Erarbeitung eines webbasierten Dokumentationskonzepts zur Low-Code-Plattform "yeet"

Studiengang Medieninformatik

Bachelorarbeit

vorgelegt von

Chiara Knipprath
geb. in Gießen

durchgeführt bei der vectorsoft AG, Heusenstamm

Referent der Arbeit:	Dipl.-Ing. (FH) Benjamin Einert
Korreferent der Arbeit:	Prof. Dr. rer. nat. Dominik Schultes
Betreuer bei vectorsoft:	Miqueas Cramer

Friedberg, 2021

Abstract

The present work has the goal to create a web-based documentation concept for the low-code platform "yeet" of the company vectorsoft AG.
Basic requirements for API documentations have been collected from related work as well as the state of the art and a first overview of modern documentation systems was created. Based on this knowledge, an individual user test / interview with subsequent survey was designed. Thereby it was possible to highlight many requirements.
The "must-have" and "nice-to-have" requirements, which emerged from the analyses, were fundamental components of the concept. For these are to be designated now particularly as requirements of low-code documentations.
Because of the compiled requirements, the technology selection was the next step. Several documentation technologies were compared with a matrix specially constructed for this purpose. In the result two technologies were outstanding.
Finally, a documentation concept for the new low-code platform was created. It consists of all informations compiled before and contains requirements as well as technology recommendations particularly for "yeet".

Danksagung

Zuerst möchte ich der Firma vectorsoft AG danken. Obwohl ich erst vor drei Monaten in die Firma gekommen bin, wurde ich vom ersten Tag an unterstützt und meine Anliegen wurden ernst genommen. Mit mir zusammen wurde ein Thema für diese Arbeit gefunden und ich hatte immer eine Anlaufstelle für Fragen. Wenn ich ein Problem hatte, konnte ich mich immer an meine Kollegen wenden und erhielt eine freundliche, kompetente Antwort.
Ganz Besonders möchte ich noch den Teilnehmern des User Tests danken. Ohne sie hätte ich nie so viele informative Ergebnisse bekommen.
Auch bei meinen Korrekturleserinnen bedanke ich mich ganz herzlich. Eine zweite unbeteiligte Person die Arbeit lesen zu lassen war sehr wichtig für die Rechtschreibung, Grammatik und generelle Lesbarkeit.
Zuletzt danke ich all meinen Freunden und speziell meinem Freund, der mich während der letzten Wochen ausgehalten, motiviert und mental unterstützt hat.

Selbstständigkeitserklärung

Ich erkläre, dass ich die eingereichte Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Solms, 14. Oktober 2021

Chiara Knipprath

Inhaltsverzeichnis

Kapitel 1

Einleitung

Diese wissenschaftliche Arbeit wurde in gemeinsamer Zusammenarbeit mit der Firma vectorsoft AG erarbeitet und befasst sich mit der Konzeptionierung einer Anwender-Dokumentation für die Low-Code-Plattform "yeet", welche die Firma zurzeit entwickelt. Dazu werden Anforderungen an eine Anwender-Dokumentation für Low-Code-Systeme analysiert und aktuelle Dokumentationstechnologien verglichen.

1.1 Vorbemerkung

Aus Gründen der besseren Lesbarkeit wird auf die gleichzeitige Verwendung männlicher und weiblicher Sprachformen verzichtet. Sämtliche Personenbezeichnungen gelten gleichwohl für beiderlei Geschlecht. Allerdings soll dies niemanden in seiner Person und seinem Geschlecht diskriminieren und jeder interessierte Leser soll sich dementsprechend angesprochen fühlen.

1.2 Motivation

"An api documentation is like an entrance into your Api and should provide a warm welcome to the API's customers." [IJRJ18, Seite 230]
Dieses Zitat ist ein gutes Beispiel dafür, wie wichtig heutzutage eine optimierte Dokumentation für das Gesamtbild einer Anwendung ist.

Viele Menschen unterschätzen die Bedeutsamkeit einer Dokumentation immens. Angefangen bei alltäglichen Dingen, wie ein neues technisches Gerät zu bedienen, ein Regal aufzubauen, oder ein Rezept nachzukochen. Bei all diesen Dingen ist es wichtig, eine gute, verständliche Anleitung zu haben. Andernfalls passiert es schnell, dass aus Verzweiflung aufgegeben wird, oder durch Unverständnis Fehler entstehen.
Genauso ist dies bei Dokumentationen für Softwares. Bei einer neuen Software muss ebenso zu Beginn gelernt werden, wie diese richtig benutzt wird. Fehler müssen schnell verstanden und behoben werden können und Grundinformationen müssen leicht zu finden sein. Durch eine gut verständliche Dokumentation stellt dies kein Problem dar. Fehlt allerdings diese konkrete Dokumentation, führt dies gleichermaßen zu Verzweiflung und Unverständnis. Außerdem ist in vielen Fällen eine hilfreiche Dokumentation für den Anwender ein wichtiges

Kriterium bei der Auswahl einer Software. Eine schlechte oder sogar lediglich schlecht zu findende Dokumentation kann ausschlaggebend für eine Entscheidung gegen die Anwendung sein.
Die Herangehensweise vieler Programmierer, überhaupt nicht mehr in der Dokumentation der Software nach einer Lösung für ihr Problem zu suchen, ist ebenfalls auffällig. Sie benutzen vorzugsweise als erste Anlaufstelle Google und finden dabei andere Hilfe-Websites oder Foren, wie "Stack Overflow" [HZH19, vgl. Seite 232]. Viele haben demzufolge schon eine schlechte Erwartungshaltung an eine Dokumentation. Diese Erwartungshaltung zu durchbrechen und aufzuzeigen, dass Dokumentationen dennoch hilfreich sein können, ist das Ziel des Konzeptes, welches mit dieser Arbeit erbracht werden soll. Denn eine optimierte Software-Dokumentation kann die "Usability" (Benutzerfreundlichkeit) des gesamten Produktes verbessern [Bie06, vgl. Seite 62].
Die Firma, mit dessen Hilfe diese Arbeit entsteht, ist in der Position bald eine neue Software auf den Markt zu bringen, welche eine klassische Low-Code-Plattform ist. Für dieses neue Produkt "yeet" wird eine Dokumentationslösung benötigt. Das Besondere hierbei ist, dass "yeet" eine Low-Code-Software ist und somit erst analysiert werden muss, ob Anforderungen an eine "normale" API-Dokumentation genügen. API-Dokumentationen sind zurzeit eine der häufigsten Formen von Software-Dokumentationen. Durch die Low-Code Elemente werden auch Abschnitte gebraucht, welche die gesamte grafische Benutzeroberfläche (engl. Graphical User Interface, kurz GUI) der Software erklären können.
Aus diesen Gründen muss wissenschaftlich herausgefunden werden, welche generellen Anforderungen an eine Low-Code-Dokumentation bestehen und mit welchen derzeitigen Technologien diese Anforderungen am besten umgesetzt werden können. Dabei wird insbesondere auf die speziellen Anforderungen der Low-Code-Plattform "yeet" geachtet.

1.3 Problemstellung

Was macht eine gute Anwender-Dokumentation aus? Einige andere wissenschaftliche Arbeiten behandeln diese Frage bereits. Jedoch wird in diesen Arbeiten meist nur Bezug auf API-Dokumentationen genommen. Die vorliegende Arbeit soll dieses Problem aufgreifen und Anforderungen an eine optimierte Low-Code-Dokumentation aufzeigen.
In dieser Arbeit soll, mithilfe von wissenschaftlichen Methoden, ein Dokumentationskonzept erarbeitet werden, welches auf die Bedürfnisse der Low-Code-Plattform "yeet" ausgelegt ist. Um dies zu erreichen, werden verschiedene Instrumente des wissenschaftlichen Arbeitens angewendet.
Zuerst sollen Informationen aus verwandten Arbeiten recherchiert und ausgewertet werden. Der Fokus soll hier auf den wichtigen Aspekten einer Dokumentation liegen, welche für eine gute Qualität verantwortlich sind. Aus diesen Erkenntnissen sowie des Wissens aus dem aktuellen Stand der Technologien, wird eine Umfrage erstellt, welche mithilfe verschiedener Personen durchgeführt wird. Diese Umfrage wird aus zwei Teilen bestehen, einem User-Test / qualitativem Interview und einem Umfragebogen, welcher am Ende noch einmal konkret die wichtigsten Fragen des Tests abdeckt. Durch diese Umfrage sollen, zusätzlich zu den bereits bekannten Eigenschaften einer guten Dokumentation, Merkmale gefunden werden,

