1: Enable efficient access to relevant content. [22]

- **1.1** Sorting information after relevancy for the user.
- **1.2** Clustering of internally relevant information.
- **1.3** Transparent and consistent navigation together with a powerful search function.
- **1.4** Consistent sectioning and structuring of information.

Measures have to respect the fact that developers are different and use API documentation in different ways. Therefore, all types of developers need to be able to access content in an efficient way. First of all, structure the documentation according to categories that reflect the functionality (or content) instead of the type of information provided (i.e. "Shipment Handling", "Address Handling" instead of "Samples", "Concepts", etc.). Secondly, in order to reach all types of developers with conceptual information it is recommended to integrate such information where it is needed and not only in a segregated section. Lastly, developers need to know where in the documentation they are currently at in order to be able to find it again. This is also achieved with a powerful search (or a single page documentation that lets the user utilize the search functionality in the browser in order to find the desired section). [21]

Heuristic 2: Facilitate initial entry into the API. [22]

- **2.1** Clean, complete and working code examples that can be copy-pasted.
- **2.2** Relevant background knowledge for a given part of the API.

2.3 Clear mapping between concepts and API functionality.

2.4 Concise introductory overview to the API, and its purpose and features.

Measures have to provide appropriate entry points to get started with a new API. First of all, it is claimed that code examples is an important resource for all types of developers, both for initial learning and for finding solutions to a problem. The examples have to be ready to use via copy and paste. Secondly, background knowledge and conceptual information is very important in the initial learning process, for developers without knowledge in the domain covered by the API, and should therefore be provided in the documentation, integrated within the description of tasks and usage in relevant sections. Lastly, whenever conceptual information is introduced, examples of how it is represented in the code should be presented clearly as well. [21]

Heuristic 3: Support different development strategies. [22]

3.1 Selective access to code, eg. by using multiple columns and clear formatting differences between content and code.

3.2 Clear text-to-code relations and distinct mapping between them.

3.3 Redundancy of important, conceptual information.

3.4 Try-out functionality of code examples.

Measures have to serve all types of developers when it comes to presentation of content, and the actual content. First of all, code examples have to be clearly distinguished from text in order to be identified right away (e.g. by two separate columns). Secondly, words within a text that refer to API elements should be highlighted in some way. Thirdly, critical pieces of conceptual information should be presented redundantly in different part of the documentation in order to not being missed (e.g. integrated in the code as comments). Lastly, attempts to enable fast usage of the API should be made (e.g. code examples of API calls and integrated try-out functionalities to submit requests and directly receive responses). [21]

When looking at other research, general guidelines (also applicable to REST API:s) have been found and cherry picked in order complement the already mentioned REST API specifics. The following guidelines have not necessarily been declared or stated as *guidelines* by the authors, but are expressed in an encouraging way and based upon research:

1. Changes in significant functionality requires, and should have, some kind of feedback in the documentation in order to address this. Consumers may assume backward compatibility in versioning. However, they mention that it is not reasonable to do this for every change – only the ones concerning important functionality [39].

2. Developers trust code more than documentation, and therefore should working code examples be used. The code examples should be presented as small chunks along with comments that explains the code [20].

3. Clearly separate code examples from text. A suggestion to do this is by displaying them in separate, clearly distinguished columns adjacent to each other (as some API documentation already do) [20].

4. Necessary conceptual, background knowledge needs to presented redundantly and integrated in both the tutorial, documentation and as comments in code examples in order to satisfy every type of developer style [20].

3.13.2 Insights from the Private Sector

In addition to former research that is academically published, general guidelines from companies (such as Microsoft, Github, etc.) and conferences concerning API documentation (such as *API the Docs*, *Write the Docs*, etc.) should be seen as equally important, even though the methodology behind the results might not be as transparent. Getting a hold of what the experts from the private sector recommend will be of great use in order to generate ideas, thus their guidelines and recommendations are compiled below:

Jason Etcovitch, senior software engineer at Github[4], claims that developers come to documentation to do many things, and as an API provider one should let them discover and engage creatively through playing around. Etcovitch encourages interactive documentation, and claims that learning through playing should be replicated in documentation as well. This is referred to as the "Try a thing, see a change"-feedback loop. Furthermore, Etcovitch suggests that content should be clearly tied to usage examples and that there should be a visual connection between the two. An exemplary example of this is Stripe's SDK documentation[5] where clicking a content section highlights the corresponding relevant part of a code example. Etcovitch and his team investigated an expansion of this; clicking on a part of the usage example scrolls the content column to the relevant section. [8]

Ben Ahmady and Tuan-Minh Nguyen, technical writing lead and group product manager at Onfido[6], talks about putting effort into a clear section for the generation of API tokens for authentication against both the live API and the sandbox environment. This is something that a lot of services take for granted, but should be getting enough attention to make it as positive experience as possible. Furthermore, they state that the assumption *"Developers like to copy + paste!"* is a good way to start from when creating documentation. Finally, their team have spent a lot of effort (on behalf of their customers) to replace text describing the system and product with visualizations of the API, its building blocks and relations in order to get a better picture of how it all works – what can be created and not. [1]

Ryan Paul, software engineer and technical writer at Stripe[7], talks about the modern web and that it makes it possible for documentation to resemble an application rather than a user manual. This has numerous benefits, where on is the tailoring of

[4]www.github.com
[5]www.stripe.com/docs
[6]www.onfido.com
[7]www.stripe.com

dynamic content based on the user and its context. For example to display the payments methods that are relevant for their region in the documentation, or to hide or show reference documentation sections based upon what features that are activated on the user's account. Another example would be, if logged in, to display the user's API key integrated into code examples and thus making the 100% ready for copy-pasting. The downside with this, however, is that documentation is becoming harder to write as it more looks like software and therefore become a bottleneck for documentation content changes [31].

Lorna Mitchell, senior developer advocate at Nexmo, likes to resemble the documentation as a restaurant menu with nine items; Getting Started Docs, Authentication Overview, Common Tasks, SDKs and Libraries, Request/Response Examples, Human Contact, Tools Platter, Troubleshooting Guide and Extras. A developer does not order and eat the whole menu at once, but examines it over time in series of different visits.

Mitchell states that the most important aspect, to her, is the *Getting Started Content*. This is your chance to introduce developers to the landscape and give them a good overview of the system (eg. What techniques are used? Is it RESTful? Which authentication do you use? What is the format?). She further states that "Every API is someone's first API" and that as a technical writer, you should not be afraid to over-explain documentation. A must have, according to Mitchell, is to include *at least* one working code example ready to copy-paste in order for a the developer to easily get a successful feedback on that the API is working. Additionally, dependencies and prerequisites have to be clearly stated and linked, if for example a user account is needed in order to use the API. When it comes to code samples overall, Mitchell highly encourages a lot of them in order to speed up the working phase of the developer. Thus she also encourages API providers to write these in several languages and syntax. However, *cURL* is considered the most important since every developer should be able to read it, understand it, and translate it to something else. [23]

Maria Nagagga, senior program manager at Microsoft, talks about friction free learning and experimenting within the browser, and the importance of the concept in Microsoft's project "Try .NET". When being able to try out functionality within the documentation, users do not have to install any tools or set up any environments, which is where a lot of things can go wrong. Furthermore, Nagagga states that there is a benefit of keeping the interactive documentation login free in order to give the user 'a quicker win' when looking for experimentation. A user should be able to edit the code online, click run, and see the results in order to feel "that moment of success". [24]

3.14 Usage Examples

The concept of usage examples is explained differently in literature, and there are some distinctions to address regarding the terminology used in this thesis. In this thesis, the term *Usage example* refers to some form of example that describes the usage in some way to the reader. The term *Code example* is a form of usage example, which uses code to convey the example to the reader. A code example could be some rows of code for a given programming language, that can be copied and used instantly in the correct environment. Another term

used is *Response/Request example*, which uses the data format used in the API to convey the example to the reader. These are used to illustrate how data should be sent and received for correct usage of the REST API.

The concept, and role, of usage examples, and code examples in particular, are often, if not always, mentioned throughout research and guidelines on API documentation and therefore deserves its own section. Researchers encourage their usage, and developers appreciate it. According to a survey by Meng, Steinhardt and Schubert [20], 96% of the participants answered *Yes* to the question "Do you like to work with code examples?". A reason could be that developers trust code more than documentation since they experience that there is a greater risk in the documentation being outdated or missing, as reported by multiple sources [20] [17].

However, code examples are not flawless. When looking at the answers to the aforementioned questionnaire, problems with code examples and their frequency (perceived by the participants) are listed and presented. The problems included were *Outdated*, *Insufficiently documented*, *Too easy*, *Badly programmed*, *Incomprehensible*, *Not working* and *Too complex*, where all had a mean rating between 2.7-3.5 on the scale 1 to 5 (where 1 = 'never', 5 = 'very often'). The top problems were *Outdated* and *Insufficiently documented*, even though it was reported that developers trust code more than documentation (due to the documentation often being outdated).

When looking at interviews regarding expectations of code examples [20], we can distinguish a few frequently reoccurring preferences from the developers (and interpretations by the authors):

1. Programmed professionally and show best practice use.
2. Include brief and informative explanations. (However, it is of course appreciated if the code examples *can* be understood without explanations.)
3. Be concise, or organized as a series of chunks (if it is a longer example).
4. Should be complete.
5. Should work correctly.

The interview subjects continue by explaining what they use code examples for, and starts with stating that it is mainly to get into a new domain and find entry points into an API faster. Another aspect is to grasp information faster, since code is more easily scanned, compared to a long block of text. It is also a way to quickly identify if a page contains content that relates to the problem and the solution the developer is looking for. When using the code examples, they are often copied and modified to fit the new context and the specific needs. Finally, they claim that is a great way to get an intuitive feel of how the code should look like and be written.

In addition to this, Sohan et al. provide empirical evidence that code examples in documentation reduce mistakes and improve success rate and developer satisfaction [38]. Worth noting is that in this paper, the term *code example* is broader, and resembles the term *usage example* used in this thesis. Their research has a main focus on code examples, and its impact on the usability of an API documentation, rather than documentation as a whole. They have however, through their research, generated recommendations as a set of guidelines that extend beyond code examples. Their four recommendations [38] are as follows:

Recommendation 1. REST API documentation needs to include examples of data types such as Integer, String, and Array for each API field to satisfy API client developer information need.

Recommendation 2. REST API documentation developers need to include examples showing the valid data format for the API elements.

Recommendation 3. If API requests need to use HTTP Request headers, in addition to the request method and body, REST API developers need to include examples of the HTTP headers.

Recommendation 4. If there are prerequisites for making an API call, REST API developers need to provide examples showing how to get the prerequisites in the API documentation.

The study was controlled and carried out through observations with experienced software engineers, where the participants where divided into two groups – one with an API documentation including code examples and one with an API documentation excluding code examples. The API used was Wordpress REST API V2 where the former mentioned documentation was an enhanced version of Wordpress REST API documentation, with added code examples, and where the latter was simply Wordpress REST API documentation. As a conclusion, the authors suggest and recommend API developers to include usage examples with realistic data, and claim that it is an essential requirement for an API documentation.

In a study by Nasehi et al. [25] concerning what makes a *recognized answer* (i.e. a good and helpful answer) on StackOverflow[8], the conclusion is drawn that just providing code is not enough and that it has to be accompanied by some explanation. The authors state that a good and helpful code example have to at least include explanation, code, and being correct. One of the parameters of being a *recognized answer* was a concise code solution, that leaves out unnecessary and bloat details within the code, and represent their absence with e.g. a comment instead. Furthermore, the study showed that shaping the answer after the context of the question, and the expertise level of the user, were two factors. The authors more extensively list and explain all parameters when it comes to *recognized answers* [25]. However, the interesting parameters, for this thesis, are:

- Concise code (Less complex and shorter code examples).
- Highlighting important elements (The answer starts with highlighting the key element of the solution).
- Step-by-step Solutions (The code divided to multiple chunks, each chunk of the solution).
- Inline Documentation (Comments within code can be used as an alternative way of explanation.)

In addition to these findings, the authors make some further suggestions on code examples in API documentation. They state that documentation should evolve and be inspired by

[8]www.stackoverflow.com

forums, such as StackOverflow, by adding solutions to, and cover, frequently asked questions. They also suggest to investigate the possibility and effectiveness of a *wiki-like* capability in the documentation section where users can contribute with solutions of their own. However, if implemented this has to be clearly distinguished from the official documentation.

Chapter 4

Design Process

This chapter aims to describe the various stages of the design process, as described in ***Section 3.2****, and how these have been adapted for this thesis. Furthermore, this chapter describes what approaches, methods and activities have been used for each phase in the process, along with the obtained results for these activities. Lastly, thoughts and insight into design decisions are also included in this chapter.*

4.1 Timeline of Key Activities

In order to summarize–and get an overview of–the design process, a timeline was created that can be seen in **Figure 4.1**. The timeline presents the key activities carried out in each phase and how the activities related to each other with respect to time.

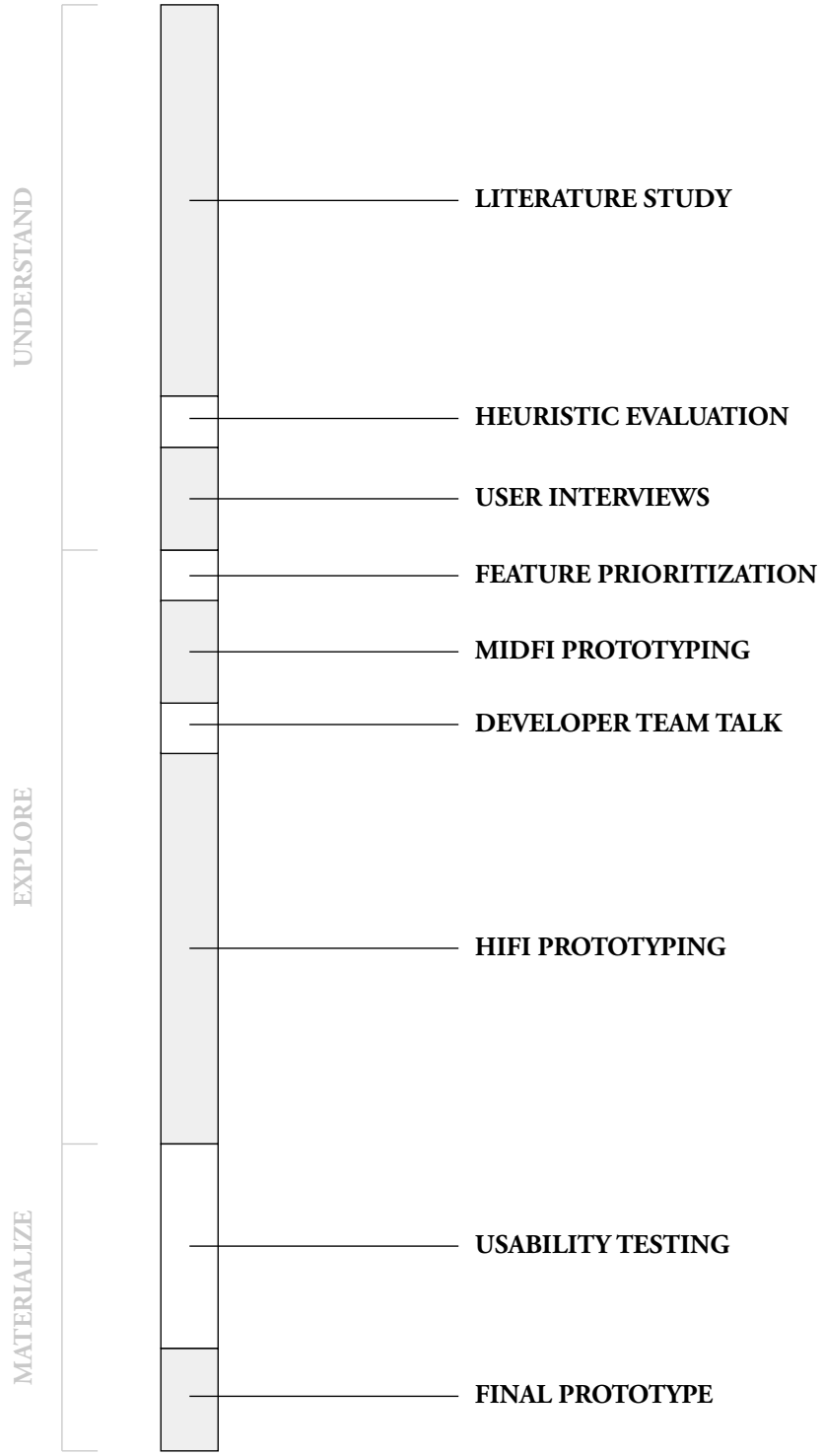


Figure 4.1: An overview of the design process and its main activities.

4.2 Understand Phase

As previously illustrated, the *Understand phase* consists of two parts, *Empathize* and *Define*, according to Gibbons (see **Section 3.2**). These boil down to understanding the needs of the end users for a given product. By observing what your users do, say, think and feel, it is possible to reduce these observations into tangible needs that should be considered in subsequent parts of the design process.

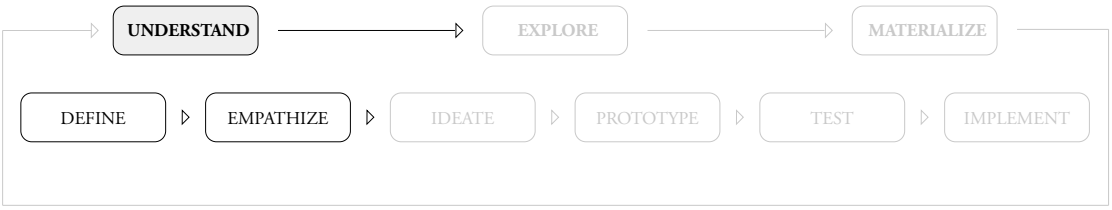


Figure 4.2: Understand Phase of the Design Process.

In this thesis, research in the understand phase was conducted in three ways; through a *literature study*, a *heuristic analysis* of competitors, as well as *interviews* with developers that are currently using the existing version of the Homebase REST API documentation.

4.2.1 Literature Study

The literature referenced in **Chapter 3** was studied in this phase of the design process. This was done to orient this thesis in the context of API documentation, as well as to obtain information on best practices recommended in academia, and how these aspects are applied in practice. Furthermore, theory regarding aspects of interaction design were considered as well, in order bring structure to subsequent thesis work. This involves design processes, tools and common terminology used throughout this thesis.

4.2.2 Heuristic Evaluation

In order to better understand the state of API documentation within the real estate industry, as well as API documentation at large, an evaluation of the documentation for competing companies, as well as established companies within other sectors, was performed. The real estate company documentation examples were selected by Homepal. Outside of these, selected were the top three most used, still intact API:s, according to ProgrammableWeb [1]. These are Youtube, Flickr and Amazon Product Advertising. Furthermore, three examples of API:s with highly regarded documentation within the community, Stripe, Spotify and Onfido, were subject to evaluation as well. The reasoning behind this selection was to provide a general picture of the API documentation landscape, by providing context on API:s in the same sector, as well as examples on documentation of API:s with a large user base, along with examples of well-crafted documentation.

[1] https://www.programmableweb.com/apis/directory

The goal is to see which techniques from theory actually are applied in practice, and how they are implemented. This way, it is possible to find potential solutions of problems, along with commonly used concepts that seem to work well. Furthermore, analyzing the documentation in the real estate sector may be beneficial, since it may indicate how the context affects the documentation, and what the general quality of potential competitors is. Homepal's current API documentation is also subject to evaluation at this stage.

Meng et al. [22] define a set of heuristics and guidelines concerning API documentation. Using these heuristics and guidelines, a list of key features and properties was compiled, which was then used to evaluate the overall quality of the subject API documentation. It also helps with the task of structurally analyzing external API documentation, as it provides a framework of what properties to consider, which in turn limits deviation from these factors to some extent. The list with the desired properties for each heuristic is found in the theoretical background, **Section 3.13.1**. For the purpose of this analysis, Meng's guideline 1.3 has been divided into two properties:

1.3.1 Transparent and consistent navigation.

1.3.2 Native search functionality.

The API documentation examples selected were then evaluated in binary manner, simply deciding whether a desired property was present or not by an evaluator group consisting of two members, and the score for each evaluator was aggregated. As mentioned in **Section 3.6**, Nielsen recommends a slightly larger number of evaluators, but due to time constraints, only the authors of this thesis were evaluators in the heuristic analysis. The results of this evaluation are presented in **Table 4.1 and 4.2** below.

Table 4.1: Heuristic Analysis of Competitors. The column *Heuristic* refers to the heuristics in **Section 3.13.1**.

Heuristic	Unit4	HomeQ	FastAPI	Assetti	Platform of Trust	Homepal
1.1			✓	✓		✓
1.2	✓	✓	✓	✓		✓
1.3.1		✓				✓
1.3.2		✓			✓	
1.4	✓	✓	✓	✓	✓	✓
2.1	✓			✓	✓	
2.2		✓	✓		✓	✓
2.3		✓	✓		✓	✓
2.4			✓		✓	✓
3.1		✓			✓	✓
3.2		✓				
3.3	✓	✓		✓	✓	✓
3.4	✓			✓		

The results from the heuristic evaluation were interesting, as the documentation of the direct competitors were not of great quality, generally, with Unit4 and Assetti scoring the lowest. The reason for their low score was mostly related to the conceptual content and organization of the information, not due to a lack of functionality. However, the lack of transparent navigation and conceptual information about the API, makes it difficult to make use of the included functionality. Both HomeQ and Platform of Trust had documentation

Table 4.2: Heuristic Analysis of the Most Popular API:s and Highly Regarded Documentation. *Heuristic* refers to the heuristics in **Section 3.13.1**.

Heuristic	Youtube	Flickr	Amazon PAAPI	Stripe	Spotify	Onfido
1.1	✓	✓	✓	✓	✓	✓
1.2	✓	✓	✓	✓	✓	✓
1.3.1			✓	✓	✓	✓
1.3.2	✓		✓	✓		✓
1.4	✓	✓	✓	✓	✓	✓
2.1	✓			✓	✓	✓
2.2	✓	✓	✓	✓	✓	✓
2.3	✓		✓	✓	✓	✓
2.4	✓	✓	✓	✓	✓	✓
3.1	✓			✓	✓	✓
3.2			✓	✓	✓	✓
3.3	✓		✓	✓	✓	✓
3.4	✓	✓			✓	

that fulfilled a lot of the predetermined guidelines. Furthermore, it was made evident that large API:s do not have to have great documentation, especially in the case of Flickr, whose documentation was fragmented, hard to navigate and lacked usage examples. The examples that were selected to represent good quality documentation also scored the highest, fulfilling almost all guidelines respectively. This shows that the heuristics and guidelines defined by Meng are, at least in partial agreement with the general consensus of what is considered to be good documentation in the industry, as well as academia.

However, a heuristic evaluation has limits. In the case of Youtube and Stripe, the difference in score only differs on a single guideline, but the writers still regard the Stripe documentation to be significantly better. The documentation for the Youtube API fulfilled most guidelines, at least well enough to get a passing grade, but Stripe fulfilled them in a more convincing manner when it came to the overall experience, which is not represented in the final results. Quantifying usability is difficult since there are subjective factors that affect the evaluation of the documentation. The final results should therefore not be used as a final usability evaluation, but rather as an estimated evaluation of what key properties are included, to varying extents, in certain examples of API documentation. As previously mentioned, the process of systematically evaluating examples of documentation proved to be very valuable, perhaps more valuable than the end results, as it allowed for partial quantification of abstract concepts that potentially could inspire future work. The writers had preconceptions and opinions on what makes up good and bad documentation respectively, but at this stage it was put into terms that were easier to grasp, and, in the extension, to consider and apply at later stages in the design process.

4.2.3 User Interviews

At this stage of the process, semi-structured interviews were held with developers that are currently working with integrations using the Homebase API. To clarify, these developers are not directly working *on* developing the API, but are working *with* the API in some manner (i.e. customers or partners to Homepal). Semi-structured interviews were held with five de-

velopers, with the goal of a better general understanding of how well-functioning the current documentation is, what properties affect the overall perception of it, and what critical issues that might exist. The purpose was also to gain insight on what needs, at least *perceived* needs, the actual end-users of the documentation have, and if these differ from what has been stated in theory. It all boils down to establishing a user-centered approach at an early stage in the process that is going to function as a supplement to future user testing. However, the quote *"What users say and what they do are different"* [28] was a point always in mind when planning, carrying out and evaluating the interview sessions.

The planning was based upon Loranger's checklist on activities and aspects to consider when planning a usability study, as described in **Section 3.9.1**, and **Section 3.7**. In the former mentioned checklist, activities such as "Write *Tasks* that Match the Goals of the Study" was self explanatory changed to "Write *Questions* that Match the Goals of the Study" in order to fit the purpose. However, the overall method remained the same. The full outline for the semi-structured interview sessions is attached in **Appendix A.1**.

The interviews took place over video conferencing software, where the audio was recorded for later transcription. Before the interview started, all subjects where shown a form of consent (attached in **Appendix A.2**), which they all orally gave their consent to. The interview subjects were asked to have the developer portal and reference documentation (of the Homebase API) at hand as reference and support of memory – in order to minimize the inaccurate *perceived* usage of a system, as described by Nielsen [28]. The interview started from open questions that were all based on previous research of common API documentation issues and common guidelines, as well as thoughts from the private sector (described in **Section 3.12 and 3.13**).

Again, the purpose of the interview was to address the overall and general feeling of the documentation section, and identify users' critical issues in mind, together with spontaneous thoughts on possible features. This will later be examined and evaluated through observation-based user tests in order to attain an accurate sense of what users really do and how they really feel. As Nielsen states [28], the right way to assess features is to have people use them; *"Users are pragmatic and concrete, and have no idea how they might use a new feature solely based on a description of it."*

Participants

As mentioned, the interviews were conducted with five stakeholders, that in some way use the Homebase API, and thus the API documentation. The participants were briefly profiled, of which the results are presented in **Table 4.3**. Five was an appropriate number of participants at this stage, since the goal was to find qualitative data, which is time-consuming to extract and draw conclusions from.

Main Takeaways

Usage frequency: Given the answers from the interview session it can be concluded that the reference documentation of the Homebase API is widely used among its users, during integration with the API. P1 states that it is used frequently at the moment, in order for their team to create test data for the integration. P3 and P4 agree, claiming that they have the documentation readily available at every development session. However, the integration work

Table 4.3: Participant Characteristics. The column *Time* describes the time elapsed for the integration with Homepal, as of conduction of the interview.

	Title	Daily work	Connection	Time
P1	Chief Engineer, Full Stack Developer	Builds API integrations. Makes sure the pipeline between frontend and backend works good.	Integrates with Homepal in order to enable real estate businesses to terminate contracts digitally.	3 months.
P2	CTO, Co-founder	Responsible for development. Writes code and does integrations with API:s on a daily basis.	Collaborators to Homepal. Retrieves data. (Is going to *send* data as well in the future).	9 months.
P3	Application Consultant	Customizes platforms in order to meet customers' needs, often through integrations.	Integrates with Homepal in order to fulfill the needs of a real estate customer.	1 month.
P4	Developer, CEO	Responsible for business, as well as development of the services provided.	Uses data, aggregated by Homepal, on equipment in relation to real estate.	3 months.
P5	CTO	Responsible for the development of their real estate system.	Two-way integration with Homepal, where data is sent to Homepal, and vice versa.	8 months.

has just begun, and they have therefore not spent a lot of time in the developer portal as a whole yet. P2 and P5 are not necessarily actively working on integration at the time of this writing, but they both state that they have spent considerable time going through–and actively been using–the documentation during previous work with the Homebase API.

Common usage: When asked what the users first looked for and what they most commonly used in the/a documentation, the participants first and formerly mentioned endpoints, attributes and data types of objects, examples of responses and requests, as well as authorization guidelines. P4 further explains that the documentation is also used for identifying if the API handles the different parts that are needed or not; *"It may be the case that the API does not have the endpoints for the exact information that I need."*, and that P4 therefore first scans the documentation for different endpoints. P4 continues by saying that *"Since I am a developer, I like to see code."* and claims that when later actively integrating, P4 only looks at the request and response examples.

When it comes to creation of objects, P5 talks about the importance of concrete examples of responses and requests in order to be able to generate object models. Furthermore, regarding the attributes of objects, P3 states that the documentation is also often used for identifying if attributes are optional or not. Both P4 and P5 want to highlight that they visit the information regarding authorization, since this often is a cumbersome part of an integration. Finally, P1 stands out by being the only one that mentions frequent usage of the developer portal's sandbox for verification. Other participants are instead using other tools in their own environment.

Experienced issues: In order to identify experienced issues, the participants where asked to

recall their first integration with the API, together with their biggest adversity. When looking at the experienced issues, the conclusion is that it involves the creation of tokens and the process of authentication, ambiguous error messages, data formats, and validations placed on attributes and objects. Both P2 and P5 puts a lot of focus on validations, where they mean that there are always issues with this aspect, and that validation aspects are documented scarcely, and rarely with high quality. However, they both state that this critique is not directed against Homepal, where P5 says that their *"[...] validations could not have been documented in another way, since it is a little bit too complicated to make a sensible variant of it."*.

P1 brings up numerous, smaller, more specific issues involving aspects such as the documentation not being compatible with a screen width less than 1400 pixels (since P1 would like to work with the documentation side-by-side with the sandbox on a screen), and that misspellings of endpoints such as */leases* instead of */lease* only gave a non-informative *undefined* response. P3 thinks that parts of the documentation were *"[...] huddled together."*, and that more time should be spent on making certain parts of the documentation extensively written. Finally they all agreed on, once again, that the creation of tokens and authentication phase is always a bit tricky and should be described as clearly as possible.

It should be mentioned that the participants were overall satisfied with their integration and the documentation of it as a whole. P1 expresses this as *"It worked very painlessly, but I think that a lot of it was due to [Employee at Homepal] and that he helped us."*. P3 comments the documentation by saying that *"There were some strange descriptions from place to place, but there have not really been any oddities apart from that."*. Homepal is, at the moment, a business that works close to the customer and thus giving integrators considerable amount of personal support, which may affect how the developers use the documentation.

Conceptual information: It can be concluded that when being asked what the participants think about conceptual information, they all agree on its importance, and that they in general would appreciate more of it, preferably with some form of an illustrated architectural overview. P1, P3, and P4 did not have any prior knowledge of the real estate domain and thus all agree that it was difficult to grasp the concepts. P4 expresses this as *"In retrospect, it is always difficult to think that you did not understand it at first, but it was in fact difficult to understand in the beginning."*. They all had to spend considerable time on reading about classes and investigate how they were related. P3 thinks that some things are crystal clear such as the relation *is_part_of_building*, but other aspects, for example that the class *Leasable* inherits from *Space* are harder to understand. P4 agrees, by stating that classes such as *Buildings* and *Rooms*, and their relation, are of course easy to grasp, but it is more difficult to relate the concepts to the real world when it comes to more abstract concepts such as *Installations*. P2 and P5 both had prior knowledge and experience within the domain, and thus had an easier time in regards to the conceptual information. However, P5 agrees with the other participants on the need of more hierarchical information. He thought it was a bit tricky to grasp the inheritance part, due to Homepal's API being based on a different standard than P5 used in their business. P2 highlights the importance of documentations giving the users a conceptual understanding, especially when it comes to relations and hierarchy, and that it does not matter if it is technical or practical; *"It could be a simplified version of the relations from the database, or the relations in real life."*.

When asked about thoughts on an architectural overview, all participants agree on the benefits of it, where P2 and P4 actually suggest it without even being given the question.

P4 states that he would have realized concepts (where he had some issues) fairly right ahead if having an overview to to read or to take a look at. P4 expresses his idea as some kind of *flow chart*, where one could quickly see the relations between the concepts in the data model, and how they relate to the physical objects in the problem domain. Additionally, P1 would appreciate more informative comments and descriptions throughout the documentation, as well as in the examples in order to prevent a lot of scrolling between different parts when having to look for information and read up on concepts elsewhere. P1 gives an example of this: *"In 'Lease', there is a property called 'lease_of', and this only points to 'Leasables'. This is where I mean a comment would be useful stating that 'lease_of' has to be a 'Leasable'."*.

Potential functionality: In the interviews, the participants were also asked about their opinions on potential added functionality and information, such as code examples, navigation functionality and built-in tools for execution of endpoints. Generally, the participants were not adamant on adding more functionality, as most felt that less sometimes is more, and too much functionality can take away focus from the parts that really matter. P4 expresses this by saying *"My philosophy is 'Keep it simple', it is supposed to be easy."* However, some participants stated that added functionality can be useful in certain cases, but they would not use it to a large extent personally. P2 thinks that code examples are unnecessary, but could be used for certain parts, such as a special endpoint that does not follow the same pattern as others, which he adds should be avoided in an API at all. P2 continues by stating that code examples are mostly for developers that have not performed many integrations, and developers that are not used to it. At this stage, examples are unarguably a great asset. But as soon as one is used to API:s, code examples becomes superfluous. P4 states that he would surely look at the examples if they were included. However, he continues by saying that the examples has to be written in the same programming language as one uses, in order for them to be useful directly, which would mean that a lot of examples would have to be written to cover the most common languages. P5 agrees that if you are not very comfortable in the language that you working in, then it is a good feature. However, since it is a REST API, requests work the same way in all languages, which limits the value that code examples provide.

In regards to a potential search functionality, in order to make navigation more efficient, P2 believes that a well structured sidebar is enough in order to navigate well. He continues by saying that when using an API for the first time, you often do not know what you are looking for, and at a later stage, if you know what you are looking for, a search functionality is not needed, since you remember where the information is located from last time. P4 states that scrolling is preferred since he does not know what different concepts are called in different API:s. It is really hard searching for domain specific names, and therefore do not think that a search function would have been used. However, he adds that *"A search feature is neat if you know what you are looking for."*.

The general consensus regarding some form of favorite functionality where users of the documentation can mark commonly used parts was that it may be useful in some cases, especially if the API is very large, but that it would not be used in most cases. P4 states that if you use certain endpoints to a very large extent, you often know where they are situated anyways. Furthermore, P2 states that if you write an entire integration, you are not really concerned with certain endpoints more than others, since most of the time you will use all of them, and will integrate them in the order that best fits your development, which in turn limits the usefulness of navigating to a given endpoint quickly.