welche speziell für die Firma, beziehungsweise die Low-Code-Software "yeet", wichtig sind. Nachdem alle essenziellen Faktoren einer guten Dokumentation gefunden sind, werden funktionale Anforderungen angefertigt. Diese sollen helfen, den Überblick zu wahren und sind ebenfalls dazu da, um die verschiedenen Anforderungen nach Relevanz zu priorisieren. Um die Anforderungen zu fertigen, werden die Ergebnisse der Umfrage in "must-have"- und "nice-to-have"-Anforderungen unterteilt und später mit den zuvor recherchierten Anforderungen verglichen. Ein weiteres Teilziel dieser Arbeit soll eine Vergleichsmatrix der verschiedenen Dokumentationstechnologien sein. Die Vergleichsfaktoren werden die zuvor erarbeiteten Anforderungen sowie existierende Technologien sein, welche zunächst ausgewählt wurden. Die Dokumentationstechnologien, welche ausgewählt werden, sollen hinsichtlich relevanter Eigenschaften für die Software "yeet" verglichen werden.
Eine Verfahrensauswahl wird anschließend anhand der Recherche- und Umfrageergebnisse sowie an den Anforderungen und dem Vergleich der Technologien getroffen.
Das gesamte Konzept wird am Ende kritisch analysiert und bewertet werden. Im Fazit werden die Ergebnisse der gesamten Arbeit nochmals vorgestellt und in einem Ausblick werden die weiteren Vorgehensweisen umfassend erläutert.

1.4 Zielsetzung

Durch diese Arbeit soll dem Leser die Relevanz von Anwenderdokumentationen, in der heutigen Zeit verdeutlicht werden. Darüber hinaus sollen grundlegende Verständnisse zum Thema "technische Dokumentationen" sowie der Unterschied von API- und Low-Code-Dokumentationen vermittelt werden. Relevante Anforderungen an Low-Code-Dokumentationen sowie die passende Auswahl einer Dokumentationstechnologie sollen ausreichend begründet aufgeführt werden.
Ziel dieser Arbeit ist es, ein optimiertes Dokumentationskonzept, welches auf die Anforderungen der Low-Code-Software "yeet" zugeschnitten ist, zu erschaffen. Diese Arbeit soll nicht nur einen Nutzen für den Leser haben, vielmehr soll sich das Produkt "yeet" dadurch in Zukunft positiv von seiner Konkurrenz abheben können.

1.5 Aufbau der Arbeit

Das nachfolgende Kapitel *Stand der Technik* (Kapitel 2) wird sich mit den verwandten Arbeiten zu dem Thema sowie um den aktuellen Stand der Technologien zu Dokumentationssoftwares und dem Thema Low-Code befassen. Es werden einige Arbeiten zum Thema API-Dokumentation aufgeführt und erläutert, mit welchen Problemstellungen sich diese beschäftigen. Ebenso wird erklärt, warum die Erkenntnisse aus den verwandten Arbeiten noch nicht genügen, um das Problem dieser Abschlussarbeit zu lösen. Der Abschnitt *Stand der Technik* soll dem Leser die aktuellen Technologien darlegen, mit denen Dokumentationen für Softwares umgesetzt werden. Ebenfalls wird die Relevanz von Anwenderdokumentationen in Bezug auf die Benutzerfreundlichkeit einer Software herausgestellt. An dieser Stelle wird kurz aufgezeigt, inwiefern das Thema Low-Code-Plattformen immer präsenter in der Softwareentwicklung wird.

Im *Grundlagenkapitel* (Kapitel 3) werden dem Leser alle nötigen Informationen zu den wichtigsten Themengebieten vermittelt. Hier findet er das Fachwissen, welches Voraussetzung für das Verständnis der Arbeit ist.
Danach geht es um die *Analyse der Low-Code Dokumentationen* (Kapitel 4). In diesem Kapitel werden die Anforderungen der verwandten Arbeiten herausgearbeitet und aufgeführt. Des Weiteren werden verwendete Umfrage- und Analysemethoden erklärt sowie die Konkurrenzprodukte vorgestellt, deren Dokumentationen in dem User-Test vorkamen.
Im Kapitel *Analyse der Tests* (Kapitel 5) wird der User-Test sowie die Umfrage analysiert. Die Auswertung der Ergebnisse erfolgt im nächsten Kapitel.
Erstellung der funktionalen Anforderungen (Kapitel 6), dieses Kapitel behandelt die Ergebnisse des vorherigen und erstellt daraus funktionale Anforderungen, in Bezug zur Low-Code-Dokumentation. Diese werden anfänglich nach "must-have"- und "nice-to-have"-Anforderungen sortiert, um danach einen Vergleich dieser Anforderungen mit denen der verwandten Arbeiten zu erstellen.
Bevor das finale Konzept vorgestellt wird, bedarf es der optimalen Dokumentationstechnologie. Die Auswahl dieser erfolgt in Kapitel *Auswahl der Technologien* (Kapitel 6). Zuerst wird dort die genutzte Methode der Technologiematrix erklärt. Danach werden die beiden "besten" Programme vorgestellt sowie deren Stärken und Schwächen in Bezug auf die Software "yeet" herausgearbeitet.
Im Kapitel *finales Konzept* (Kapitel 8) findet sich das fertige Konzept der Dokumentation. Es wird erklärt wie und warum das Problem gelöst wurde und das entstandene Konzept wird vorgestellt.
Das letzte Kapitel behandelt die *Zusammenfassung* und den *Ausblick*. Die Zusammenfassung gibt die gesamte Arbeit nochmals kurz wieder und reflektiert die Ergebnisse. Im Ausblick wird das weitere Vorgehen der Firma vectorsoft AG mithilfe des Dokumentationskonzeptes erläutert. Außerdem werden entstandene Probleme der Arbeit reflektiert und mögliche Lösungsansätze aufgezeigt. Neue Konzepte und Ideen, welche beim bearbeiten der Arbeit entstanden, werden ebenfalls in diesem Kapitel kurz vorgestellt.

Kapitel 2

Stand der Technik

2.1 Verwandte Arbeiten

In der Literatur finden sich schon einige Arbeiten über das Entwickeln optimierter API-Dokumentationen. Da die Dokumentation, welche durch diese Arbeit konzeptioniert werden soll, keine reine API-Dokumentation sein wird, sind diese anderen Werke zwar nützlich, beinhalten jedoch nicht explizit diese Informationen, die gebraucht werden. Die Dokumentation, welche erarbeitet werden soll, ist für die Low-Code-Anwendung "yeet" gedacht. Low-Code-Plattformen beinhalten mehr Funktionalitäten als reine API's. Ob die Anforderungen an eine Low-Code-Dokumentation sich aus diesem Grund zu den Anforderungen an API-Dokumentationen unterscheiden, wird daher im Laufe dieser Arbeit untersucht. Trotz allem enthalten die verwandten Arbeiten über API-Dokumentationen hilfreiche Feststellungen und da es einige Punkte gibt, in denen sich API-Dokumentationen mit der angestrebten Low-Code Variante überschneiden, werden diese dennoch als verwandte Arbeiten vorgestellt. Ebenso gibt es noch keine Untersuchungen, welche optimierte Eigenschaften einer Low-Code Dokumentation aufzeigen, weshalb diese verwandten Arbeiten ebenso als Hauptinformationsquelle für die spätere Anforderungsanalyse der vorherigen Arbeiten sein werden.

Darunter fällt zum Beispiel die Arbeit "API Documentation - A Conceptual Evaluation Model"[IJRJ18] von Inzunza S.,Juarez-Ramirez R. und Jimenez S., welche 2018 erschien. Die Autoren untersuchten die grundlegenden Eigenschaften der API-Dokumentationen, gewichteten diese nach Relevanz und entwickelten daraus ein konzeptuelles Modell, zur Bewertung von API- Dokumentationen [IJRJ18, vgl. Seite 229]. Um dies zu bewältigen, studierten sie am Anfang bereits vorhandene Arbeiten zu diesem Thema und kristallisierten so die grundlegenden Eigenschaften heraus, welche jede API-Dokumentation mindestens besitzen sollte [IJRJ18, vgl. Seite 230 ff.]. Laut den Autoren seien die wichtigsten Eigenschaften:

- API Übersicht
- Anleitung für den Einstieg
- Beispielcode

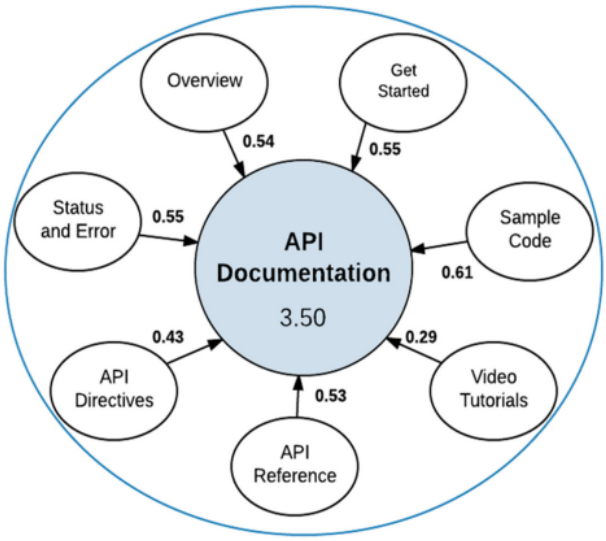

Fig. 1. Basic documentation elements and its importance

Abbildung 2.1: API Documentation Evaluation Model von Inzunza S. et al, Quelle: [IJRJ18, Seite 234.]

- Video-Tutorials
- API-Referenzen dokumentieren
- die API-Richtlinien dokumentieren
- die Status- und Fehlercodes dokumentieren

[IJRJ18, vgl. Seite 231 f., eigene Übersetzung].

Nachdem diese Daten gesammelt wurden, starteten sie eine Umfrage. Bei dieser Umfrage nahmen 95 Softwareentwickler teil, welche die oben genannten Eigenschaften jeweils in fünf Stufen, von "sehr Wichtig" bis "nicht Wichtig", einteilen sollten. Durch diese Umfrage fanden sie heraus, dass "Beispielcode" und "Anleitung für den Einstieg" die wichtigsten Eigenschaften sind und "Video-Tutorials" die unbedeutendste. Mithilfe dieser Erkenntnisse entwickelten sie ein Modell, welches zur Bewertung einer API-Dokumentation verwendet werden kann. Dieses ist in Abbildung 2.1 zu sehen. Inzunza S. et al. gaben in diesem Modell allen Eigenschaften eine Punktzahl zwischen 0 und 3.50, basierend auf den vorher herausgefundenen Signifikanzen. Danach testeten sie ausgewählte API-Dokumentationen mit diesem Modell und erhielten als Ergebnis jeweils ebenfalls einen Wert zwischen 0 und 3.50, mit dem sich die API-Dokumentationen verglichen ließen. Inzunza S. et al. haben mit dieser Arbeit, allgemeine Richtlinien aufgezeigt, welche helfen sollen, gute API-Dokumentationen

zu bauen. Weiterhin haben sie ein Modell konzipiert, welches die Möglichkeit eröffnet, API-Dokumentationen untereinander zu bewerten. Für die Zukunft stellen sie sich vor, dass es ein Programm gibt, welches selbstständig API-Dokumentationen bewerten kann. Dies würde funktionieren, indem es alle Seiten der Dokumentation prüft und bestimmt, ob die Dokumentation diese grundlegenden Elemente aufweist.
Die Arbeit ist sehr hilfreich, wenn es darum geht Anforderungen einer API-Dokumentation zu bestimmen. Ebenso zeigt sie einen Weg auf, diese Dokumentation sogar zu bewerten. Für die gegenständliche Arbeit muss jedoch erst herausgefunden werden, ob Dokumentationen für Low-Code Plattformen dieselben Anforderungen besitzen wie API-Dokumentationen. Bis dahin kann nicht davon ausgegangen werden, dass dieser Fall gegeben ist. Des Weiteren wird in der vorliegenden Arbeit, ein Konzept für die Dokumentation einer realen Anwendung entwickelt, inklusive Technologieauswahl. Auch diesen Aspekt kann die Arbeit von Inzunza S.,Juarez-Ramirez R. und Jimenez S. nicht erfüllen.

Die Arbeit von Huesmann, R., Zeier, A. & Heinemann, A., welche die "Eigenschaften optimierter API-Dokumentationen im Entwicklungsprozess sicherer Software"[HZH19] behandelt haben, erschien im Jahr 2019. Huesmann et al. haben sich in ihrer Arbeit besonders auf die Sicherheit der Software konzentriert und wollten eine optimierte API-Dokumentation erstellen, um fehlerhaften Code zu minimieren. Denn oft werden von Entwicklern Codes aus unsicheren Internetquellen kopiert, anstatt nach Lösungen in der Dokumentation zu suchen [HZH19, vgl. Seite 232]. Mithilfe eines Brainstormings konnten fünf Dokumentationsarten festgestellt werden: "API Spezifikation des Herstellers, Blog, textuelles Tutorial, Video-Tutorial und Frage-und-Antwort-Plattform (Q&A Plattform)" [HZH19, Seite 233]. Dann haben sie in Fokusgruppen 26 Programmierer an sieben Tagen über die Vor- und Nachteile dieser Dokumentationsarten befragt[HZH19, vgl. Seite 234].
Letztlich wurde von Huesmann, R. et al. die Eigenschaften einer, aus Sicht der Probanden, optimierten API-Dokumentation gesammelt. Sie haben herausgefunden, dass solch eine Dokumentation **viele Beispiele** haben sollte, **gut strukturiert** und **leicht über Google auffindbar** sein sollten. **Verschiedene Niveaus** und eine **IDE Integration** wären ebenso ein wichtiger Bestandteil der Dokumentation. Weitere wichtige Eigenschaften wären, **ein Tutorialbereich**, **eine klassische Referenz**, **Videos für den schnellen Überblick** und zu jedem Themenbereich bedarf es der **Möglichkeit für Fragen, Antworten und Diskussionen**. Für diese Diskussionen sollte es ein **Bewertungssystem** geben, welches gute Lösungen aus dieser, in die Beispiele einfließen lassen sollte. Ebenso **Hintergrundwissen zu der API**, **Negativbeispiele** und **Einblick in den Sourcecode** wurde sich von einigen Probanden gewünscht.[HZH19, vgl. Seite 237]
Die von Huesmann, R. et al. herausgearbeiteten Eigenschaften einer optimierten Dokumentation sind lediglich als Ergebnis verschiedener Studien mit API-Dokumentationen entstanden. Mit Dokumentationen von Low-Code-Software wurde nicht getestet. Eine Dokumentation für eine Low-Code-Software besitzt auch einen Anteil, welcher durch eine API-Dokumentation umsetzbar wäre. Jedoch muss eine Dokumentation einer Low-Code-Software auch andere Teile, als nur die API abdecken und darauf wird in der vorgestellten Arbeit nicht eingegangen. Dennoch sind Teile der Ergebnisse sehr interessant und es kann angenommen werden, dass viele Eigenschaften einer guten API-Dokumentation auch bei einer guten Low-

Code-Dokumentation zu finden sein sollten.

Robillard, M.P. und DeLine, R. haben in ihrer Arbeit "A field study of API learning obstacles"[RD11] untersucht, welche Lernhindernisse es für Entwickler beim Erlernen einer neuen API gibt. Dafür haben sie insgesamt 440 professionelle Microsoft-Entwickler in mehreren Studien befragt und sind zu dem Schluss gekommen, dass die größten Hindernisse für Entwickler beim Erlernen neuer APIs die Dokumentation und andere Lernressourcen sind [RD11, vgl. Seite 703]. Daraufhin untersuchten sie die Erschwernisse mit besonderem Schwerpunkt auf Hindernisse im Zusammenhang mit der API-Dokumentation.
Ihre erste Studie war eine Umfrage, in der die Probanden Fragen über ihre jüngsten Lernerfahrungen mit einer öffentlich freigegebenen API beantworten mussten [RD11, vgl. Seite 708]. Danach führten sie mit einer kleineren Gruppe an Testpersonen qualitative Interviews, in denen diese ihre Arbeit mit der API zusammenfassen und die Hindernisse, auf die sie gestoßen sind, erklären sollten [RD11, vgl. Seite 709]. Zuletzt führten Robillard, M.P. und DeLine, R. noch eine Folgebefragung durch, welche sie mithilfe des Wissens aus den vorherigen Studien erarbeiteten. In dieser sollten die Probanden abermals Fragen zu ihren Erfahrungen mit der zuletzt gelernten API beantworten sowie ihre eigenen Meinungen dazu darlegen, wie eine API-Dokumentation zu verbessern wäre [RD11, vgl. Seite 711 ff. + Seite 727]. Durch all diese Studien konnten Robillard, M.P. und DeLine, R. fünf wichtige Faktoren identifizieren, welche beim Designen einer API-Dokumentation berücksichtigt werden sollten. Diese wären,

- die Absicht der Dokumentation
- Code-Beispiele
- Abstimmung von APIs mit Szenarien
- die Durchlässigkeit der API
- Format und Präsentation der API-Dokumentation

[RD11, vgl. Seite 703].
Ebenso wie die vorherige Arbeit, bezieht sich dieses Werk auch explizit auf API-Dokumentationen. Des Weiteren muss berücksichtigt werden, dass diese Studien im Jahr 2011 durchgeführt worden sind. Gerade Technologien verändern sich sehr schnell, sodass nicht gesagt werden kann, ob dieselben Studien aus heutiger Sicht zu völlig anderen Erkenntnissen kämen. Trotzdem wurden grundsätzliche Erschwernisse des Lernens und auch grundsätzliche Lösungen für diese Probleme vorgestellt, welche sich nicht signifikant verändert haben sollten.