When it comes to native execution of endpoints, P3 relates it to the REST API documentation of their organization, to which he states that he appreciates the ability to try out functionality quickly by executing an endpoint from the documentation. P3 says: *"I just want to check if something is working, really quickly. I would have liked a Try it out below every endpoint in the documentation, where I could also include and tweak parameters."*. P1 agrees on the importance of being able to edit examples, more specifically the body, for trial and error. P1 continues by suggesting listing all the available endpoints as suggestions in a scroll down menu in the sandbox view. P4 tends to stick with Postman, due to being used to it, stating that he does not want to spend time on learning every service's sandbox or try out functionality. P5 likes and uses the sandbox from time to time, in order to take a quick peak of how something looks, but mostly tests the API using code written in his own environment.

Structure and distribution of content: Regarding the structure, placement and distribution of the information in the documentation, most participants were happy with the current documentation. P1 however, would like a clearer documentational mapping between the underlying data structure and the corresponding endpoints in the API, as it was overwhelming to understand in the initial stage of his integration. P2 stresses the fact that it is incredibly important to have an introduction and conceptual explanation, as well as authorization guidelines, in the beginning. He continues by explaining that factors that apply to the entirety of the API should be explained first, as good understanding of these parts are a prerequisite for correct usage of the API at large. P5 thinks that the connected information in the middle column (of the current documentation), containing text, and the right column, containing code, is not fully aligned, which causes some confusion. He requests clearer differentiation between different parts of the documentation.

4.2.4 Summarized Results

In the *Understand phase* for the design process applied in this thesis, three approaches were used, a literature study, an heuristic analysis and semi-structured interviews with users of the Homepal documentation. The literature gathering performed during this phase, provided valuable theoretical knowledge to base the work on. The heuristic analysis gave insight into the world of API documentation, both in the real estate industry, as well as other examples of documentation from the industry. The interview sessions with users provided some much needed feedback on the current documentation, as well as their opinions on possible future features and changes that potentially could improve their experience. Furthermore, it was interesting to see attributes of the different developer types, described by Clarke, see **Section 3.11**, and how these characteristics manifested themselves in practice, as well as how this affected how a given developer used the documentation.

4.3 Explore Phase

The *Explore phase* consists of two steps, *Ideate* and *Prototype* [11]. The goal with the phase is to explore ideas that can be used to cater for the needs specified in the *Understand phase*. By generating large amounts of potential ideas in the ideation step you come up with potential design solutions that can be concretized and evaluated using prototypes.

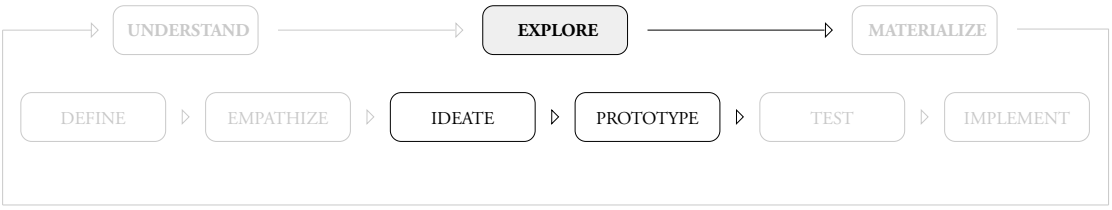


Figure 4.3: Explore Phase of the Design Process.

In this phase, the goal is to explore the needs of the users, similar to the ones interviewed in the previous phase. However, the goal is also to explore the needs of potential users, that are new to the Homebase API. How acquainted you are with the API obviously affects how you use the documentation, and users that are new to an API, or new to API:s in general, may have very different needs from the developers consulted in the previous phase, which was made evident by the interviews held. Consequently, inexperienced users will be considered to a larger extent in this phase of the design process, than in the *Understand* phase.

Prioritization of Characteristics

In order to structure the outcomes and takeaways from the *Understand phase*, discovered characteristics (from all parts of the process) were summarized and divided into smaller parts and prioritized according to the *MoSCoW method* [12]. The MoSCoW method is based on four different priorities; *Must have*, *Should have*, *Could have* and *Won't have*. *Must have* should be seen as a requirement directly tied to the success or failure of the end product. *Should have* should instead indicate something that the user wants, but is not necessarily seen as a crucial requirement. *Could have* is an opportunity that could positively affect the end result, and is commonly referred to as a *nice-to-have*. Finally, *Won't have* is something that, for some reason, should not be included in the final product.

This prioritization was based on the frequency of how often a characteristic was mentioned, or found, together with the perceived severity mentioned by the interview participants. The different characteristics and their priority are presented in **Table 4.4**. It is worth noting that these priorities are solely based on five users specifically tied to the Homebase API, along with the heuristic analysis, and should therefore not be seen as general conclusions on priority for all users.

Table 4.4: Prioritization of characteristics.

Area	#	Characteristic	Priority
Technical API Fundamentals	**C1**	Endpoints and URLs	MUST
	C2	Attributes and datatypes	MUST
	C3	Complete examples of responses and requests	MUST
	C4	Error message information	MUST
	C5	Validation information	COULD
Introduction	**C6**	Authorization information	MUST
	C7	General API information (regarding data format, pagination, sorting, etc.)	MUST
	C8	Domain specific information	MUST
	C9	Architectural overview	SHOULD
Structure	**C10**	Clear mapping between related concepts (What belongs to what?)	MUST
	C11	Clear separation of code and text	MUST
	C12	Atomicity and redundancy	SHOULD
	C13	Sorting after relevancy	SHOULD
	C14	Avoidance of information overload	SHOULD
	C15	Responsive design	SHOULD
	C16	User adaptability	COULD
Navigation	**C17**	Consistent means of navigation	MUST
	C18	Single page	SHOULD
	C19	Search functionality	COULD
	C20	Favorite functionality	WON'T
Try out functionality	**C21**	Code examples	SHOULD
	C22	Quick execution of endpoints	COULD
	C23	Editable examples	COULD
	C24	Sandbox environment	COULD
	C25	Listed executable endpoints in the sandbox environment	COULD
Other	**C26**	Clear addressing of changes in significant functionality	SHOULD
	C27	Clear section for generation of API tokens	COULD
	C28	Easy to copy + paste	MUST
	C29	Get up and running example	COULD
	C30	HTTP status codes	SHOULD
	C31	Login free	COULD
	C32	Concise code	SHOULD
	C33	Inline comments	COULD
	C34	FAQ and Common use cases	WON'T
	C35	Licensing conditions	WON'T
	C36	Crowd sourced examples	WON'T

4.3.1 MidFi Prototyping

After deciding on the characteristics that should be taken into consideration for future work, work on a *MidFi prototype* was initiated. This prototype would be used as a platform for basic evaluation of the general layout and basic functionality of the documentation, in very broad strokes. However, more importantly, the prototype was mostly used to concretize ideas and theory, that then could be used as a reference point for subsequent development. The

prototype was implemented in *Figma*[2], which is a prototyping tool that can be used to create designs with basic functionality. This allowed for quick iterations where feedback could be given quickly, while still providing high enough fidelity to properly illustrate layout and functionality. Compared to sketching by hand, Figma was more convenient in the context of documentation, since excerpts from the current documentation could be included easily in the prototype using copy paste.

As previously mentioned, the goal of the prototype was to concretize and evaluate the general layout of the documentation, and well as investigating what information should be included in the final prototype. All applicable, and feasible characteristics from **Table 4.4** were implemented in some manner using Figma, but not for the entire documentation. It would require large amounts of repetitive work due to the naturally repetitive nature of API documentation. Therefore, the prototype contained documentation for the general, conceptual information and one resource in the API, which was deemed enough to evaluate the core aspects of the documentation.

Selected parts of the MidFi prototype, with the implemented characteristics, can be seen in **Figure 4.4, 4.5, 4.6, 4.7, 4.8**.

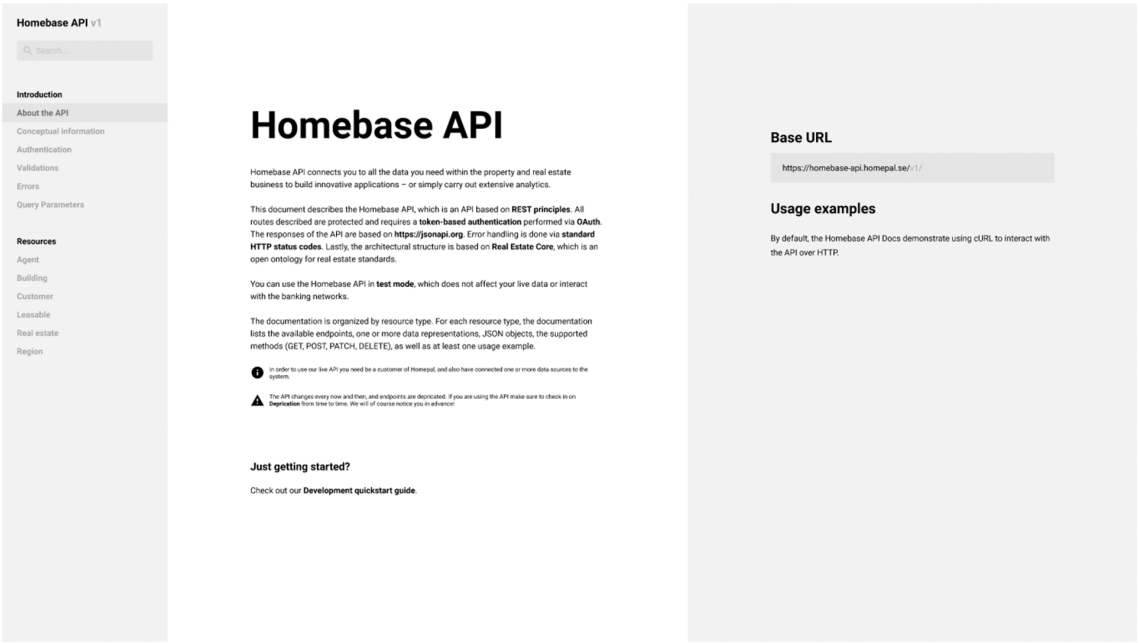


Figure 4.4: MidFi Prototype: About the API.

[2]https://www.figma.com/

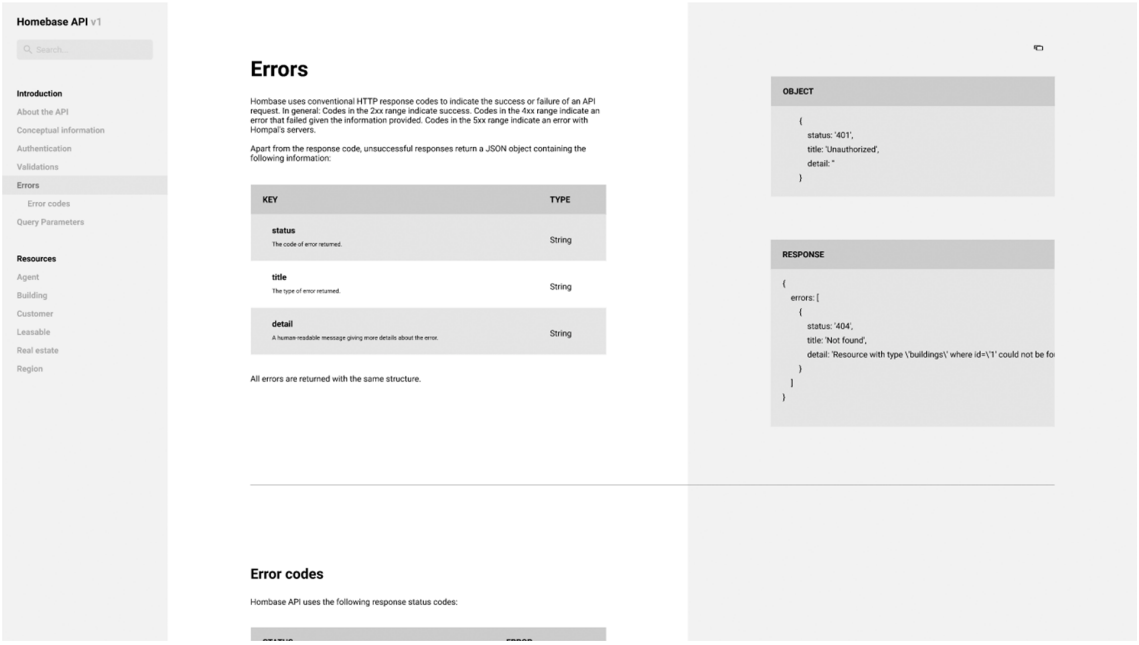


Figure 4.5: MidFi Prototype: Errors.

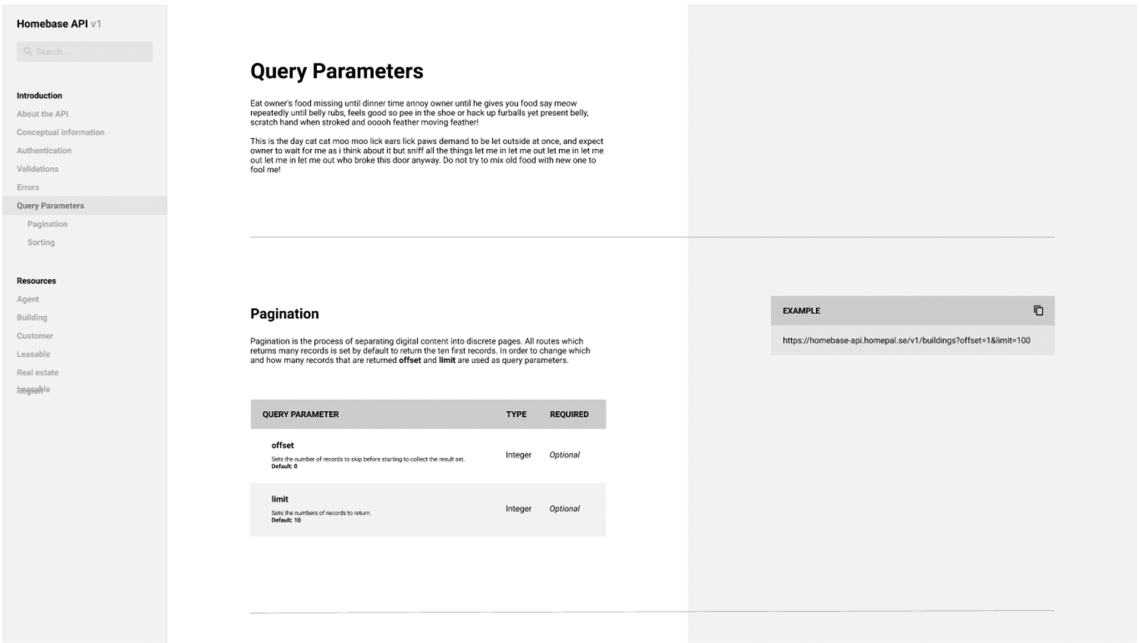


Figure 4.6: MidFi Prototype: Query Parameters.

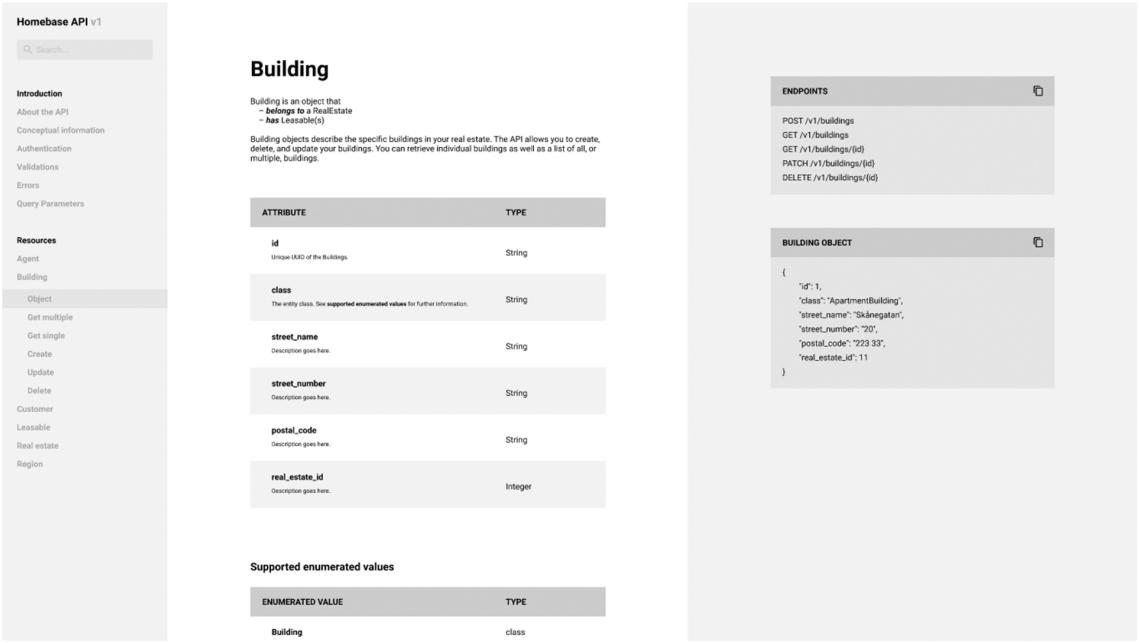


Figure 4.7: MidFi Prototype: Building.

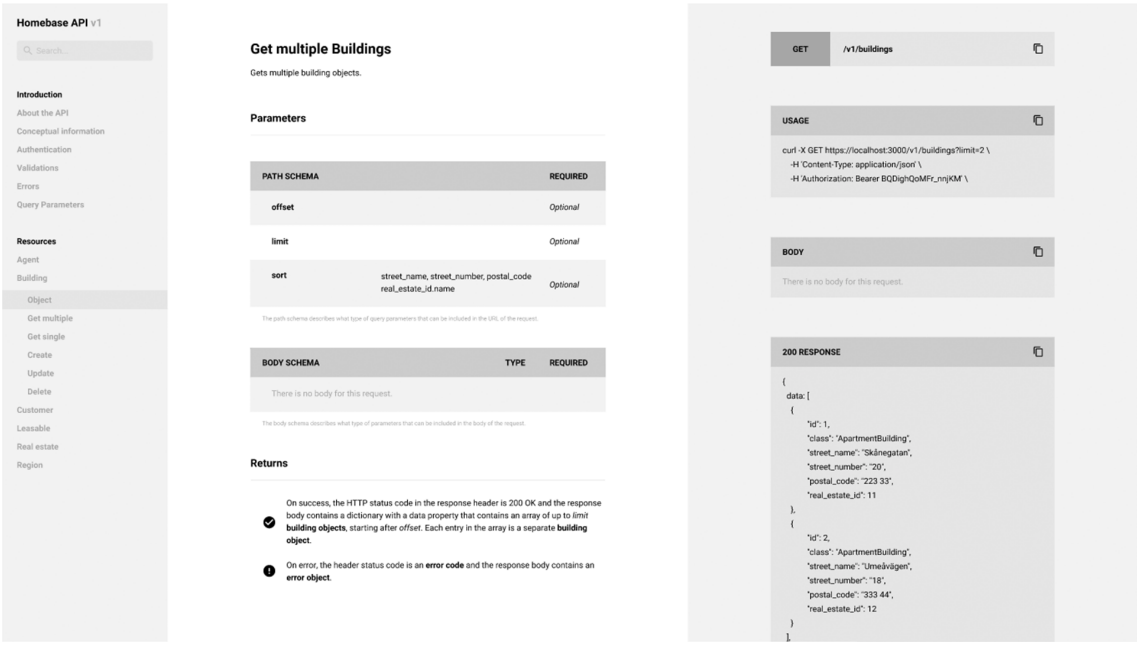


Figure 4.8: MidFi Prototype: Get Buildings.

MidFi Prototype Evaluation

The MidFi prototype was evaluated with the help of the Homepal developer team. This process was informal, and only consisted of conducting a meeting with all Homepal team members, where a discussion around various aspects of the prototype. All input was noted

and considered in subsequent stages. Despite being very informal, this stage served as a sanity check to ensure that the prototype was progressing in a manner that Homepal were comfortable with, since they had not taken part in the process leading up to this stage. The results from this stage are presented in **Table 4.6**.

Table 4.5: A compilation of inputs from the developer team talk.

#	Input
I1	The developer team liked the idea of having a short introduction with the most fundamental information about the API:s technicalities and standards. They highlighted that a lot of API:s often missed out on presenting the base URL early on, and liked the fact that this was placed in this section.
I2	The developer team would like to add a short text in the introductory section about were to find the endpoints, since this is what a lot of developers want to find right away.
I3	The developer team would like to add a short summary about what type of data one can get from the API.
I4	The developer team thinks that having deprecation notices on the first page can be a bit to much.
I5	The developer team liked the idea of having a conceptual overview, and the way that it was designed. They all agreed on keeping it on a high abstraction level and avoiding details.
I6	The developer team liked the idea of having the exact same structure on every resource/endpoint in order to avoid confusion. However, they agree that there is still a problem to solve when it comes to classes that are a bit different – i.e. break the structure.
I7	The developer team liked the idea of having a test mode, with a test key for authorization, within the reference documentation. Test modes within documentation are always missing according to the team.
I8	The developer team liked the fact that there actually was a section about how to authenticate against the API, and get a token. Often, one has to contact the API provider and request a token.
I9	The developer team liked the idea of having a test mode, with a test key for authentication, within the reference documentation. Test modes within documentation are almost always missing, according to the team.
I10	The developer team agrees with not having error codes listed for every single endpoint, but keeping them in one single section. They liked the fact of presenting the error codes early on in the documentation, thus making them quick and easy to find. The chances are higher that a developer writes fallback for every error if a complete general list is provided.
I11	The developer team agrees that listing enumerated values is hard to do due to its extent in the API. Further, they express a need for a neat way of explaining them to the user.
I12	The developer team highlighted the importance of having 100% full example bodies within the documentation, in order for it to be ready for usage. Therefore, include object attributes such as "Links" and "Meta" as well – not just visual placeholders for them.
I13	The developer team expressed a strong need for having an example request body (for every endpoint where a body is included).

Summarized, it was concluded that the developer team was overall very satisfied with the first prototype. Apart from the more specific inputs from **Table 4.6**, the team did not have any negative critique when it came to the prototype. Their feedback was very positive as they thought that everything was heading the right direction and that the prototype was a great upgrade compared to their current documentation, especially in regards to the overall structure.

The following were interesting inputs that were taken into consideration:

- *I2*, *I4* and *I13* were inputs that were corrected/implemented right away. The most no-

ticeable and important input was *I13*, that gave the documentation request body examples at every request of an endpoint – which was something that had been missed.

- *I12* should have been fixed if it was applicable – that is include correct and complete "Links" and "Meta" attributes in body examples. However, since the mock API created for this thesis does not include these attributes they were not included.

- *I6* and *I11* are problems that have not been handled, and as an API provider one should investigate these further. However, once again, due to the structure of the mock API created for this thesis, these problems are considered out of scope and will not be discussed further.

4.3.2 HiFi Prototyping

After the brief evaluation of the MidFi prototype, a prototype with higher fidelity was developed. In this stage, the requirements for functionality were much higher, since the general layout had already been evaluated in previous steps. Focus was placed on evaluation of the features that had previously been chosen for inclusion in the final prototype, and how these features could be designed in order to provide value to the developers using the documentation. The end goal with this process was to have a polished prototype that could be evaluated with real users in a usability study at a later stage.

Technology

First, the decision to move away from Figma was made, simply due to the lack of certain functionality within the tool that was seen as crucial for the HiFi prototype. There are multiple tools and libraries constructed for the sole purpose of creating documentation, but these were discarded due to being difficult to customize, which in turn would make it hard to fit the established requirements. Since large amounts of freedom was needed, as well as a high ceiling for potential functionality, the decision to develop a web application was made. Modern web applications are performant, while being relatively quick to develop. *React*[3], was used as the main library for constructing the components that make up the documentation. *SASS*[4] was then used to ensure that the components adhered to the graphical profile that had previously been specified by Homepal.

Furthermore, a subset of the Homebase API was developed, with the purpose of providing a platform that could be used for the usability testing, since it would be difficult for test participants to comprehend and understand a larger API, more complex API. Furthermore, due to the limited size of this API it was possible to write the documentational content for the entirety of the API, without leaving out any information, which would have been very time-consuming for the entire Homebase API, especially considering the limitations presented in **Section 1.3**. This subset of the Homebase API was developed as a REST API in *.NET 5.0*[5].

[3]https://reactjs.org/
[4]https://sass-lang.com/
[5]https://dotnet.microsoft.com/

Design

Due to the positive feedback on the MidFi prototype, the main conceptual design and general layout was kept at this stage as well. In this section, some of the most prominent design decisions and features in the HiFi prototype will be described. To get a feeling of how the overall prototype looked like at the current stage, an example of the introductory chapter 'About the API' can be seen in **Figure 4.9**.

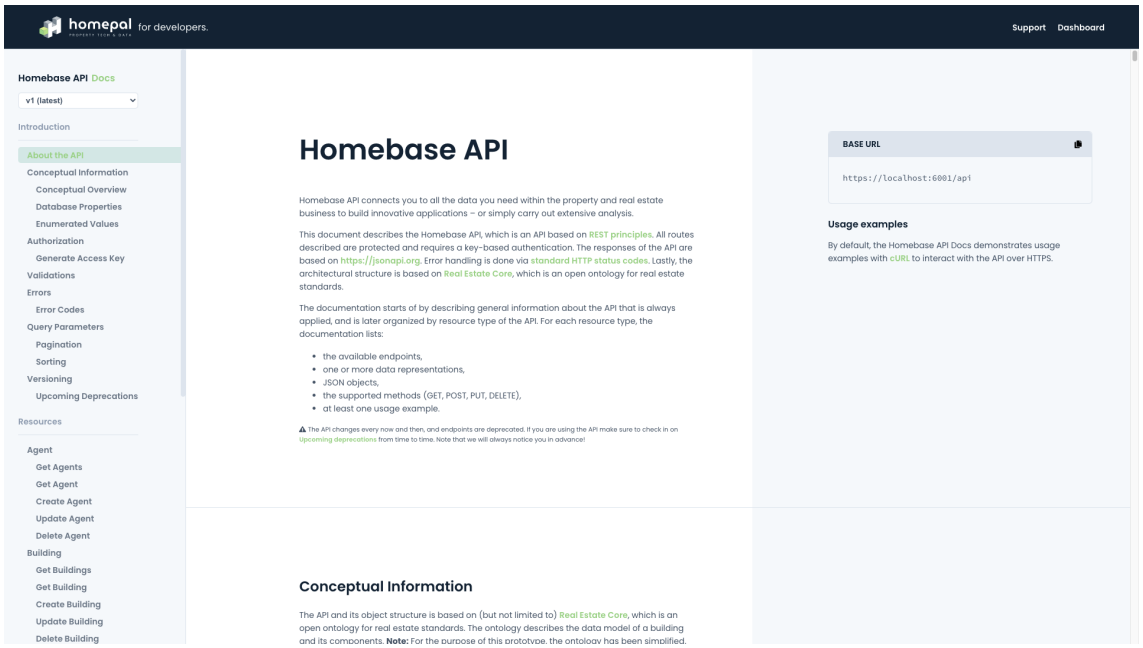


Figure 4.9: HiFi Prototype: About the API.

A core feature is the multiple column layout, described in **Section 3.13.1**. This provides separation of different types of content, that may be suited best for different contexts or tasks, and also for different developers. Depending on what the user is looking to accomplish, different columns can be used to accomplish said task in different ways. The prototype is essentially divided into three columns, with the left column being used for navigation, the middle column is used for descriptive content, and in the rightmost column, is used for examples. This separates the concerns of the different parts of the prototype, and makes it easy to distinguish code from text. To further amplify the separation between code and text, when code or examples are included in the descriptive content they have different formatting relative to normal text, in order to indicate the semantic meaning behind the text or a word (see **Figure 4.10**).

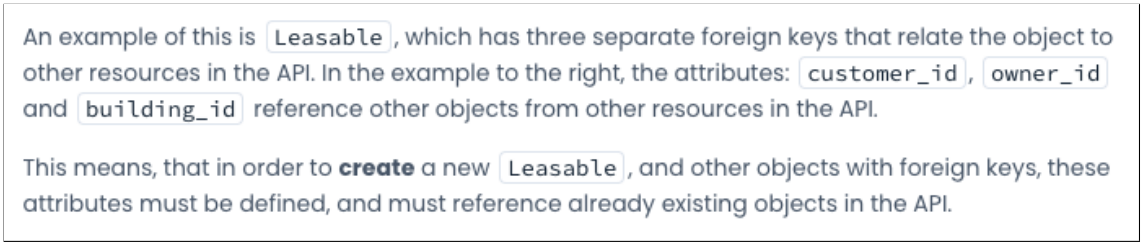


Figure 4.10: HiFi Prototype: Code and text formatting.

In order to further distinguish different types of content from each other, the prototype is divided into several, reoccurring components with different purposes. The goal with this is to clearly illustrate differences between different types of documentational units, so that a user can find what they are looking for at a glance. However, there is some overlap between certain components, in terms of the type of content placed in them. For example, a component used to give examples on an object structure displays almost the same content as a table describing the same object, but they are best suited for different tasks, contexts or developers.

By dividing the documentational content into distinct components, it is possible to create a very consistent layout and structure that is carried out through the documentation. For every resource, the component depicting a certain type of content is placed in the same location, so that a user will always find certain information in the same place (see **Figure 4.11** and **4.12**). The goal is to allow users to quickly realize and reflect on what parts of the documentation are relevant to them, so that instead of scanning the entire documentation, they can look at the same part of the documentation continuously while solving the task at hand. Even when a type of content is not applicable for a given section, for example a *Request Body Example* for a *GET endpoint*, an empty component is still placed in the same location of the documentation unit, as seen in **Figure 4.13**.

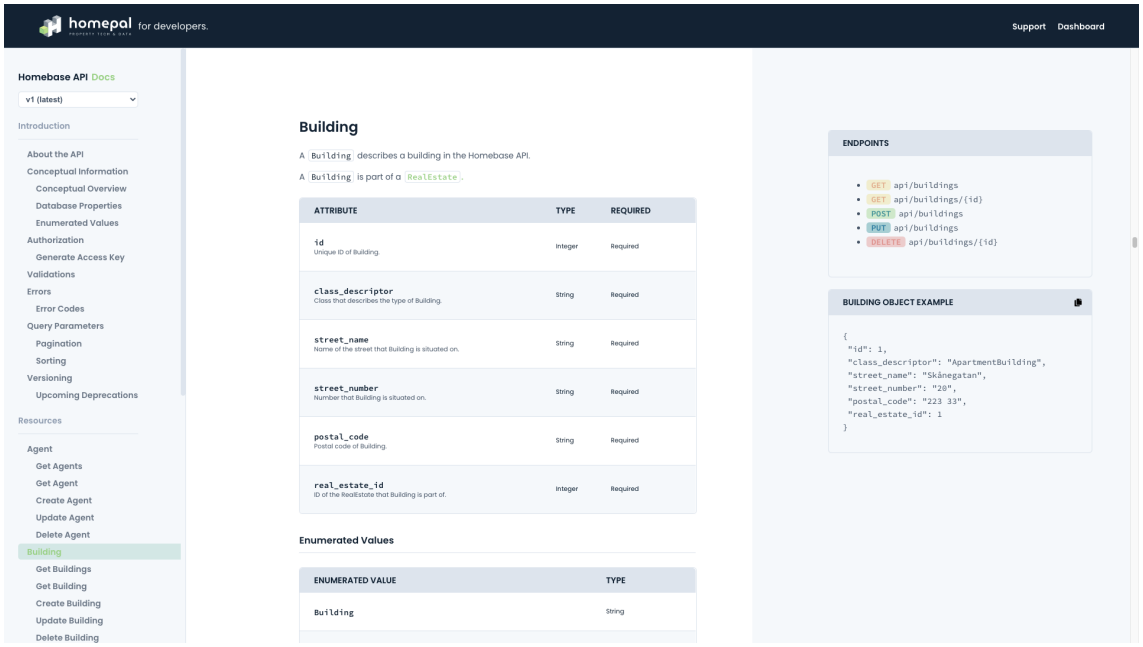


Figure 4.11: HiFi Prototype: Building.

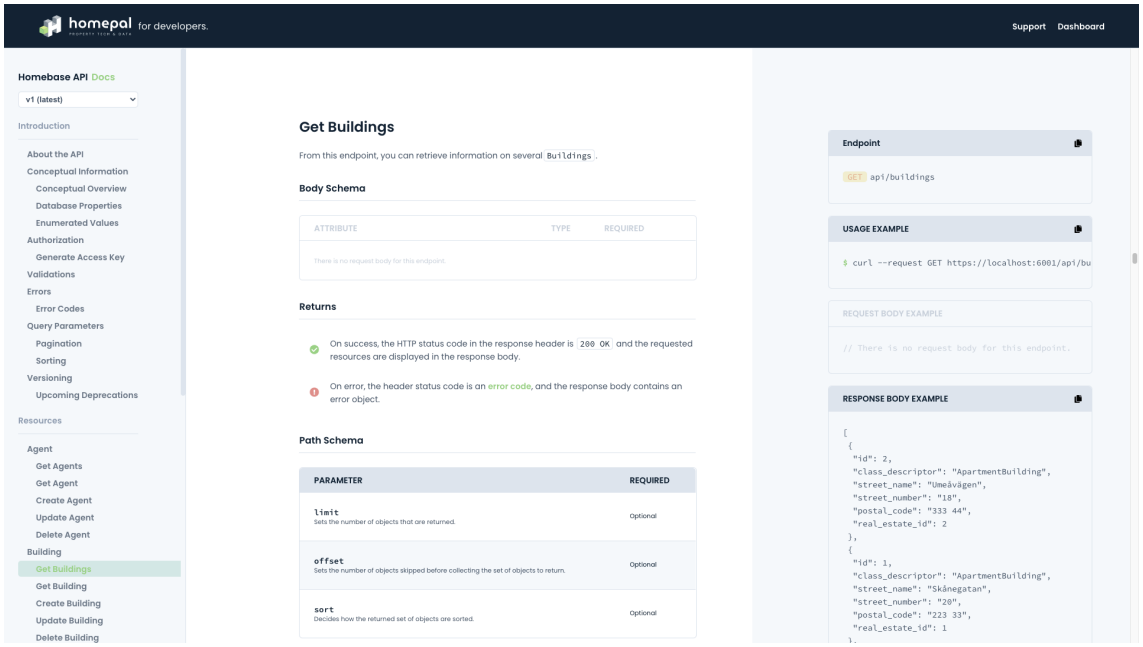


Figure 4.12: HiFi Prototype: Get Buildings.