In der Arbeit von Watson R., Stamnes M., Jeannot-Schroeder J. und Spyridakis J.H. geht es um "API Documentation and Software Community Values: A Survey of Open-Source API Documentation". Die Studie erschien 2013, ist folglich auch etwas älter, jedoch behandelt sie ein sehr interessantes Thema. Diese Arbeit vergleicht Dokumentationen, erstellt von Open-Source-Software-Communitys, mit den Wünschen und Bedürfnissen, welche Software Entwickler an API-Dokumentationen hätten [WSJSS13, vgl. Seite 165]. Dazu forschten sie

zunächst in früheren Studien zu API-Dokumentationen nach Eigenschaften, die eine Dokumentation unbedingt besitzen sollte. Folgende Punkte bildeten sich dabei bei fast allen Studien ab,

- Übersichtsdokumentation
- kurze Codeschnipsel, welche die Verwendung einer API im Kontext demonstrieren
- Codebeispiele, die bewährte Verfahren mit einer API zeigen
- Szenario- und aufgabenbezogene Dokumentation
- Begrenzung und Fehlerbehandlung
- 'aussagekräftige' Dokumentation
- Genauigkeit, Vollständigkeit und Korrektheit
- Szenario- und aufgabenbezogene Beispiele
- Inhalte, die nicht das Offensichtliche wiederholen, z. B. was man von der Benutzeroberfläche lernen kann

[WSJSS13, Seite 166, eigene Übersetzung].
Danach erstellten sie eine Liste von insgesamt 33 zu testenden Dokumentationen, aus den hundert populärsten Open-Source-Softwares dieser Zeit. Diese Dokumentationen prüften sie dann auf die vorher erforschten Eigenschaften. Hierfür teilten sie die zu testenden Eigenschaften in drei Kategorien ein und erarbeiteten sich mithilfe vierer Tester, einer aufwändigen Methode und vieler Tabellen die nötigen Informationen [WSJSS13, vgl. Seite 167 ff.]. Zusätzlich untersuchten sie die Design und Schreibqualität der verschiedenen Dokumentationen [WSJSS13, vgl. Seite 169 ff.]. Als Ergebnis können Watson et al. aufzeigen, dass Dokumentationen, welche von Open-Source-Software-Communitys erstellt wurden, fast alle oder zumindest die meisten Eigenschaften besitzen, welche sich in anderen Studien über optimierte API-Dokumentationen gewünscht wurden [WSJSS13, vgl. Seite 172]. Damit festigen sie die erforschten Eigenschaften der anderen Arbeiten, indem sie den tatsächlichen Gebrauch - durch Open-Source-Communitys - aufzeigen. Dass sich diese Arbeit nur um Open-Source-Dokumentationen dreht, ist nicht tragisch für die Übertragung auf andere API-Dokumentationen. Es wurde schon oft belegt, dass es eine große Überlappung zwischen Open-Source-Entwicklern und professionellen, kommerziellen Software-Entwicklern gibt [WSJSS13, vgl. Seite 172]. Das heißt, die erarbeiteten, festen Eigenschaften gelten gleichermaßen. Eine weitere interessante Entdeckung ist, dass neben diesen Eigenschaften auch das Design und die Schreibqualität einer Dokumentation von Bedeutung sind. Denn zumindest werden sie von den Softwareentwicklern so geschätzt, dass fast immer deutlich auf sie geachtet wurde [WSJSS13, vlg. Seite 172].
Die Problemstellung der vorliegenden Arbeit, löst die Arbeit von Watson et al. nicht. Ebenso wie die vorherigen Arbeiten bezieht sich diese wieder hauptsächlich auf API-Dokumentationen.

Wobei sie spezifisch Open-Source-Dokumentationen behandelt, welche nicht so viele Unterscheidungen zu normalen API-Dokumentationen besitzen. Trotzdem zeigt sie, dass in Studien gesammelte und gewünschte Eigenschaften, durchaus auch in der Realität essentielle Kriterien sind und nicht unbeachtet bleiben sollten. Diese Arbeit bestätigt den Sinn eines Dokumentationskonzepts, nämlich dass es durchaus ein wichtiger Schritt in Richtung Erschaffung einer Low-Code Dokumentation ist.

Die nächste Arbeit bezieht sich ebenfalls auf Dokumentationen von Open-Source-Projekte. "Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors", herausgegeben 2010 von Dagenais B. und Robillard M.P.. Auch dieses Werk ist etwas älter, jedoch beleuchtet es Eigenschaften einer Dokumentation von einer etwas anderen Seite. Die Autoren untersuchen Open-Source-Software-Dokumentationen unter der Fragestellung, wie diese erzeugt und gewartet werden.
Dazu haben sie Entwickler interviewt, welche solche Dokumentationen schreiben und andere Entwickler, welche solche Dokumentationen lesen [DR10, Seite 127]. Sie fanden interessante Erkenntnisse durch Recherchen der Open-Source-Projekte. Eine davon ist: "Wir haben beobachtet, wie die Aktualisierung der Dokumentation bei jeder Änderung zu einer Art 'penibler Entwicklung' führte, was wiederum eine Verbesserung der Codequalität zur Folge hatte."[DR10, Seite 127, eigene Übersetzung]. Daraus folgerten sie, dass es besser für Entwickler sei, ihre Änderungen an der Software schnell zu dokumentieren, um sich dadurch diese Gegebenheit zunutze zu machen und die Codequalität zu verbessern [DR10, vgl. Seite 135]. Eine andere Beobachtung war, dass fast alle Open-Source-Projekte, welche anfangs ihre Dokumentation in einer öffentlichen Dokumentationssoftware betrieben, aufgrund der hohen Wartungskosten und der geringeren Autorität der Dokumentation irgendwann zu einer besser kontrollierbaren Dokumentationsinfrastruktur wechselten [DR10, vgl. Seite 127]. Als Hauptmethode in ihrer Arbeit nutzen Dagenais und Robillard die "Grounded Theory" von Corbin und Strauss [SC98]. Sie nutzen drei unterschiedliche Quellen für ihre Datensammlung. Daten von Interviews mit Personen, welche zu Open-Source-Projekten und deren Dokumentationen aktiv Informationen beitragen. Daten von Interviews mit Personen, welche Open-Source-Projekten und deren Dokumentationen aktiv benutzen und als dritte Quelle analysierten sie die Entwicklung von 19 Dokumentationen, aus 10 Open-Souce-Projekten [DR10, vgl. Seite 128]. Durch all diese Daten fanden sie zusätzlich heraus, dass es unterschiedliche Stufen in der Entstehung einer Dokumentation gibt, welche alle unterschiedlich aufwendig sind. Die Stufen, welche mit dem größten Arbeitsaufwand verbunden sind, wären der "Initial Effort", welcher mit dem Start des Projektes beginnt, und die "Bursts", welche durch große Veränderungen im Projekt selbst entstehen. Dazwischen gäbe es immer wieder "Incremental Changes", diese bedeuten jedoch nur kleine, unkomplizierte Änderungen in der Dokumentation [DR10, vgl. Seite 130]. Laut Dagenais und Robillard, hilft dieses Verständnis anderen Forschern dabei, bessere Dokumentationstechniken zu entwickeln, welche besser zu dem Entstehungsprozess von Open-Source-Dokumentationen passen [DR10, vgl. Seite 135]. Wie anfangs erwähnt, beleuchtet diese Arbeit Probleme, welche beim Hinzufügen der Informationen zu einer Dokumentation bedeutsam sein könnten. Für das Einfügen und Pflegen der Informationen hilft die Studie zu verstehen, mit welchen Methoden dies am besten gelingt. Jedoch sind dies Dinge, welche nicht direkt für das Hauptkonzept wichtig sind. Sie

werden erst dann gebraucht, wenn dieses vollständig durchdacht ist. Dennoch gibt es in dieser Arbeit interessante Erkenntnisse, welche nützlich für die Auswahl der Technologie sind. Das Werk von Dagenais und Robillard löst die Probleme dieser Arbeit nicht, enthält jedoch wissenswerte Informationen für das weitere Vorgehen.