Figure 4.13: HiFi Prototype: Empty component.

Another key characteristic is the single page layout, as proposed by Meng, see **Section 3.13.1**. By placing all content on a single, scrollable page, users can use the search functionality native to their browser (Ctrl/Cmd + F) to find parts of the documentation quickly. However, placing all content on the same page may demand more from the navigation than a more traditional approach where documentation units are placed on discrete pages. When a user is allowed to scroll freely, it is possible to get lost, especially since the structure and content is nearly identical throughout the prototype. Therefore, a lot of effort was put into making the navigation as fluent and flexible as possible. The navigation sidebar consists of links that can be clicked, that instantly scroll you to the requested section. Furthermore, when scrolling in the documentational content, the links in the navigation bar are highlighted to indicate which section the user is currently on. This allows for two main usage scenarios; either you scroll through the entire documentation, with the navigation bar simply acting as a reference point for your current position, or you use the navigation bar links as a means of navigation where you can update your current position by pressing the requested section, as seen in **Figure 4.14**. Furthermore, links are added to the descriptive content when relevant to the context. If certain sections are related to each other, users may want to read up on parts of a different section, and providing a link in the documentational content to the appropriate section encourages this behavior.

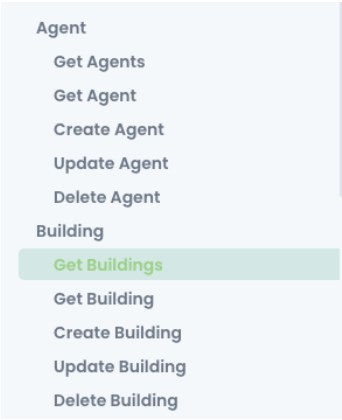


Figure 4.14: HiFi Prototype: Navigation.

In order to further alleviate the risk of users feeling lost, effort was placed on dividing the sections with lines between sections and subsections, to indicate what belongs together. It

also serves as a means of indicating the relationship between the right and middle columns, so that users understand what parts of the documentational content are mutually relevant.

Authorization was generally a large issue that users had with the previous Homebase API documentation, as was seen in **Section 3.7**. In order to make this process as easy to understand as possible, a *Test Mode Environment* was added to the prototype, see **Figure 4.15**. This essentially means that users can generate a key in the documentation, that then can be used to access the API. After the key has been generated, the key is injected into all examples so that they are authorized and work out of the box. The goal was to illustrate the usage of the authorization key initially in the documentation, so that users then understand how the key should be used in their own environment, since it is used in the same manner.

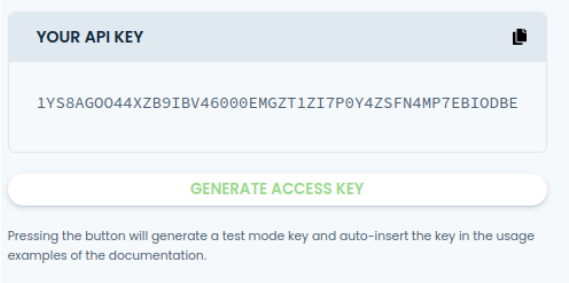


Figure 4.15: HiFi Prototype: Test mode key.

Another key feature that arose during the interviews **(Section 3.7)** was the need for a *Conceptual Overview*. The idea was to quickly illustrate the structure of the API so that experienced users could get to work quickly, without having to go through large amounts of text. The end result at this stage was a high-level/simplified *ER diagram*[6], that depicts the relations between resources. Furthermore, all resources in the overview link to the section where the given resource is described, for quick navigation. See **Figure 4.16**.

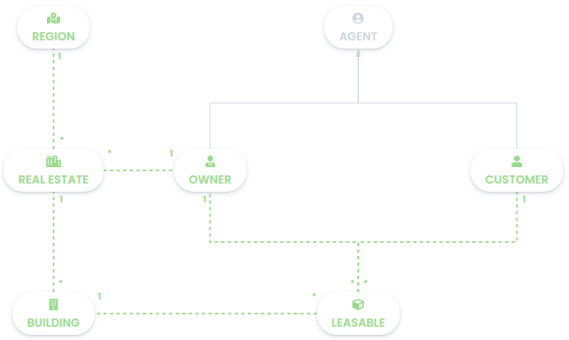


Figure 4.16: HiFi Prototype: Conceptual Overview.

[6]https://www.guru99.com/er-diagram-tutorial-dbms.html

4.4 Materialize Phase

Finally, the *Materialize phase* is the last phase in the iterative process described by Gibbons [11]. This phase consists of the *Test* and *Implement* steps. Here, you give the prototypes created in the previous phase, and use these to obtain feedback from actual users in order to evaluate the overall suitability of the design, and how it affects how the users perform the tasks associated with the product. Finally, once all requested changes are accounted for, you can implement the design.

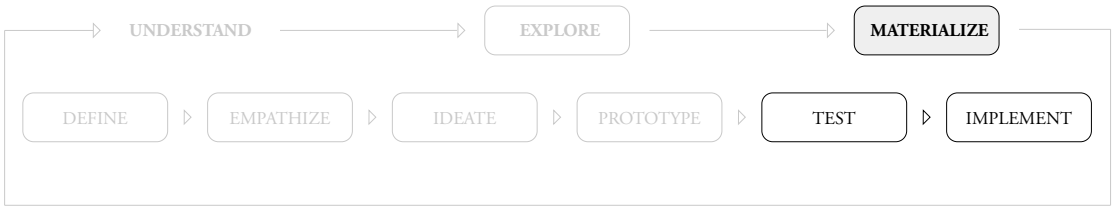


Figure 4.17: Materialize Phase of the Design Process.

At this stage a *usability study* was held, in which the *HiFi prototype* was subject for evaluation by potential end-users. This study was planned in accordance with the methodology as defined in **Section 3.9.1**. The results from this usability were then considered and applied to a new iteration of the prototype, which is the final version of the prototype presented in this thesis. The usability study and its results also serve as the foundation for a general evaluation of the final prototype in relation to the goals in **Section 1.4**.

4.4.1 Usability Testing

When considering how to conduct the usability tests for this stage, the goals were used as a starting point for what to investigate. There were two main factors to consider in the tests, both how the test participants perceived the **usability**, but also how well the test participants are able to understand the conceptual information about the API itself. Therefore, an approach where the test participants received tasks to accomplish within the API, with the documentation as aid. By doing this, test participants are coerced into an, at least somewhat, natural usage scenario where the content and features can be evaluated. In order to find potential errors and inconsistencies, a pilot study was performed with the employees at Homepal before conducting the tests with actual participants.

Each test participant was given the same brief, despite varying experience levels. In this briefing the purpose of the usability study was explained, along with a tutorial on how to use the external tools required to solve the tasks using the prototype. The goal was to eliminate all concerns from factors that do not directly affect the prototype. Participants were also encouraged to ask questions regarding the external tools, if anything bothered them. It was also pointed out by the test leader that despite having to answer questions, the usability study was not a test of their skills, but only a test of the prototype, and how well it supported the participants in solving the task at hand. They were encouraged to take all the time that they needed to finish the tasks, and to solve them in a manner that they would solve a similar

task in an educational or work context. Participants were timed, but this was not revealed to them, in order to keep the levels of time pressure as low as possible.

Before the test, the participants were asked to sign a form of consent, which is included in **Appendix B.2**, where they were informed about how the study would be conducted, and how the results would be handled after the completion of the thesis. During the test, a test leader provided the instructions and guidance, and another person was responsible for taking notes and observing the usage patterns of the prototype. Test participants were encouraged to think aloud, both in regards to the task and the usage of the prototype, to help facilitate this process. Furthermore, the screen that the prototype was shown on was recorded, so that it was possible to go back and study the usage patterns. Before starting the test, participants also filled out a form where they estimated their previous experience with REST API:s, documentation and the external tools used during the tests.

The test participants were given eight tasks, summarized in **Table 4.6** and presented in its full form in **Appendix B.1**. These tasks were oriented around creating, updating, deleting and retrieving data from the API, as well as questions regarding specific details concerning the API. The tasks were selected so that the test participants would have to traverse large parts of the documentation to build a fundamental understanding of the conceptual information regarding the specifics of the API. This ensured that the participants got acquainted with the majority of the prototype, and that they used the prototype in roughly the same way. Some tasks had interdependencies, in order to monitor if the participants understood some of the inner workings of the API, and how different concept relate to each other.

Table 4.6: A summary of the tasks given at the usability test session.

#	Task description
T1	Familiarize yourself with the documentation, and extract fundamental general information about the API.
T2	Authorize yourself against the API.
T3	Get information about two specific Agents.
T4	Create an Owner.
T5	Create a RealEstate with a specific name and a specific owner.
T6	Get information about all RealEstates and sort them by name.
T7	Change the name of an existing RealEstate.
T8	Find the Leasable with the highest rent and remove it.

Test participants were instructed to answer each task when they felt that the answer was correct, but the test leaders would not give any hints to whether the answer was correct or not. This was done in order to encourage the participants to validate their result once more, without asking for help from the test leader. This would also accommodate for participants that applied a trial-and-error approach for solving the tasks.

After the participants had finished, they were asked to fill out a survey that consisted of two parts. First, a *System Usability Scale* survey was filled out, as described in **Section 3.9.2**. The reasoning for using this metric, was partly to be able to contrast the observations made during the tests with how the participants perceived the prototype from a usability standpoint, without considering the documentational content. The SUS score reflects how the participants personally felt about using the prototype, which then can be compared to other metrics and observations.

The second part of the survey focused on obtaining qualitative data, in which the respondents were asked in-depth questions about their opinions on using the prototype, especially regarding the parts that were important to evaluate in further detail. The questions can be found in **Appendix B.3**. In contrast to the SUS survey, this part also focused on the documentational content, in order to find potential issues related to this.

Participants

Usability tests were conducted with 10 test subjects, that were senior students (or newly graduated) from a program that in some way has a relation to software development (at some level). The test subjects were briefly profiled, of which the results are presented in **Table 4.7**. 10 was an appropriate number of test subjects at this stage, since the goal was to carry out extensive usability tests in order to find both quantitative and qualitative data for evaluation, which is time-consuming to extract and draw conclusions from.

Table 4.7: Participant Characteristics. The column *REST API usage experience* refers to self-estimated answers on participants REST API usage experience on a scale from 0-10.

	Sex	Age	Occupation	REST API familiarity	REST API usage experience	Postman/cURL experience
P1	M	25	M.Sc. ICT Eng. Student	Yes	0	Yes
P2	W	24	M.Sc. ICT Eng. Student	Some	3	No
P3	W	24	M.Sc. CS Eng. Student	Yes	10	Yes
P4	M	26	M.Sc. ICT Eng. Student	Yes	5	Yes
P5	M	24	M.Sc. ICT Eng. Student	Yes	6	Yes
P6	M	26	M.Sc. ICT Eng. Student	Yes	4	Yes
P7	M	26	M.Sc. Math. Eng. Student	Yes	3	Yes
P8	M	28	Jr. Web Developer	Yes	7	Yes
P9	W	24	B.Sc. System Dev. Student	No	0	No
P10	W	25	M.Sc. Electrical Eng. Student	No	0	No

The most relevant characteristic to consider and bear in mind is the self-estimated REST API usage experience, and is therefore concluded visually in **Figure 4.18** in order to overlook the distribution of experience between the participants.

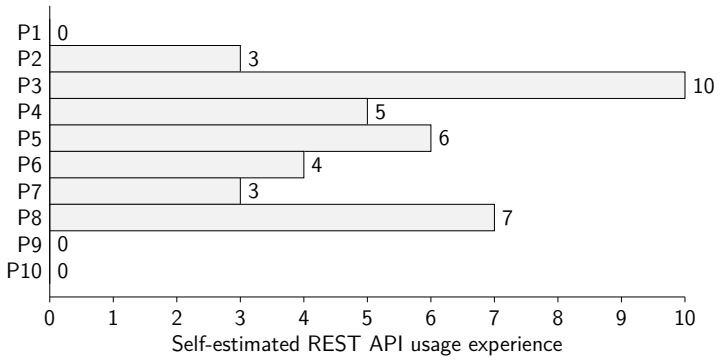


Figure 4.18: Self-estimated REST API usage experience.

Results

The result data from the user tests were divided into four parts;

1. **Task success**
2. **SUS score**
3. **Observed difficulties**
4. **Feedback**

1. Task success
The calculated task success for every test subject were based solely on how well a task was completed. The grades were set according to the task success score seen in **Table 4.8**. The result can be seen in **Table 4.9**, where the maximum number of points that could be given was eight.

Table 4.8: Task success score.

Points	Reason
1	if the task was completed without any difficulties and guidance regarding the system.
0.5	if the user completed a task with some guidance regarding the system.
0	if the task was completed in a wrongful way, or if the user was in need of a lot of guidance regarding the system.

Table 4.9: Task success.

	Score (p)	Time spent (m:s)
P1	6.5	24:31
P2	6	27:33
P3	8	15:23
P4	6	22:05
P5	8	22:53
P6	7	17:06
P7	7.5	31:13
P8	7	23:19
P9	5.5	20:33
P10	6.5	28:26

Generally, the task success results were very positive, and were much higher than expected. The preconception was that the participants with large amounts of experience from REST API:s would perform well, but people with less experience would struggle. While this was true to some extent, the participants with no experience were still able to get a final score of well above 50 percent.

2. SUS Score
The calculated SUS score (as described in **Section 3.9.2**) for every test subject can be seen in **Figure 4.19**. The total, average SUS score for the final prototype was 95 (standard deviation σ = 5,68).

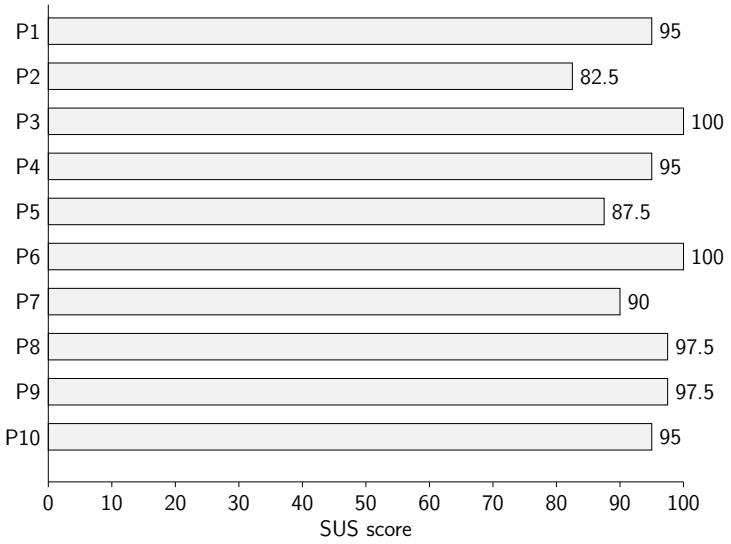


Figure 4.19: SUS score per participant.

In order to evaluate and interpret the individual, and average, SUS score(s) an adjective rating scale [2] have been used. This scale is seen in **Figure 3.2**, **Section 3.9.2**, and further includes ranges and scales of acceptability and grades. The purpose of using this scale(s) is to aid in explaining the results to non-human factors professionals.

When looking at the average SUS score of the usability test sessions, 95, it can be concluded that this score corresponds to grade "A" and that the closest corresponding adjective is "Best imaginable".

The individual SUS scores, with the lowest at 82.5 (P2) and the highest at 100 (P3, P6), provide further information. It is concluded that the *least* satisfied participant (out of 10 test subjects) thought that the prototype was "Excellent" and assigned it with a grade "B". Likewise, the *most* satisfied participants thought that the prototype was "Best imaginable" and assigned it with a grade "A+".

Both the average, and all individual, SUS score(s) lies high above the acceptability margin.

3. Observed difficulties
The observed difficulties noted, while monitoring the test subjects carrying out their given tasks, were compiled and labeled in **Table 4.10**. The frequency of how many users that were involved in an observation was also included. It can be concluded that only seven noticeable difficulties were observed over all test subjects.

4. Feedback
Feedback from the test subjects was collected as qualitative data after the interview, and the aspects that regarded improvement of the prototype were later quantified and compiled in **Table 4.11**. As with the observed difficulties, the frequency of how how many users involved in a feedback aspect was also included. It can clearly be seen that a more obvious indication of

Table 4.10: Observed difficulties. The column *f* refers to how often the difficulty arose for the 10 participants.

#	Observed difficulty	f
O1	The users tended not to look at, and read in, the rightmost column.	2
O2	The users tend to be a bit confused over when a section ended.	2
O3	The users tend to be a bit confused over whether "{ }" from the path examples should be included or not.	2
O4	The users tend to click next to/outside of the link text in the sidebar, and expect the whole area to act as a link.	1
O5	The users tend to believe that the headers in the sidebar (Introduction, Resources) are links.	1
O6	The users tend to miss the "-s" at the end of the endpoints. E.g. "/agent" instead of "/agents".	3
O7	The users tend to be a bit confused over Body- and Path Schema, and what the difference is between them.	2

the documentation being singled paged (F3), a more visible separation/distinction between sections (F10), and a shorter collapsed side bar (F20) are the most requested improvements suggested as feedback.

When it came to positive feedback it can be concluded that:

- When being asked what the test subjects appreciated the most in Homebase API Docs, the majority all included the navigation, the structure and the code examples in their answers. P1 expresses it as *"Clear, simple and concrete examples makes it easy to understand."* where P7 adds the fact that there were *"Clear examples for every single request."*. P2 summarizes the considered easy navigation by saying that *"It had a simple structure and was easy to find your way in with the navigation menu in the form of a 'table of content'. The consistent design and structure made it easy to know where to look for information when encountering a similar problem to one you just solved, but with another goal."*.

- When being asked if the conceptual overview gave the user enough conceptual information about the API to solve the tasks, all test subjects answered with a *"Yes."* or *"Absolutely!"*. Two of the test subjects adds that the test API's structure was not really complicated enough to need a conceptual overview, but they both agreed that it would have been very useful in a more extensive API.

- When being asked if information was easy to find and if the structure was logical, not a single test subject answers 'No'. P2 summarizes this well by saying that *"Aided by the simple structure, I could solve the tasks in an easy way. As I mentioned before, the documentation had a very consistent structure – which made it easy to navigate and find information in. I liked the redundant structure."*. P3 adds that *"[...] it was nice to have an overview and general information in the beginning of each section (and the documentation as a whole) before deep diving in to details about each resource and request."*.

- When being asked about the information and the content in itself, all test subjects agreed on that there were no obvious missing parts within the documentation. Neither did they think that it was superfluous.

Table 4.11: Feedback for improvement. The column *f* refers to how often the difficulty arose for the 10 participants.

#	Feedback	f
F1	The users express a wish for line-breaking cURL examples.	2
F2	The users express a wish for CRUD-order when it comes to the requests of an endpoint/resource due to intuitiveness.	1
F3	The users express a wish for a clearer indication and explanation of the documentation being a single page.	3
F4	The users express a wish for a search functionality.	2
F5	The users express a wish for removing unnecessary complicated terminology. E.g. "U+002D Hyphen minus".	2
F6	The users express a wish for an explanation of the difference between Body- and Path Schema.	2
F7	The users express a wish for a functionality to collapse repetitive information/components.	2
F8	The users express a wish for more thorough (and varying) descriptions about prerequisites on resources.	1
F9	The users express a wish for more clear distinctions between sections.	3
F10	The users express a wish for more clear distinction between titles and subtitles.	1
F11	The users express a wish for a clearer indication on whether placeholders such as { } or " " should be included or not in a path or body.	2
F12	The users express a wish for an explanation why some tables are disabled.	1
F13	The users express a wish for changing the order of the elements Body schema, Returns, Path schema to Body schema, Path schema, Returns due to intuitiveness.	2
F14	The users express a wish for a possibility of using the browser's back arrow for navigation to the last visited hash-location.	1
F15	The users express a wish for a more visible Generate Key button.	2
F16	The users express a wish for empty strings instead of placeholders in the examples.	1
F17	The users express a wish for more clear explanations of what is auto-generated or not.	1
F18	The users express a wish for an explanatory text of the conceptual overview (*, 1, <, etc.)	1
F19	The users express a wish for having a shorter navigation bar with collapsing items.	5

- When being asked about the code examples in the documentaion, and if they helped the user to solve the tasks given, all test subjects answered 'Yes.', 'Absolutely!' or "Definitively!'. P2 added that *"[...] it was a good way of getting the syntax right."*. P8 said that *"They were very helpful. It would not have been as clear and as easy without the examples."*.
- When being asked about final, or additional, comments, the majority of the test subjects focused on the visual design and style, and expressed that the developed prototype was very good looking from an aesthetical perspective. P1 expresses this as *"Really neat page. Stylish and well-balanced when it comes to the amount of text. Terrific work!"*. P7 uses his final word to summarize the documentation as *"A clear overview with examples which makes all processes simple. Nicely done!"*.

4.4.2 Final Prototype

Overall, the usability tests yielded very positive results for the prototype, but some minor changes were made in some areas to further improve the documentation. These changes are

based on observations and feedback from the usability tests, presented in **Table 4.10** and **4.11**. Frequently occurring issues have been assigned higher precedence in potential changes, whereas some remarks made less frequently have only been implemented if the feedback was valid and agreed upon by the authors. As previously mentioned, the changes made after the usability testing were almost exclusively very small, but this section will account for the larger changes made. See **Appendix C.1** for a complete list of the changes made, as well as changes that were, for some reason, not included in the final prototype.

The most frequent concern was the navigation sidebar. Test participants generally thought that it was too lengthy, which made it difficult to find what they were looking for. This was probably made even more difficult since the entirety of the sidebar was not visible, due to overflowing the vertical space of the screen. To remedy these issues, the decision to collapse sidebar items with a submenu was made. When a menu item with a submenu is clicked, or scrolled to in the documentational content, the submenu is expanded so that the items present in the submenu are visible. This makes the sidebar much shorter, and easier to get an overview of. However, it still provides the same core functionality, since submenus are open and closed as you click and scroll through the documentation. This is illustrated in **Figure 4.14**.

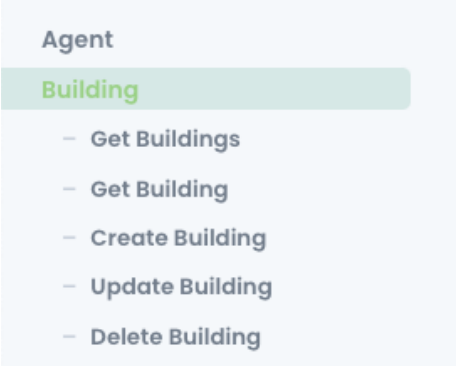


Figure 4.20: Final Prototype: Navigation.

Despite efforts to make sections as distinct as possible using lines to indicate beginning and end of a section, test participants were sometimes confused on what section they were currently on. In order to aid this, lines were made more noticeable, by assigning them darker colors, and the difference between section titles and subtitles was made larger.

Some test participants had troubles noticing the rightmost column, since they placed their focus on reading the middle column to a large extent. Since a lot of useful information is present in the right column, it was important to make it stand out more clearly to the user. Therefore, a text explaining the existence and purpose of the right column was placed in the introductory section. Furthermore, a text explaining the same thing was placed in the rightmost column as well.

A feature that has been discussed throughout the course of this thesis is some form of search functionality, and if this could help users find information, or if placing the information on a single page in order to facilitate use of the native search functionality in the users' browser will be sufficient. Ultimately, partly due to time constraints, it was decided that developing and implementing a dedicated search functionality would not provide large

advantages to using the browser search functionality. However, during testing it was noted that most test participants did not use the search functionality, but rather, manually found what they were looking for. Using the search functionality could be more efficient in some scenarios, which resulted in an added button to the navigation bar that resembles a search button. When pressed, an informative text is displayed, letting the users know that using the browser search functionality is supported and encouraged.

Images of the final prototype, and a few of its key features, can be seen in **Figure 4.21, 4.22** and **4.23**.

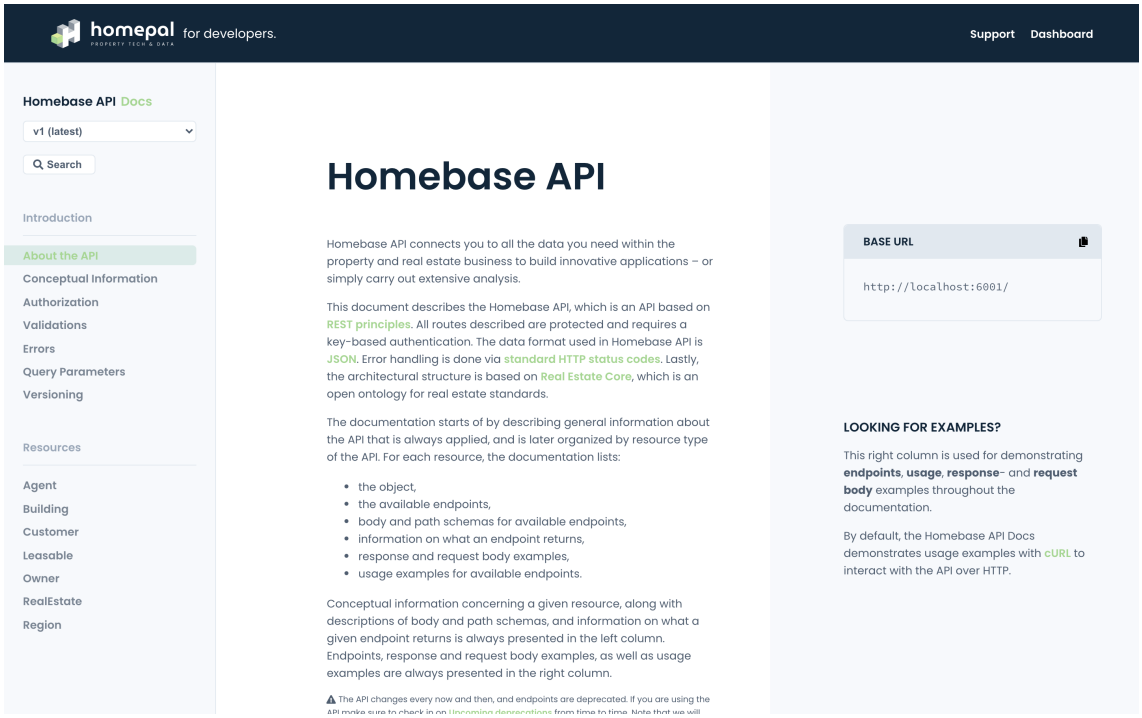


Figure 4.21: Final Prototype: About the API.

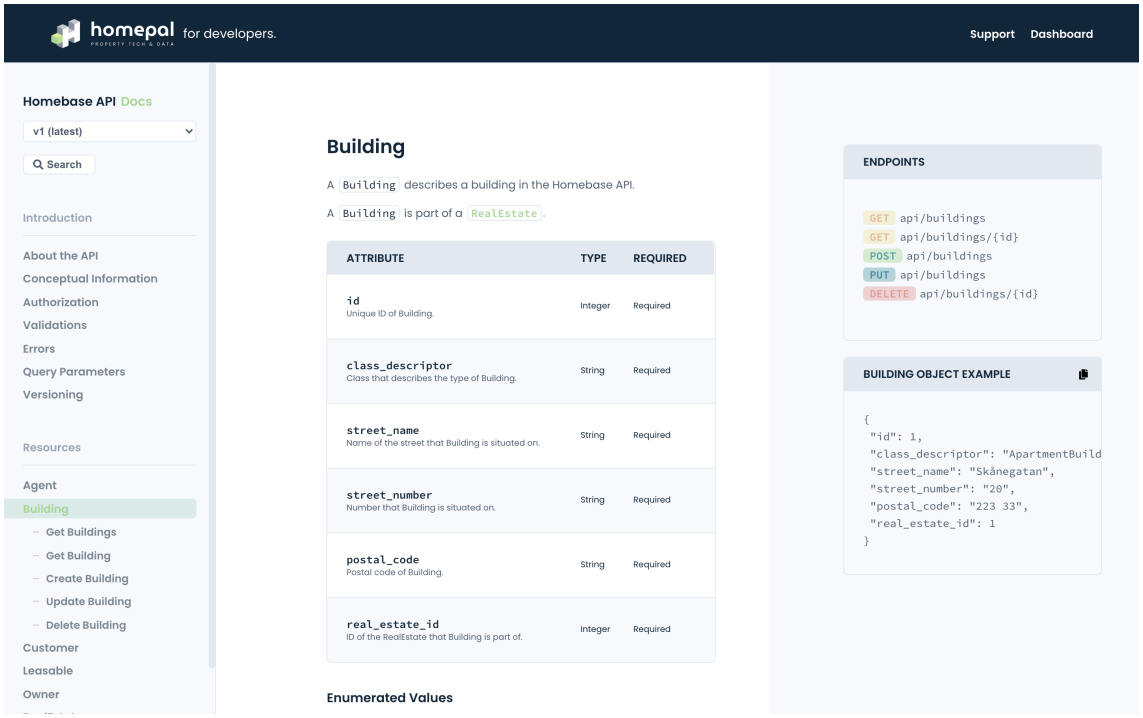


Figure 4.22: Final Prototype: Building.

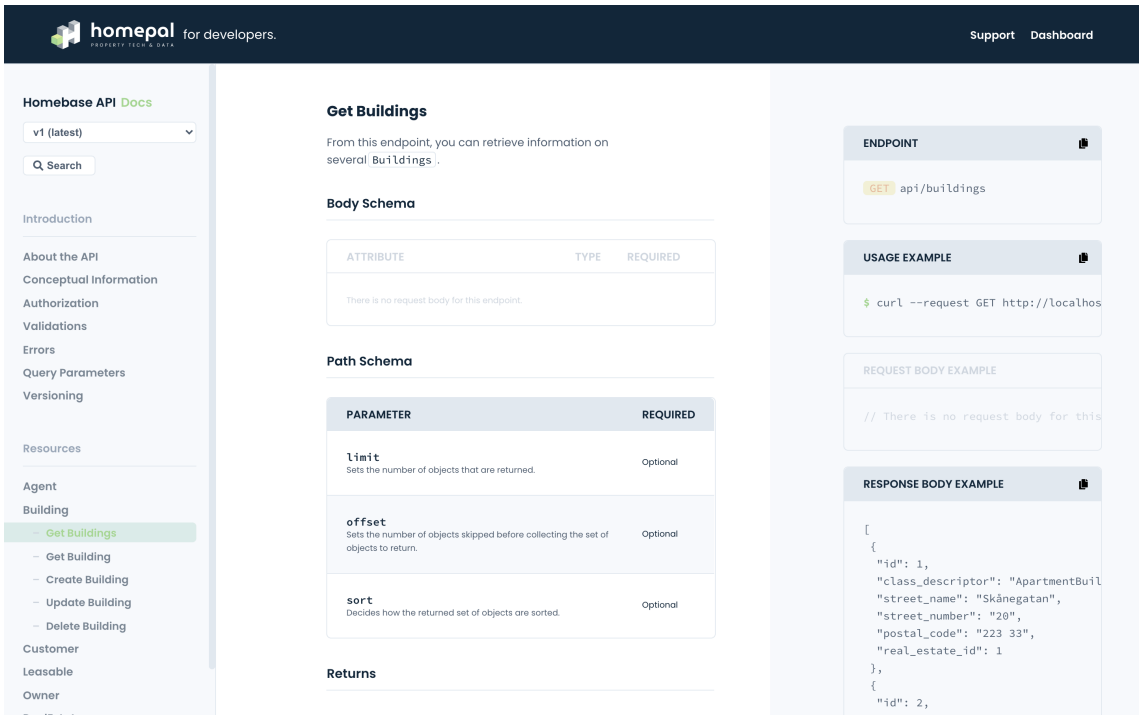


Figure 4.23: Final Prototype: Get buildings.

Chapter 5
Discussion

This chapter will discuss each phase of the design process, along with the activities performed during each phase. The goal is to find strong points, as well as limitations with this thesis, and its results. Furthermore, this chapter includes evaluations of the obtained results, along with suggestions on future work. Some remarks evaluating the viability of the activities and their results, have already been made in the corresponding section in ***Chapter 4****. However, in this section, the goal is to further expand on these remarks, as well as bring up positive and negative aspects of the activities.*

5.1 Understand Phase

The overall goal of this phase was to gain a solid foundation in theory, and understanding of the users, that could act as building blocks in subsequent phases. In order to achieve this, a literature study, revolving around both REST API technology and documentation, and interaction design theory was conducted. Furthermore, an heuristic evaluation of the current Homebase API documentation, along with competitors and other examples from the industry, was performed. Finally, interviews were held with developers currently using the Homebase API and its documentation.

5.1.1 Literature Study

Large amounts of time was spent on making the literature study as extensive as possible. This was time well spent, since it yielded a detailed picture of the landscape in areas that intersect with the thesis. Most of the efforts were placed on understanding areas related to API:s, and their documentation in particular, and how documentational content relates to UX. Some time was spent on researching interaction design theory, in the more general sense, but it was not prioritized, since documentation has some unique properties that differentiates it from more conventional applications. Furthermore, more conventional interaction design

theory is considered in the final prototype, but it is not explicitly mentioned, due to space constraints in the thesis report.

A frequently occurring issue was finding literature that was actively concerned with REST API reference documentation, not just API documentation in general. Local API:s are fundamentally different from REST API:s, and this affects the way they are documented. Furthermore, some authors did not clearly specify what type of documentation was considered in their research, which made takeaways difficult to apply in the context of this thesis.