2.2 Stand der Technologien

2.2.1 Dokumentationstechnologien

Durch den Fakt, dass Dokumentationen immer wichtiger für das Gesamtbild einer guten Software wurden, sind immer mehr Technologien für das Erstellen von Dokumentationen entstanden.
Früher wurde zu einer Software ein entsprechendes Handbuch mitgeliefert. Dort waren alle wichtigen Informationen zu finden. Auch war es üblich in der Anwendung selbst eine Möglichkeit zu haben die Dokumentation aufzurufen. Dies ist auch heutzutage noch üblich. Mit "Shift+F1" oder auch nur "F1" wird beispielsweise die Hilfe einer Anwendung aufgerufen, falls diese vorhanden ist [Sie14, vgl. Seite 5]. Entweder öffnet sich eine eigene Anwendung, welche die Dokumentation beinhaltet, oder es wird eine Website geöffnet, die diesen Zweck erfüllt. Lange Zeit wurde die eigene Anwendung, welche sich öffnet, oft durch das Dateiformat CHM (Compiled HTML Help oder Compiled Help Module(s)) realisiert. Das Programmentwicklungssystem "Conzept 16" von der vectorsoft AG bedient sich auch dieser Dokumentations-Option.
Die meisten moderne Anwendungen sind zurzeit Webanwendungen. Ebenso die neue Software der "yeet", welche in Kapitel 3.4 näher erklärt wird, wird eine webbasierte Low-Code-Plattform sein. Solche online Anwendungen brauchen demzufolge auch Dokumentationslösungen im Internet. Jedoch auch andere lokal installierte Anwendungen setzen neuerdings auf Onlinedokumentationen. Der klare Vorteil besteht darin, die Dokumentation schnell und einfach für alle Nutzer anzupassen, ohne dass ein Update für den Endkunden erforderlich ist. Daraus folgt jedoch der Nachteil, dass die Dokumentation immer online sein muss. Gibt es einen Serverausfall oder Ähnliches, können die Nutzer nicht darauf zugreifen. Ebenso müssen die Anwender eine Internetverbindung besitzen, um die Dokumentation erreichen zu können. Durch herunterladbare PDF-Dokumente, welche die Dokumentation ebenso beinhalten, kann dieser Nachteil umgangen werden.
Eine weitverbreitete Technologie, um Dokumentationen umzusetzen, ist die Auszeichnungssprache "Markdown"[1]. Mithilfe dieser können einfache und lesbare Dokumente erstellt werden, welche im Web veröffentlicht werden können. Ziel von Markdown ist es, so einfach lesbar und schreibbar wie möglich zu sein [Mar, vgl. Absatz 1]. Dies erzielen sie, indem nur Zeichen verwendet werden, deren Aussehen ihrer Bedeutung entspricht. Zur Veranschaulichung sind in Abbildung 2.2 einige Beispiele abgebildet. Ein Vorteil von Markdown ist, dass HTML-Elemente einfach in die Markdowndatei eingebunden werden können. Markdown ist zum einen eine Syntax zur Formatierung von einfachem Text und zum anderen ein in Perl[2]

[1] https://markdown.de/
[2] https://www.perl.org/

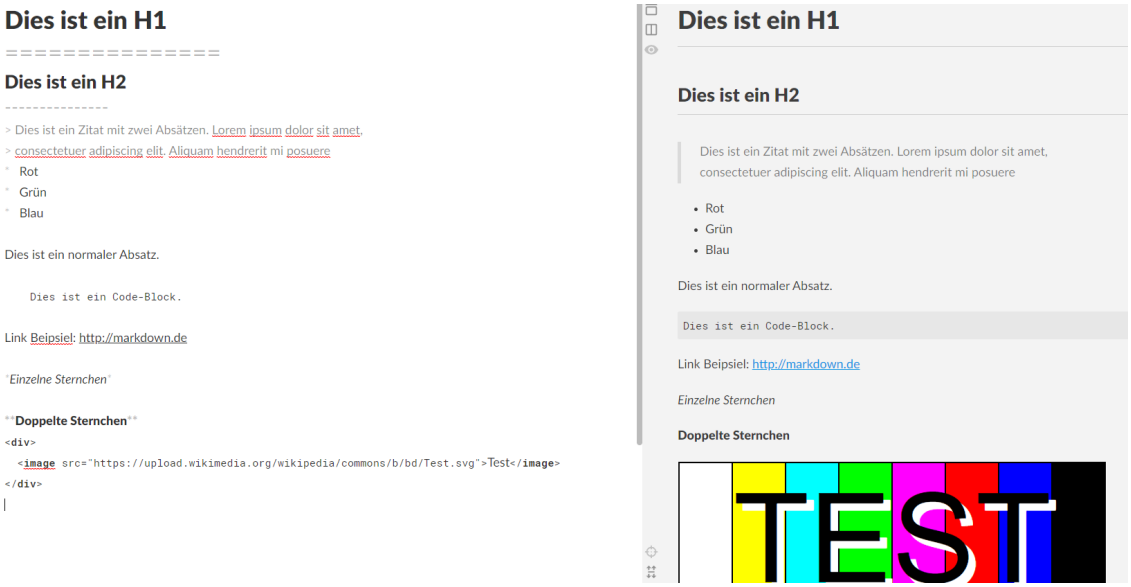

Abbildung 2.2: Beispiele Markdown, Quelle: Eigene Darstellung

geschriebenes Softwaretool, das die einfache Textformatierung in HTML umwandelt [Fir, vgl. Absatz 1]. Durch seine einfache Schreibweise ist es auch Menschen möglich, ohne große Programmierkenntnisse Dokumentationen zu schreiben. Viele Webanwendungen, mit denen sich Dokumentationslösungen umsetzten lassen, bieten einen eigenen Markdown-Editor oder zumindest die Möglichkeit Markdown Files zu importieren.
Es gibt noch andere Arten, wie Dokumentationen erstellt werden können. In Bezug auf yeet ist eine besonders interessante Variante "Vuepress"[3]. Vuepress kombiniert das beliebte Framework "Vue.js"[4] mit der soeben vorgestellten Auszeichnungssprache Markdown. Für yeet ist es so interessant, da die Low-Code-Plattform ebenfalls zum Teil mit dem Framework Vue.js realisiert wird. VuePress bietet somit die Möglichkeit, in die bestehende Entwicklungsumgebung eingebunden zu werden und dort genutzt zu werden. Dies würde es einfacher machen in der Dokumentation interaktive Beispiele anzuzeigen, da die Komponenten schon im Projekt vorhanden sind. Ein klarer Nachteil von Vuepress ist jedoch die Schwierigkeit für "Nicht-Programmierer" sich im Projekt zurechtzufinden. Außerdem müssen dafür die Projektstrukturen installiert und zugänglich sein. Die Möglichkeit sich in eine Online-Anwendung einzuloggen und etwas an der Dokumentation zu verändern, besteht nicht. Des Weiteren muss die Dokumentation bei VuePress von Grund auf neu gestaltet werden sowie alle Features müssen selbst programmiert werden. Es gibt zwar einige Plugins, welche weitere Features anbieten, jedoch müssen diese auch erst einmal installiert werden. Eine Dokumentation mit VuePress zu realisieren, benötigt also Zeit- und Programmieraufwand. Wenn das Grundgerüst jedoch besteht, beinhaltet es nützliche Eigenschaften, gerade wenn