5.1.2 Heuristic Evaluation

As mentioned in **Section 4.2.2**, the process of the heuristic analysis was valuable, since it provided both good and bad examples of documentation, along with ideas on potential solutions. Since each example was evaluated using the same heuristics, it was very systematic, which made it easy to compare examples of documentation to each other, in order to define what properties affected the quality of the documentation. If you wish to place more emphasis on the results, rather than the process, it might be beneficial to conduct a more thorough study, using several sets of heuristics from multiple sources, in order to further contrast the chosen examples of documentation. Also, using a larger number of evaluators may yield a more complete set of results.

5.1.3 User Interviews

The interview sessions with users of the current Homebase API documentation were very rewarding. The fact that these took place early in the process was an advantage in certain aspects, and a disadvantage in other aspects. It provided early insight into what real users prioritize, and how the documentation has been used historically, as well as the general experience level of the developers using the documentation. The interviews provided qualitative data, but in general, the data was speculative, and a lot of information regarded perceived usage. This is generally advised against in literature (see **Section 3.9.1**), but it was accounted for when compiling the results, since their opinions on perceived usage were only used as inspiration for what aspects to consider in subsequent stages, and were not seen as hard facts on what to include or how to solve certain issues. The reason for not eliminating the speculative aspect of the questions asked was in order to find new ideas and approaches that could provide value to the users. Including these new solutions and ideas would also not be an issue, since it had already been decided that an entirely new documentation would be developed during the process of this thesis.

However, due to the interviews taking place this early on in the design process made certain parts of the interviews irrelevant, since the scope of the thesis was frequently altered during the process, especially in the earlier stages. More relevant data could potentially have been generated if the interviews were held at a later stage. Furthermore, postponing the interviews would have given the authors more time to get acquainted with API:s and their documentation, which could have resulted in more detailed questions, and hopefully, more detailed answers.

5.2 Explore Phase

During this phase, the goal was to explore ideas and solutions that fit the needs of the end user. In order to categorize what was learned during the previous phase, potential features were first selected and prioritized. The features that were given high priority were brought to the next step, where a MidFi prototype was constructed. This prototype was evaluated in a discussion held with the Homepal developer team. In order to further evaluate potential ideas and solutions, a HiFi prototype was constructed, where focus was placed on providing an experience as similar as possible to a potential end product.

5.2.1 Feature Prioritization

The prioritization of features was based on every single interesting feature or aspect found in the understand phase, and was thus a great way to stop and think about the work so far. When listing every single feature, it was obvious that some had been lost in the crowd, and now came out in the light again due to this activity. To stop and quantize gathered data and insights (when it is not explicitly quantitative) is something that should be done frequently throughout a process in order to keep the discoveries alive.

However, when it comes to the actual prioritization of features, the fact that it was largely based upon the perceived usage, and needs, of five single API users can be discussed. The opinions of these played a major role in how the features were labeled (in combination with what the theory said). It is, on the other hand, hard to do this in any other way – except increase the number of users interviewed.

5.2.2 MidFi Prototyping

As described in the design process, the MidFi level was chosen as a starting point for the prototype due to the fact that LoFi was considered as a too basic and superfluous level of prototype for the purpose and the investigated features. In retrospect, this was considered the right choice to make since it hastened the process and saved the effort put into prototyping and usability testing for later, more suitable, levels of prototypes.

Starting of with a MidFi Prototype was a great way to put together the basic structure of, and lay the foundation for, the documentation and to make sure that all the prioritized features fit well together. However, it was difficult to create something useful (that could be tested) as an API documentation is based on a huge amount of repetitive components and pages, which is not optimal to create in Figma.

5.2.3 Developer Team Talk

In order to evaluate and iterate the MidFi prototype, the developer team was used as a sounding board in order to include new eyes and a new perspective – as well as to provide inputs, as they are people that work with API:s and API documentation on an every day basis. This was something that worked very well despite the informal form of "show-and-tell" talks, where the developers came with inputs (on what was good and on what was missing).

These talks gave both the authors of this thesis, and the developers, a chance to reflect over what had been done since every single piece of the prototype had to be explained and motivated. A bonus was to actually include the developers in the usability process and to make them engaged and committed to the cause.

However, the awareness of the developer team not being an optimal evaluation group was something always in mind throughout this process. Since this is a group that know the documentation and the domain by heart, everything is obvious for them, and their inputs should be taken with a pinch of salt. On the other hand, this was the only group that always was available in person (due to the ongoing pandemic at the time of writing).

5.2.4 HiFi Prototyping

The HiFi prototype was programmed, together with an associated API, in order to include functionality that enabled useful user testing and meaningful evaluation. However, this had its downsides due to the fact that every single change in design required change in code. The changes in code were sometimes complex, and required a lot of work – thus giving less incentive to actually make a change. This was something that the authors were fully aware of, but it was also a trade-off that had to be done in order to test the investigated functionality.

The fact that the size, and complexity, of the API and its documentation had to be reduced (a lot) – due to the limited amount of time and the experience level of the user test group – was something that had impact on certain features and functionality. Features that was considered as interesting and important from the understand phase (such as e.g. the conceptual overview) did not really have a purpose when using a fundamental and basic API. Likewise were sections such as 'Validations' and 'Versioning' included, but did not really hold any useful information due to the lack of this functionality within the API. However, despite the fundamental API, all of the prioritized features were included and given a place within the documentation, in order to mediate their actual importance.

5.3 Materialize Phase

In the final phase of the design process, the primary goal was to evaluate the HiFi prototype from the previous phase, through usability testing. Several factors of the prototype were evaluated, and all feedback and observations made during the study were collected, and used to further improve the prototype.

5.3.1 Usability Testing

Usability testing is a great tool for evaluating the general usability of a prototype, and the methodology provided a lot of insight for this thesis. However, there are always aspects of how the study was conducted that should be considered before drawing conclusions on the result. First and foremost, the group of participants were a very homogeneous group, where almost all participants were students from a technical background. This was intentional, since it was deemed that they had sufficient knowledge to understand the purpose of the prototype, while being able to complete the tasks. Despite having similar backgrounds, the

experience levels between the participants varied enough to draw conclusions on how experience affects usage. The reason for not using developers with several years of experience, was the concern that they would be able to solve the tasks without using the documentation extensively, just based on knowledge alone, which obviously would make it difficult to draw conclusions on how the prototype was used.

Another reason for selecting the participants in the aforementioned manner was convenience, especially when related to concerns with the pandemic. The participants were exclusively fellow students and colleagues of the authors, which may have affected the results negatively, since acquaintances probably are less likely to give negative feedback. In order to remedy this issue, participants were instructed to be as honest as possible, and that negative feedback was the most beneficial for the end product, but it is not clear how much this factor affected the results of the study. Conducting a study with more participants, from a less homogeneous group would probably have given different results, but due to the pandemic, it was not possible to follow through with plans of this nature.

The usability tests revolved around the participants solving a fixed number of tasks using the documentation prototype. This made it possible to observe how the participants used the documentation, and it also provided a tangible metric to use when evaluating how well the participants solved the tasks. However, it made the usability study feel like slightly like an exam, despite efforts in ensuring the participants that they were not evaluated based on their performance. This may have placed unwanted stress on the participants, which may have affected the results negatively. Furthermore, the participants sometimes may have experienced some time pressure, despite being explicitly told that time was unlimited. This probably has to do with being observed, while solving the tasks at hand. In a work or educational context, you tend to solve tasks alone, and are therefore more likely to feel stressed when being observed. This stress and time pressure could probably have been remedied by leaving the participants to solve the tasks alone, and instead, observing them through screen recording or teleconferencing software. However, this would remove some of the natural interaction between the test leader and participant, which could provide less qualitative data, due to a lack of real-time communication. But still, it is a solution, with clear trade-offs that could be considered when conducting similar studies.

Another concern with the usability study was evaluating the prototype in isolation. The documentation has a direct dependency to the mock API that was written for the usability study. This is clearly impossible to avoid, since the documentation always has to document something to fulfill its purpose. However, in order to test the functionality of the API, some external tool had to be used. As previously mentioned, Postman and/or the terminal was recommended in the briefing before each usability test. Using these tools may have colored the perception of the prototype, and may have subsequently affected the final results.

5.3.2 Result Evaluation

One of the main goals when planning the usability study was to obtain results of several types, from various angles of approach. This is why a task based approach was used, since it made it possible to measure success in task completion. The System Usability Scale was then used to collect the opinions on the overall usability of the prototype. These two approaches provided quantitative data. In order to provide qualitative data, participants were asked to fill out a survey of more detailed questions. Observations on how the documentation was used by the

participants were also collected during the tests, in order to not exclusively include how the participants themselves perceived the prototype, but rather, how it was actually used. Conclusively, the study provided both qualitative and quantitative data, from the perspective of the participants, but also from the perspective of the test leader. Furthermore, the results could be used to evaluate the overall usability of the prototype, as well as how well the documentation supported the participants in solving the tasks given, in relation to the API and its functionality.

The task success of the results was higher than expected, especially among the participants with less experience. Participants with more experience still performed better, both in relation to time spent and task success, but some participants with very little experience performed significantly better than expected. Hopefully, this is a testimony to the quality of the documentation, but external factors may have affected the results as well. A lot of effort was put on correctly assessing the difficulty of the given tasks, since it could affect how the documentation was used. It is possible that the tasks given were too simple, which may have contributed to the high general task success. However, if the tasks would be been made significantly more difficult, it could have been hard to draw conclusions on experience levels, since the tasks may have been too hard for less experienced participants to complete. Furthermore, due to the nature of API:s, and the small size and limited functionality of the mock API created, it would have been almost impossible to create significantly more difficult tasks.

The average score for the System Usability Scale was 95, which is a high score, that lies much above the acceptability limit. Once again, this is hopefully a sign that the documentation has good usability, but there are external factors that may have contributed to the good average score. As previously mentioned, the participants were all acquaintances of the authors, which may have resulted in them giving a higher score. Furthermore, a lot of effort was spent on making the prototype aesthetically pleasing, which is a common pitfall when making HiFi prototypes. Participants may be more likely to comment positively on a more polished product, and may also be more likely to glance over negative aspects of the prototype. Since a lot of the participants commented that the prototype looked polished and nice, it is possible that this may have ballooned the SUS score.

The observations made by the test leader are inherently subjective, since the test leader may misinterpret the actions of the test participant. In order to ensure that the observations made were correct, the participants were asked about the validity of the observation. However, this may still have introduced some bias, since the participants were actively asked about the way that the used the product, similar to the query effect described in **Section 3.7**.

5.3.3 Final Prototype

After having carried out usability tests, all difficulties discovered and the feedback received were discussed and prioritized before, if necessary, being implemented in the final prototype.

At this stage in the design process, the authors had learned enough about theoretical aspects and developer needs, in order to prioritize from knowledge. However, there were three questionable omissions that can be discussed further. First of all, all suggested improvements that opposed the industry standard were removed, even though they might have been more intuitive. The motivation behind this was simply to avoid confusion for users, especially users with more experience, that are used to certain standards.

Second of all, refinements that involved basic development knowledge were not prioritized. The fact that this was done is something that, in a way, opposes concepts of UX and usability, but is, by the authors, motivated as a way to avoid bloat and superfluous text sections. Some things are not in the scope of the documentation for an API, and is better learned elsewhere.

Lastly, upgrades that were time consuming, due to complexity of implementation, were disregarded. Unfortunately, this is one of the downsides with HiFi prototyping, especially when writing code, but it is a trade-off that must be made, in order to carry out useful usability tests. If more time had been available, these changes would naturally have been investigated further.

Something that has been discussed throughout the process is concept of bloat, which in this context means inclusion of irrelevant or redundant information. Initially, it was considered in relation to how experience levels affects the needs of the users. Fairly early, it was made evident that more experienced developers are less focused on textual content, and focus more on examples and endpoints. Developers with less experience seemed to need the support of the textual content to make use of the other documentational content, at least when first getting acquainted with the API. Therefore, including too much textual content may disrupt the usage patterns of experienced developers, and vice versa. Since support for different developers was one of the goals of this thesis, this resulted in a trade-off problem, where the needs of the two users clashed. Bloating the documentation was not only an issue in regards to textual content, but also in relation to maintaining the deliberately consistent layout, which sometimes resulted in empty components, which, in a sense, also bloats the documentation. This is a very interesting subject, since it can be applied to all forms of user interfaces, not only in relation to API documentation.

Another aspect that was considered, was the single page with infinite scroll. At first glance, it appeared to confuse users, but after getting acquainted to the system, it was appreciated. The main reason for choosing this layout, was the possibility of using the native search functionality of the browser, as well as avoiding using pages of several depths, which could result in users getting lost. The confusions of the users were related to the fact that they experienced difficulties understanding when a specific section ended. The final design decision was to keep this single page, and instead try to highlight and amplify the division between sections. Furthermore, as previously mentioned, much effort was put into making the navigation as transparent as possible, to aid users in finding what they were looking for.

In addition, a dedicated, integrated search functionality was discussed back-and-forth throughout the process, due to the fact that it was encouraged by certain users and theory. On the contrary, other users and theory both expressed the feature as superfluous due to the (earlier mentioned) fact of the documentation being single paged, and can therefore be searched with the search functionality of the browser. Furthermore, users also brought up fact that searching in documentation sometimes is difficult, since naming is unique to each API. However, due to limited use of the browser search functionality in the user tests, a search field was added at this stage. When pressed, the search field gives the user an encouraging message of using the browser's search functionality. Hopefully, this could be an easy way of reminding users of the browser native search functionality, and the fact of the documentation being searchable. However, this feature has not been user tested, which it should when considering taking the prototype and the design further.

Usage examples was a feature that had been in focus ever since the beginning of the design

process. It was interesting, and fascinating, to see the wide usage of this feature in the user tests, where every single participant used the feature to some extent. Thus, the feature of usage examples was kept in the final prototype, unfortunately with a flaw, however. The usage examples do not have any line breaks due to technical complications and time limit, and is therefore presented on a single line with a horizontal scroll. This is clearly not ideal, due to the fact that the majority of the example code is hidden at first glance and that it requires precise scrolling in order to view a desired section of it. If taking the prototype and the design further, the improvement of this feature should be considered as a must.

Another aspect that had high priority throughout the development of the prototype, was the feature of giving all types of developers documentation suitable for their specific needs. Different developers have different approaches and experience, and thus have different needs. Two approaches that were discussed was either letting users *Customize* aspects of the documentation, or attempting to design for all types through *Universal* documentation. The latter was chosen, mainly due to a lower technical complexity level, and kept in the final prototype. In order to fulfill the needs of different developers and creating a universal documentation, the characteristics of finding information fast as well as being able to read detailed information at the same time were both prioritized. This was the main reason behind the two columned layout, with one column containing pure descriptive information and the second column containing fundamental, usage information. During the user tests, different types of developers were identified, which agreed with the theory as described in **Section 3.11**. The usage of the different columns were confirmed; some spent a lot of time in the descriptive column, while others where satisfied with the concrete and concise usage examples in the usage column. Some test subject read the entire documentation before starting their tasks, while some found an appropriate example in the rightmost column and began experimenting. More experienced developers were more likely to start experimenting, using the examples for a given endpoint, while developers with less experience were more thorough in their approach to solving the task at hand. It was rewarding to see the prototype being used in different ways, and that different developers with varying levels of experience could find parts of the documentation useful to them.

Worth noting, is that the final prototype has not been evaluated extensively in any way, with the reason being time constraints. However, since the changes made to the final prototype were generally very small, chances are that the result would have been very similar. However, the main purpose for this final prototype is to illustrate all the learnings that have been collected during the process of this thesis. The idea is that Homepal can use this prototype as a single point of entry for inspiration, and through this, choose what properties and features to integrate into their current documentation, in order to improve it further.

5.4 Future Work

This section presents ideas for future work, discovered in the design process of this work, in regards to the API documentation.

FW1: The impact of deviation from industry standards: In several cases, both the authors and the users considered themselves having a better and more intuitive way of presenting information or usage examples, than the general industry do. An interesting future work would be to investigate the impact of deviating from the industry standard when

considered having a more intuitive way of doing something. Does the deviation cause more confusion than the proposed alternative improve the intuitiveness?

FW2: User customization of documentation: This thesis had a focus on creating one documentation that was suitable for all types of developers with all levels of experience. However, an alternative way is to create a documentation that can be customized, and toggled, between different needs – thus avoiding trade-offs. An interesting future work would be to investigate this alternative method and/or compare them.

FW3: Optimization of the symbiosis between external developer tools and API documentation: The fact that developers tend to stick with their developer tools, and not use the integrated tools in a documentation (such as sandboxes, etc.) creates a gap between the two. An interesting future work would be to investigate further how one can decrease the gap and stronger the cooperation. Is an optimal merge possible, or is there a reason behind the separation of the two?

FW4: Inclusion of deviating features within a consistent structured documentation: An aspect that was discovered in this thesis (but not investigated further) was that fact that there are features of certain resources (e.g. enumerated values) that need the possibility of deviating from the (otherwise) consistent and structured format. An interesting future work would be to investigate how this can be done in the most optimal way – without confusing the user.

FW5: Integration of features that are relevant but considered out of scope within a documentation: A feature such as authorization considered central and fundamental when it comes to API. However, they are almost never present within API documentation. An interesting future work would be to investigate if there is a reason behind this or if there is possibility of strengthen the usability by integrating this functionality within the documentation.