[3] https://vuepress.vuejs.org/
[4] https://vuejs.org/

es um Vue.js Merkmale geht, um die Arbeit zu simplifizieren.
Eine populäre Art Informationen im Internet zu erschaffen sind Wiki-Softwares. "Ein Wiki ist eine Web-Software, mit deren Hilfe Seiten im Web erstellt und bearbeitet werden können, die von mehreren Autor/innen bzw. Benutzer/innen bearbeitet werden können."[Cas16, Absatz 1]. Wiki-Softwares werden heutzutage auch als Dokumentationstechnologie eingesetzt. Meistens jedoch in Open-Source-Anwendungen, damit jeder problemlos seinen Betrag leisten kann. Wikipedia[5] beispielsweise läuft mit einer Wiki-Software, nämlich MediaWiki[6]. Ein weiteres bekanntes Tool dafür ist die Open-Source-Software Wiki.js[7]. Diese Wiki-Software bietet jedoch weit mehr Features an, als die Definition von Wiki-Softwares beschreibt. Wie bei VuePress handelt es sich hier um eine leistungsfähige Software, welche jedoch noch weiter Entwickelt ist. Sie bietet dem Entwickler Möglichkeiten wie Multilingualität, ein Login-System, oder User Management, welche sonst nur bei kostenpflichtigen Anbietern zu finden sind. Dennoch ist Wiki.js noch in der Entwicklung und einige der Features sind zwar für die Zukunft versprochen, jedoch zurzeit noch nicht vorhanden. Des Weiteren verhält es sich so wie bei VuePress. "Nicht-Programmierer" werden sich schwierig in der Software zurechtfinden. Somit können nur Programmierer, nach einer vorherigen Einarbeitung, effizient damit arbeiten.

2.2.2 Usability und Dokumentationen

"Dokumentation = Usability Merkmal" so heißt eine Überschrift der Arbeit von Biesterfeldt [Bie06]. Die Usability eines Produktes oder einer Anwendung bezieht sich auf die Benutzerfreundlichkeit dieses Produktes.
In seiner Arbeit stellte er damals schon die Vermutung auf, dass die Usability eines Programms stark mit der Dokumentation verknüpft sei. Dies schließt er daraus, dass Usability und die Anwenderdokumentation die gleichen Ziele hätten - den Anwendern die optimale Nutzung des Produktes zu ermöglichen [Bie06, vgl. Seite 62]. Die Usability eines Produktes, gerade wenn es sich um eine Software handelt, kann durch viele Eigenschaften beeinflusst werden. Eine hohe Usability garantiert, dass der Nutzer die Software möglichst intuitiv bedienen kann. Es existieren extra "Usability Tests" in denen ausschließlich die Interaktionen der Tester mit der Software geprüft werden. Eine hohe Usability wirkt sich ebenfalls positiv auf den Umfang der Dokumentation aus. "In der Theorie reduzieren Usability Maßnahmen am Produkt den Umfang der Dokumentation. Sie sollen helfen, das Produkt intuitiv bedienbar zu machen. Es beschreibt sich selbst."[Bie06, Seite 62].
Laut Biesterfeldt gibt es wesentliche Argumente, welche für die Berücksichtigung der Dokumentation im benutzerzentrierten Designprozess sind. Diese sind:

- "Der Stellenwert der Dokumentation für die Usability interaktiver Produkte steigt."[Bie06, Seite 62]
- "Eine enge Zusammenarbeit von Usability- und Dokumentations-Spezialisten zahlt sich für beide und damit auch für den Anwender aus."[Bie06, Seite 62]

[5] https://de.wikipedia.org/wiki/Wikipedia:Hauptseite
[6] https://www.mediawiki.org/wiki/MediaWiki
[7] https://js.wiki/

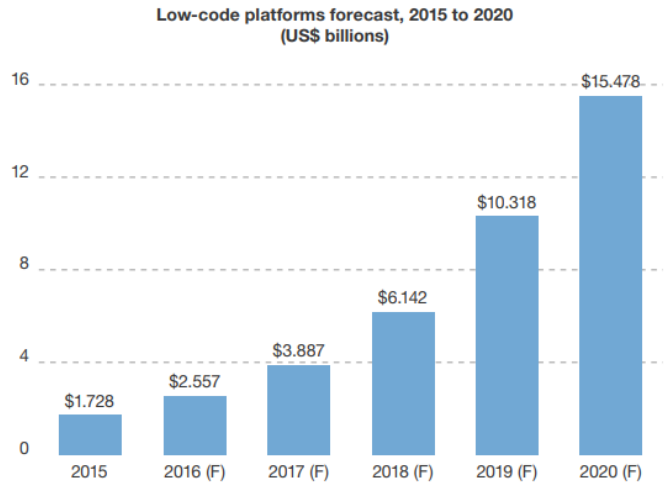

Abbildung 2.3: prognostiziertes Wachstum des Low-Code Markts in Milliarden $, Quelle:[RR16, Seite 16]

- "Häufig lassen sich GUI und Dokumentation gar nicht mehr trennen. Wo ziehen wir die Grenze? Sind Fehlermeldungen nicht auch Dokumentation? Überschriften im GUI, die zu einer Handlung auffordern („Klicken Sie hier")? Was ist mit Tooltipps?"[Bie06, Seite 62]

Auch heutzutage gelten die Annahmen von Biesterfeldt noch und bei der Entwicklung einer Software sollte eine Dokumentation nicht benachteiligt werden. Eine gute Usability ist sehr wichtig in der Softwareentwicklung [Soc04, vgl. Seite 2-3]. Da die Usability stark mit der Dokumentation zusammenhängt, kann dies auf für die Dokumentation angenommen werden.

2.2.3 Low-Code Plattformen

Im Laufe der Jahre fanden Low-Code-Plattformen immer mehr Zuspruch in der Systementwicklung. Durch ihre einfache Bedienung bieten sie einem breiten Publikum die Möglichkeit eigene Systeme zu entwickeln. Die Anforderungen der heutigen Zeit an die Entwicklung einer Software haben sich stark vermehrt, auch bedingt durch die immer weiter fortschreitende Digitalisierung von Geschäftsprozessen. Auch die Corona Pandemie hat bewiesen, dass die rasche Entwicklung von Anwendungen und die ständige Iteration von Software zu einer Selbstverständlichkeit geworden ist [BK21, vgl. Seite 2]. Das Marktforschungsinstitut Forrester[8] hat schon 2016 eine Prognose erstellt, welches das Wachstum des Low-Code-Marktes in Milliarden $ darstellen soll (siehe Abbildung 2.3).
Dort ist zu erkennen, dass der Umsatz von 2015 bis 2020 um das 9-Fache ansteigen sollte. Und dieses Wachstum soll nicht aufhören. 2019 veröffentlichte dieselbe Firma einen weiteren

[8] https://www.forrester.com/bold

Bericht, welcher unter anderem eine Umfrage enthielt. In unserer Umfrage unter globalen Entwicklern gaben 23% an, im Jahr 2018 Low-Code-Plattformen genutzt zu haben, und weitere 22% planten, dies innerhalb eines Jahres zu tun [RK19, vgl. Seite 2].
Low-Code-Plattformen werden immer wichtiger in der Systementwicklung. Aus diesem Grund sollte es Dokumentationen geben, welche die Anforderungen von Low-Code-Anwendungen optimal unterstützen. An diesem Punkt setzt die vorliegende Arbeit an. Zu untersuchen, welche Anforderungen umgesetzt werden müssen und ein Dokumentationskonzept speziell für eine Low-Code-Plattform zu erarbeiten.

Kapitel 3

Grundlagenkapitel

3.1 Technische Dokumentationen Grundlagen

3.1.1 Begriffserklärung

Eine Dokumentation kann es für viele Fälle geben. Wie in Kapitel 1.2 beschrieben, benötigen viele alltägliche Dinge eine Dokumentation. Durch die Weiterentwicklung der Technik und zugleich sinkendem Bildungsniveau, werden immer mehr gute Anleitungen gebraucht [Kot11, vgl. S.1].
Viele Prozesse laufen heutzutage nicht mehr manuell, sondern über verschiedene Anwendungen. Durch diese immer anspruchsvolleren Programme, welche meist in kürzester Zeit zur Verfügung stehen müssen, werden auch sehr viele Dokumentationen gebraucht. Diese sind nötig, um die Bedienung der Anwendung dem Nutzer zu erläutern und ihm bei möglichen Fragen zu helfen. "Die steigenden Anforderungen an betriebswirtschaftliche IT-Systeme spiegeln sich auch in der technischen Dokumentation wieder"[BM07, Seite 1], fanden Birn und Müller schon 2007 heraus.
Ein weiterer wichtiger Punkt sind auch die gesetzlichen Forderungen an Technische Dokumentation, denn diese sind in den letzten Jahren immer wichtiger geworden [Kot11, vgl. S.1]. Es gibt Normen wie die VDI-Richtlinie 4500[1], welche sich spezifisch auf den Dokumentationsprozess Technischer Dokumentationen bezieht.