FW6: Documenting validations: An aspect discovered, and expressed by the interviewed everyday API users, is the need for more information regarding the validations within an API. The users are all excusing its absence by claiming that it is a hard task to accomplish, yet they express a great need for it. An interesting future work would be to investigate if, and how, this can be included within a documentation.

Chapter 6
Conclusion

As mentioned in **Section 1.4**, the purpose of this thesis was to investigate what factors positively impact REST API reference documentation, and how the needs of different developers can be supported in the Homebase API documentation. This purpose was defined early on in the process, and aims to pertain to the the scope defined in **Section 1.3**. Four goals were formulated in order to summarize the purpose, and were used as guidelines for what was focused on during the thesis. These goals are listed below, together with a summarized overview of the findings in relation to the goal.

Goal 1: **Understand what features and properties should be included in an REST API reference documentation to support the needs of the developers.**
During work on this thesis, several key factors have crystallized themselves as important for the overall usability, with most of them revolving around finding and understanding certain information quickly. A tool that enables this is differentiation of content, where content of different types that have been separated in separate columns with different general formatting in order to aid users in finding the information they need. Providing transparent navigation has also been beneficial for users in finding the information that they want at a given time. Furthermore, allowing for users to use the native search functionality in their browser by placing all content on a single page proved to be useful in certain cases. By maintaining a very consistent layout, the same information type can be found in the same manner every time, which also facilitates certain usage patterns.

Goal 2: **Understand how different types of developers use documentation differently, and how the documentation accommodates for the needs of different developers.**
No user is the other alike, and this applies to developers as well. It was found that usage patterns varied between developers, both based on experience level, and other potential factors such as background and personality. Why developers behave differently has not been researched further, but during usability testing it was made evident that developers had their

own approaches to solving the problem at hand. By constantly weighing options against each other, and reflecting on how design decisions would affect different developers, the prototype seems to be able to handle different developers well. The key to achieving this seems to be providing content and functionality that fit different usage patterns and scenarios.

***Goal 3*: Apply what was learned in relation to previous goals in order to create a prototype that performs well in evaluation.**
A prototype was created, that illustrates the learnings from this thesis. The prototype was highly and widely appreciated in the conducted usability study, with an average System Usability Scale score of 95, which corresponds to "Best Imaginable", according to the scale in **Figure 3.2**. The study was conducted with users that have different levels of experience, and generally performed well for all users, both in terms of usability, but also in terms of their understanding of the REST API.

References

[1] B. Ahmady and T.M. Nguyen. Developer experience as a product. https://www.youtube.com/watch?v=FscA-gsIQgs, 2020. [Accessed: 2020-02-15].

[2] A. Bangor, P. Kortumand, and J. Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123, 2009.

[3] L. Bellamy, M. Carey, and J. Schlotfeldt. *DITA best practices: A roadmap for writing, editing, and architecting in DITA*, page 80. IBM Press, 2011.

[4] T. Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259, December 2017.

[5] J. Brooke. Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, 189, 11 1995.

[6] S. Clarke. What is an End User Software Engineer? In *End-User Software Engineering*, number 07081 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[7] R. Fielding et al. RFC 2616, Hypertext Transfer Protocol – HTTP/1.1, 1999.

[8] J. Etcovitch. Playing with interactive documentation. https://www.youtube.com/watch?v=fJb2bpmrguw, 2020. [Accessed: 2020-02-15].

[9] P. C. Evans and R. C. Basole. Revealing the API Ecosystem and Enterprise Strategy via Visual Analytics. *Communications of the ACM*, 59(02):26–28, 2016.

[10] R. Fielding. *Architectural styles and the design of network-based software architectures*. Publication, University of California, Irvine, 2000.

[11] S. Gibbons. Design thinking 101, 2016. https://www.nngroup.com/articles/design-thinking/.

[12] Interaction Design Foundation. Making your ux life easier with the moscow. https://www.interaction-design.org/literature/article/

making-your-ux-life-easier-with-the-moscow/, 2015. [Accessed: 2020-03-11].

[13] T. C. Lethbridge, J. Singer, and A. Forward. How software engineers use documentation: The state of the practice. *IEEE software*, 20(6):35–39, 2003.

[14] J. Lewis. Usability testing. *Handbook of Human Factors and Ergonomics*, pages 1275 – 1316, 2006.

[15] H. Loranger. Checklist for planning usability studies, 2016. https://www.nngroup.com/articles/usability-test-checklist/. [Accessed: 2020-02-15].

[16] W. Maalej and M. P. Robillard. Patterns of Knowledge in API Reference Documentation. *IEEE Transactions on Software Engineering*, 39(9):1264–1282, 2013.

[17] W. Maalej, R. Tiarks, T. Roehm, and R. Koschke. On the comprehension of program comprehension. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(4):1–37, 2014.

[18] S. Marvin. What is API documentation. *Intercom*, 61(8):6–8, 2014.

[19] John McCarthy and Peter Wright. Technology as experience. *interactions*, 11(5):42–43, 2004.

[20] M. Meng, S. Steinhardt, and A. Schubert. Application Programming Interface Documentation: What do Software Developers want? *Journal of Technical Writing and Communication*, 48(3):295–330, 2018.

[21] M. Meng, S. Steinhardt, and A. Schubert. How Developers Use API Documentation: An Observation Study. *Commun. Des. Q. Rev*, 7(2):40–49, August 2019.

[22] M. Meng, S. Steinhardt, and A. Schubert. Optimizing API Documentation: Some Guidelines and Effects. In *Proceedings of the 38th ACM International Conference on Design of Communication*, SIGDOC '20, New York, NY, USA, 2020. Association for Computing Machinery.

[23] L. Mitchell. Beyond the api reference: Developer experience for apis. https://www.youtube.com/watch?v=fRLIQgjkn-g, 2020. [Accessed: 2020-02-15].

[24] M. Naggaga. Beyond copy + paste: Creating interactive documentation. https://www.youtube.com/watch?v=XWetRp1f5xg, 2020. [Accessed: 2020-02-15].

[25] S.M. Nasehi, J. Sillito, F. Maurer, and C. Burns. What makes a good code example?: A study of programming q&a in stackoverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 25–34. IEEE, 2012.

[26] J. Nielsen. How to conduct a heuristic evaluation. https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/, 1994. [Accessed: 2020-05-12].

[27] J. Nielsen. Why you only need to test with 5 users, 2000. https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/. [Accessed: 2020-02-15].

[28] J. Nielsen. Interviewing users. https://www.nngroup.com/articles/interviewing-users/, 2010. [Accessed: 2020-02-24].

[29] D. Norman and J. Nielsen. The definition of user experience, 2014. https://www.nngroup.com/articles/definition-user-experience/. [Accessed: 2020-02-15].

[30] D. A. Norman. *The design of everyday things*. Basic Books, [New York], 2013.

[31] R. Paul. Documentation as an application: enabling interactive content tailored to the user. https://www.youtube.com/watch?v=aLVvSyenA6s, 2020. [Accessed: 2020-02-22].

[32] T. Pluskiewicz. REST Misconceptions Part 0 - Do You Really Understand the REST?, 2016. https://t-code.pl/blog/2016/02/rest-misconceptions-0/.

[33] Postman. 2020 State of the API Report. https://www.postman.com/state-of-api-report-2020.pdf.

[34] J. Preece, Y. Rogers, and H. Sharp. *Interaktionsdesign: Bortom Människa-Dator-Interaktion*. Studentlitteratur, 1 edition, 2016.

[35] RapidAPI. RapidAPI Developer Survey and Insights 2019 - 2020. https://rapidapi.com/blog/wp-content/uploads/2020/02/rapidapi-dev-survey.pdf.

[36] M. P. Robillard. What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software*, 26(6):27–34, 2009.

[37] M. P. Robillard and R. DeLine. A field study of API learning obstacles. *Empirical Software Engineering*, 16:703–732, 2011.

[38] S. M. Sohan, F. Maurer, C. Anslow, and M. P. Robillard. A study of the effectiveness of usage examples in REST API documentation. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 53–61. IEEE, 2017.

[39] G. Uddin and M. P. Robillard. How API Documentation Fails. *IEEE Software*, 32(4):68–75, 2015.

Appendices

Appendix A
User Interviews

A.1 Interview Questions

- Tell us a little bit about your company.
  - What is your business?
  - What do you use Homebase API for?
- Tell us a little bit about you and your work.
  - How experienced are you in working with REST API:s?
  - How long have you been working with the Homebase API?
  - How often do you setup/integrate with Homebase API?
- Describe how you use the documentation.
  - How frequently do you use it?
  - What parts do you look at?
  - What do you find most/least useful?
- Describe your first integration.
  - Do you remember it?
  - What was easy/hard?
- Describe your thoughts on the conceptual information.
  - Have you had trouble in understanding concepts related to the domain of the real estate industry? E.g. understand what a *Leasable* is?

  - Would you appreciate more/less conceptual information?
  - Would you like an architectural overview of the API included in the documentation?

- Describe your thoughts on code examples.
  - Do you think that the API documentation could benefit from code examples?
- Describe your thoughts on the structure and distribution of content.
  - Do you think that the documentation is structured well in regards to the content?
- Describe your thoughts on missing features.
  - If you could add one feature to the documentation, what would you add?
  - Would you use a search functionality?
  - Would you use a favorite functionality?
  - Would you use a function where you could test the functionality of the API within the documentation? (Or do you like the external sandbox solution more?)
- Final thoughts.
  - Do you have any additional inputs on Homepal's API documentation?
  - Do you have any certain thoughts on API documentation overall?

A.2 Form of Consent

Consent to take part in research

I voluntarily agree to participate in this research study.

I understand that even if I agree to participate now, I can withdraw at any time or refuse to answer any question without any consequences of any kind.

I agree to my interview being audio-recorded.

I understand that the audio-recording will be destroyed after the transcription of it has been finished.

I understand that in any report on the results of this research my identity will remain anonymous. This will be done by changing my name and disguising any details of my interview which may reveal my identity or the identity of people I speak about.

I understand that disguised extracts from my interview may be quoted in a published Master's thesis.

I understand that under freedom of information legalisation I am entitled to access the information I have provided at any time while it is in storage as specified above.

I hereby give my consent, through an oral agreement, to the above stated information.

Contact details of thesis students:

Axel Holmqvist
axel@homepal.se
tfr14ah2@student.lu.se

David Jungermann
david@homepal.se
dic15dju@student.lu.se

Appendix B
Usability Tests

B.1 User Tasks

Setting

You are working your first day as an administrator for a real estate company. In order to make yourself familiar with the system (which is called Hombase API), you have been given a group of basic tasks from your colleague. The goal is to solve these tasks, and return short answers for every task to your colleague.

Task 1

First of all, it is important to make yourself familiar with the documentation for the Hombase API. Read through the introductory chapters and try to get a feeling for how the API works on a basic level.

After this, your colleague would like to know:

- What data format is used in the API?
- What is the base-URL of the API?
- What query parameters exist?
- What error codes are used?

Answer:

Task 2

Your colleague would like you to authorize yourself against the Homebase API in order to start using it. Write the five first characters of your key.
Answer:

Task 3

Now, your colleague wants you to retrieve several **Agents** from the API. Answer with **name** and **id** for the first two **Agents**.
Answer:

Task 4

Your real estate company has a new owner. Your colleague wants you to **create** an **Owner**. Answer with **id** for the **Owner** you just created.
Answer:

Task 5

Now, your colleague wants you to create a **RealEstate** with the name "Center South". "Center South" is situated in the Region of "Scania" and is, by pure chance, owned by the **Owner** you just created. Answer with a complete object structure for this **RealEstate**.
Answer:

Task 6

Your colleague wants to control that the RealEstate "Center South" was created correctly. Therefore, he/she wants you to get all ReaEstates, and that you sort those by name in order to find it easily. Answer with the name of the RealEstates that comes before "Center South" in your result.
Answer:

Task 7

The name "Center South" feels a bit old fashion. Therefore, the board have decided to change the name of the mall to "Eclipse Center South". Your colleague wants you to update the name in the API. Answer with the updated, full object structure.
Answer:

Task 8

The real estate company that you are working for is accused for being too expensive. Therefore, your colleague wants you to remove the Leasable with the highest rent. Answer with the id for the now removed Leasable.

Answer:

B.2 Form of Consent

Consent to take part in research

I voluntarily agree to participate in this research study.

I understand that even if I agree to participate now, I can withdraw at any time or refuse to answer any question without any consequences of any kind.

I agree to my session being screen captured.

I understand that all screen captures will be destroyed after the analysis of them has been finished.

I understand that in any report on the results of this research my identity will remain anonymous. This will be done by changing my name and disguising any details of my interview which may reveal my identity or the identity of people I speak about.

I understand that disguised extracts from the interview part of my session may be quoted in a published Master's thesis.

I understand that under freedom of information legalisation I am entitled to access the information I have provided at any time while it is in storage as specified above.

I hereby give my consent to the above stated information:

Full name

Date and place

Signature

Contact details of thesis students:

Axel Holmqvist
axel@homepal.se
tfr14ah2@student.lu.se

David Jungermann
david@homepal.se
dic15dju@student.lu.se

B.3 User Feedback Questions

Questions about the system

- What did you appreciate the most in Homebase API Docs?
- What disruptions did you experience in Homebase API Docs?
- Did you think that the conceptual overview gave you enough information in order to solve the tasks?
- How did you feel about finding the information you were looking for in Homebase API Docs? Did you feel that the structure was logical? Was the right information in the right place?
- Did you find that the information available helped you to solve the problems in a good way? Was there information that was superfluous? Was information missing?
- Did you find that the usage examples in Homebase API Docs helped you solve the tasks?
- Do you have any other comments?

Appendix C
Implementation

C.1 Selected Implementations

Table C.1: Prioritization of improvements, part 1.

Aspect	Priority	Note
Line-breaked cURL examples.	Agrees, but won't.	Time consuming due to linebreaks not working well in curl.
CRUD-order when it comes to the requests of an endpoint/resource (due to intuitiveness).	Disagrees, so won't.	Hard to understand the CRUD order if you haven't heard of the concept before.
A clearer indication and explanation of the documentation being a single page.	Disagrees, so won't.	Maybe smooth scroll can help, but it feels like this is a very small issue that solves itself after using the documentation for two seconds.
A search field/functionality.	Agrees, and will.	Maybe not through a search bar, since they can be difficult to implement well. But, since the docs are single page, we can encourage use of ctrl + f.
The users express a wish for removing unnecessary complicated terminology. E.g. "U+002D Hyphen minus".	Agrees, but won't.	Already been fixed.
More information about what the API could be used for.	Agrees, but won't.	Due to the narrow API, this would not be good. Not in scope of thesis.
Explain both body- and path schema, and the difference.	Disagrees, so won't.	This will contribute to bloat, and is something that a developer should know. Superfluous.
Collapse repetitive information/components.	Disagrees, so won't.	Can't search. Easy to miss collapsed stuff. Disagrees with theory.
More thorough (and varying) descriptions about prerequisites on resources.	Agrees, but won't.	Not in scope, we have written as much technical documentation that is needed to understand and use the prototype. Building on top of this is not in the scope for this thesis.

Table C.2: Prioritization of improvements, part 2

Aspect	Priority	Note
Make the Usage column more noticeable.	Agrees, and will.	We will add a little text in the column that explains its purpose.
A more clear distinctions between sections.	Agrees, and will.	Thicker lines between sections
A more clear distinction between titles and subtitles.	Agrees, and will.	Fixed!
A clearer indication on whether placeholders such as { } or " " should be included or not in a path or body.	Disagrees, so won't.	Not in scope. Can add a text that explains it, but it bloats the docs. Better that you make the error once, and then remembers it.
Remove or explain disabled tables.	Disagrees, so won't.	They are placed there for consistent layout, so you always find the same information and the same place. Could be confusing if tables switched places based on endpoints
Change the order of the elements Body schema, Returns, Path schema to Body schema, Path schema, Returns (due to intuitiveness).	Agrees and will.	It is a more intuitive order.
Explain query parameters a bit more.	Agrees and will.	More fundamental information.
Remove "$ curl" when an example is not actually a usage example.	Agrees and will.	Path examples?
Expand link hit area in the side bar.	Agrees and will.	Link the whole element.
Fix misclicks on headers in the side bar.	Disagrees, so won't.	The headers in the side bar does not in any way indicate being a link.
A possibility of using the browser's back arrow for navigation to the last visited hash-location.	Agrees, but won't.	Technical implementation. Out of scope.
A more visible Generate Key button.	Agrees and will. / Agrees, but won't.	Will be fixed if there is time, but not very important.
Fix the issue of users missing "-s" on endpoint URLs.	Disagrees, so won't.	This is a REST staple, and is something you have to have. Its a thing you probably learn in two minutes, and then never think about again. Hard to clarify without being ambiguous
Empty strings instead of placeholders in the examples.	Disagrees, so won't.	The example attributes provide knowledge on the type data associated to the given attribute, an empty string does not do that.
More clear explanations of what is auto-generated or not.	Agrees and will.	Will clarify a bit in the introduction.
An explanatory text of the conceptual overview (*, 1, <, etc.)	Disagrees, so won't.	It will only be useful for people who already know UML syntax. Out of scope.
Collapse navigation items in the side bar.	Agrees and will.	This has to be done. It was going to be implemented before the user tests, but we did not make it in time.