Der Begriff Technische Dokumentation ist jedoch leider oft irreführend. Bis heute werden oft unterschiedliche Dinge gemeint, da es noch keine eindeutige Definition gibt [Kot11, vgl. Seite 2]. Kothe meinte 2011 in seinem Buch, "Technische Dokumentation meint eigentlich alle Dokumente, die dazu dienen, ein Produkt in seinen Eigenschaften zu beschreiben" [Kot11, Seite 2]. Dabei würde zwischen der internen und externen Dokumentation unterschieden werden [Kot11, vgl. Seite 2].
Interne Dokumentationen sind dabei alle Aufzeichnungen, welche intern in einer Firma über das neu zu entwickelnde Produkt gesammelt werden. Dazu zählen, Technische Zeichnungen, Stücklisten, Arbeitspläne, Arbeitsanweisungen, etc.[Kot11, vgl. Seite 2]. In einer Softwarefir-

[1] https://www.vdi.de/fileadmin/pages/vdi_de/redakteure/richtlinien/inhaltsverzeichnisse/1810513.pdf

ma wären dies zum Beispiel Softwaredokumentationen, welche alle wichtigen Informationen über die zu erschaffende Anwendung enthalten.
Zu den externen Dokumentationen gehören alle Beschreibungen, welche für den Kunden, beziehungsweise Nutzer des Programms, gedacht sind [Kot11, vgl. Seite 3]. Um das Beispiel der Softwarefirma weiterzuführen, wären dies "Online-Hilfen" oder ein Benutzerhandbuch in Form einer Anwenderdokumentation für die Kunden.

Um eine technische Dokumentation zu schreiben ist ein Dokumentationsprozess nötig. Dieser ist jedoch von vielen Faktoren abhängig und stark beeinflusst von Unternehmens- und Produktstrukturen [Kot11, vgl. Seite 47]. Der "normale" Prozess besteht laut Kothe aus den folgenden Abschnitten:

- Projektplanung
- Zielgruppenanalyse
- Recherche
- Erstellung
- Qualitätssicherung
- Übersetzung

[Kot11, Seite 48]

Zu der Projektplanung gehört das Dokumentationskonzept, welches zum Teil mit dieser Arbeit geplant wird. Zum Dokumentationskonzept gehören unter anderem Fragen wie:

- Welcher Anleitungstyp soll erstellt werden?
- In welchem System und in welchem Format soll die Anleitung erstellt werden?
- Nach welcher Struktur soll die Anleitung erstellt werden?
- Welches Abbildungskonzept soll in der Anleitung umgesetzt werden?

[Kot11, Seite 47]

Obwohl Technische Dokumentation sehr wichtig für die Entwicklung eines guten Produktes sind, wird sie in vielen Unternehmen nur als Kostenfaktor gesehen. Gerade im Vergleich zu anderen Bereichen wie Produktentwicklung oder Marketing, wird der Dokumentation wenig Beachtung geschenkt [Kot11, vgl. Seite 1f.]. "Die Dokumentationserstellung ist zumeist mit einem hohen Aufwand verbunden, wobei der eigentliche Nutzen nur schwer messbar ist."[BM07, Seite 4], daher ist es eher abschreckend für Firmen viel Kosten in diesen Prozess zu stecken. Doch gerade bei der Anwenderdokumentation ist es sehr wichtig eine gute und auch effizient geplante Dokumentation zu erstellen, denn sie dient als Kommunikationsmittel mit dem Anwender und sollte daher eher als weiteres Marketinginstrument verstanden

werden [BM07, vgl. Seite 4]. Ebenfalls wurde im vorherigen Kapitel 2.2.2 die Relevanz der Dokumentation im Zusammenhang mit der Usability des gesamten Produktes ausgeführt. Was die konkreten Eigenschaften einer Anwenderdokumentation sind wird im nächsten Unterkapitel beschrieben.

3.1.2 Anwenderdokumentationen

"Der Anwender braucht Hilfe und Anleitung für die effektive, effiziente und zufrieden stellende Nutzung interaktiver Produkte."[Bie06, Seite 62]. Dieser Aufgabe liegt die Anwenderdokumentation zugrunde. Sie ist ein Teil der externen technischen Dokumentation. Sie dient dem Kunden oder Nutzer als wichtigstes Hilfsmittel beim Erlernen der Software. Eine Anwenderdokumentation ist „[. . .] die Dokumentation, die der Endanwender tatsächlich verwendet, im wesentlichen Bedienungsanleitungen und Online-Hilfen."[Bie06, Seite 62]. Ebenfalls bei der Anwenderdokumentation gilt es grundlegende Fragen erst zu beantworten. Zum Beispiel, welche Dokumentationsart, -form und -technologie verwendet werden soll, um den Anforderungen der Anwender zu entsprechen? Online oder Papier, embedded, PDF-Handbuch, FAQ oder eine Kombination [Bie06, vgl. Seite 64]? Bei diesen Fragen seien "Anwenderbedürfnisse, -aufgaben und -umgebung, Komplexität der Inhalte, Informationsarten, Verteilung und Aktualisierung der Dokumentation, Übersetzungsbedarf, Anbindungsmöglichkeiten an die Software / an das Produkt"[Bie06, Seite 64], bedeutende Informationen, welche vorher benötigt werden.
Das "oberste Qualitätsziel" sei dabei, dass der Anwender in kürzester Zeit die passende Information finden muss [Bie06, vgl. Seite 64]. Weitere wesentliche Aufgaben einer Anwenderdokumentation sind "Lernförderlichkeit, Fehlertoleranz, [und] Selbstbeschreibungsfähigkeit" [Bie06, Seite 62].

3.2 API Grundlagen

3.2.1 Begriffserklärung

"Everything from how we order taxis, move money between bank accounts, watch entertainment, share life moments through videos on our social media feeds, and manage or monitor our homes . . . it all happens because of APIs"[AJ18, Seite 5]. Die Zahl der öffentlichen APIs ("Application Programming Interface"), welche Entwickler in ihre eigenen Anwendungen integrieren können, ist in den letzten 12 Jahren um ein Vielfaches gestiegen [AJ18, vgl. Seite 5]. Grundlegenden Prinzipien der APIs wurden bereits Mitte des 20. Jahrhunderts erkannt, seitdem wurden sie in verschiedenen Formen angewandt. Seit der Jahrtausendwende gab es zunehmend Web-APIs sowie eine große API-Industrie für E-Commerce, soziale Medien, Cloud und Mobile [Spi17, vgl. Seite 20]. Die Suche nach innovativen Lösungen, um Produkte mehrerer E-Commerce-Webseiten miteinander zu verbinden, begann. "Web-APIs auf Basis der existierenden HTTP-Infrastruktur schienen das richtige Werkzeug für diese Aufgabe zu sein"[Spi17, Seite 21]. Vorherige APIs waren eine Software, die entwickelt und implementiert wurde. Moderne APIs enthalten ein Paket an Fähigkeiten, welche für das Publikum attraktiv sind. Außerdem sind sie unabhängig von einer bestimmten Software, die in ihrem Backend

läuft [AJ18, vgl. Seite 7]. Der Vorteil der Verwendung von APIs ist eine Beschleunigung in der Entwicklung. Außerdem besteht dadurch die Möglichkeit, die Ausgereiftheit dieser Komponenten zu nutzen. Dies hat APIs zu einem Schwerpunkt der modernen Softwareentwicklung gemacht, und es sind große Software-Communitys entstanden, welche ihre frei zugängliche API für praktisch jeden Bedarf teilen [IJRJ18, vgl. S 229]. GitHub beispielsweise, eine Software-Community-Plattform, die derzeit über mehr als 200 Million Repositories hat [Git].
Der Begriff API steht für "Application Programming Interface", meist übersetzt mit "Programmierschnittstelle" oder "Anwendungsschnittstelle". Eine allgemeine Definition für APIs ist schwer zu bestimmen. Spichale beschreibt es in seinem Buch durch eine Mischung mehrerer Definitionen [Spi17, vgl. Seite 22f.]. Einerseits sei es ein Programmteil, das von einem Softwaresystem anderen Programmen zur Anbindung zur Verfügung gestellt wird. Des Weiteren würde eine API auch mögliche Interaktionen beschreiben, mit denen sie verwendet werden kann [Spi17, vgl. Seite 22]. Deswegen sei auch eine detaillierte Dokumentation zu der API sehr wichtig. Eine weitere Definition, auf die sich Spichale speziell bezieht: "Eine API spezifiziert die Operationen sowie die Ein- und Ausgaben einer Softwarekomponente. Ihr Hauptzweck besteht darin, eine Menge an Funktionen unabhängig von ihrer Implementierung zu definieren, sodass die Implementierung variieren kann, ohne die Benutzer der Softwarekomponente zu beeinträchtige"[Spi17, Seite 22f.].
APIs können in zwei Hauptkategorien unterteilt werden. Programmiersprachen-APIs und Remote API's. Erstere werden zum Beispiel von Bibliotheken angeboten und sind sprach- und plattformunabhängig. Remote API's hingegen sind beispielsweise RESTful HTTP, SOAP-Webservices oder Messaging-API's. Auch diese API's sind durch Protokolle wie HTTP sprach- und plattformunabhängig [Spi17, vgl. Seite 23].
Auch Frameworks besitzen eine API, über die sie benutzt und erweitert werden können. Dies geschieht über das sogenannte Service Provider Interface (SPI). Das SPI ist eine API, welche von einem Benutzter erweitert oder implementiert werden kann, dadurch kann eine Applikation oder ein Framework Erweiterungspunkte bereitstellen [Spi17, vgl. Seite 23f.].

APIs sind aus der heutigen Entwicklung nicht mehr wegzudenken, daher sind zurzeit API-Dokumentationen eine weit verbreitete Art der externen technischen Dokumentationen. Aus diesem Grund bestehen auch viele Studien und Forschungen zu API-Dokumentationen.

3.2.2 API-Dokumentationen

APIs müssen richtig erlernt und auch richtig bedient werden können, um effektiv zu sein. Um dieses Ziel zu erreichen, gibt es neben der API normalerweise eine Dokumentation, mit der sich Entwickler mit der API-Funktionalität vertraut machen können [IJRJ18, vgl. Seite 229f.]. Eine API-Dokumentation ist eine spezielle Anwenderdokumentation, welche wiederum eine Form der externen technischen Dokumentation ist. Die API-Dokumentation enthält Informationen in Form von Text, Code, Bildern usw., die in einem Dokument oder heutzutage meist in Form einer Webseite präsentiert werden [IJRJ18, vgl. Seite 229f.]. Studien über API-Dokumentationen, beziehungsweise was eine optimierte API-Dokumentation beinhalten muss, gibt es sehr viele. Einige davon sind in Kapitel 2.1 näher erläutert.

Laut Inzunza et al. ist es das Ziel einer API-Dokumentation, die Annahmen und Beschränkungen der API zu verdeutlichen, den Entwicklern als Leitfaden zu dienen und Fehler zu vermeiden [IJRJ18, vgl. Seite 230.].

3.3 Low-Code Grundlagen

3.3.1 Begriffserklärung

Low-Code-Plattformen sind Entwicklungsansätze, bei denen der Entwickler mit geringen Programmierkenntnissen arbeiten kann. Mit diesen Plattformen ist es möglich eine Website oder Anwendung zu erstellen, ohne eine Zeile Code zu schreiben. Dennoch bieten Low-Code-Plattformen die Möglichkeit spezielle Anpassungen vorzunehmen, bei denen Programmierkenntnisse erforderlich sind. Low-Code-Anwendungen sind gerade für KMU's (kleine und mittlere Unternehmen) eine gute Lösung, um mit der fortschreitenden Digitalisierung des Marktes mitzuhalten. Gerade KMU's haben eventuell keine richtige IT-Abteilung, die neue Anwendungen entwickeln könnten. "Diese Möglichkeit der eigenen Anpassungen und der Reaktion auf neue Herausforderungen ist ein wertvoller Helfer für die Digitalisierung im Mittelstand."[Sch20, Absatz 10.2], meint P.Schenk dazu, ein Co-Founder eines Low-Code-Plattform-Herstellers.
Low-Code-Plattformen besitzen keine richtige Entwicklungsgeschichte. Der Begriff "Low-Code" wurde erstmals 2014 in einem Bericht von Forrester[2], zum Thema "New Development Platforms Emerge For Customer-Facing Applications", erwähnt [Sch20, Absatz 3.1]. Damals ging es darum, neue Entwicklungsansätze für kundenorientierte Anwendungen vorzustellen. Erste RAD-Ansätze (Rapid Application Development) zählen auch als Grundsteine der Low-Code-Entwicklung [Sch20, vgl. Absatz 3.1].
Low-Code-Plattformen bestehen aus vier Schichten [SIDRP20, vgl. Seite 172]. Diese sind in Abbildung 3.1 veranschaulicht. Die erste Schicht, der "Application Layer", beinhaltet die grafische Oberfläche der Software, mit der ein Nutzer interagieren kann. Durch die zur Verfügung gestellten Modellierungskonstrukte dieser Schicht, können die Benutzer das Verhalten der zu entwickelnden Anwendung festlegen. Beispielsweise kann ein Nutzer festlegen, wie er Daten aus einer externen Datenquelle bearbeitet, sammelt oder analysiert. Für die weitere Bearbeitung in der Low-Code-Plattform ist die nächste Schicht verantwortlich, der "Service Integration Layer". Dieser stellt dazu eine Verbindung mit verschiedenen Diensten über entsprechende APIs und Authentifizierungsmechanismen her. Der "Data Integration Layer" befasst sich mit der Datenintegration der verschiedenen Datenquellen. Je nach Low-Code-Plattform wird die entwickelte Anwendung in speziellen Cloud-Infrastrukturen oder auf eigenen lokalen Servern bereitgestellt. Dies ist die Aufgabe der letzten Schicht, dem "Deployment Layer". [SIDRP20, vgl. Seite 172]
Low-Code-Plattformen arbeiten mit dem Prinzip WYSIWYG ("What You See Is What You Get"). Bei diesem Ansatz gestaltet der Nutzer im Vordergrund mit Hilfe grafischer Editoren, während Quellcode im Hintergrund erstellt wird. Dies dient dazu ihm ein unmittelbares visuelles Feedback zu geben. Zugleich wird der Low-Code-Nutzer nicht mit unverständlichem

[2] https://www.forrester.com/bold

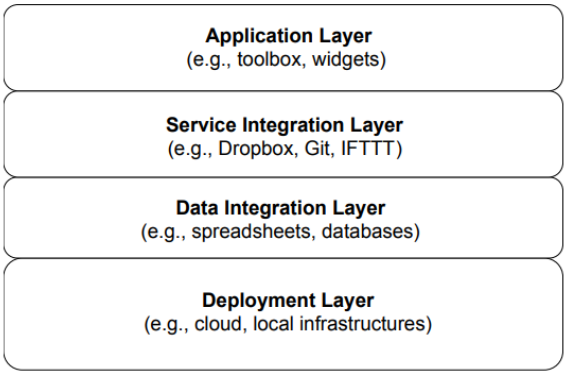

Fig. 1. Layered architecture of low-code development platforms

Abbildung 3.1: Schichten einer Low-Code-Plattform von Sahay, Apurvanand et al.,Quelle: [SIDRP20, Seite 172]

Programmiercode überfordert. " Die visuellen Editoren erstellen den Code und übernehmen ebenso die Definitionen für die Oberfläche."[Sch20, vgl. Absatz 5.1].
Low-Code-Anwendungen verfolgen die Absicht, es einem potenziellen Nutzer so einfach wie möglich zu gestalten, um eine funktionale und zugleich gut designte Software für seine Zwecke zu erstellen.

3.3.2 Unterschied zu No-Code

No-Code ist ein weiterer Ansatz, dem Entwickler ohne Programmierkenntnisse zu helfen. Wie der Name schon ausdrückt, benötigen No-Code-Plattformen überhaupt keine Programmierkenntnisse. Sie sind nach dem gleichen Prinzip wie Low-Code-Anwendungen aufgebaut, mit dem Unterschied, dass keine Möglichkeit besteht etwas zu Programmieren. Sie sind dadurch für eine breitere Masse an potenziellen Nutzern geeignet, im Vergleich zu Low-Code-Plattformen. Sie bieten eine sehr einfache Handhabung und es ist möglich eine komplette Anwendung durch simples Drag-and-drop zu erstellen. Ebenso alle Funktionalitäten sind erstellbar, ohne sie zu programmieren. No-Code sollte nicht als Weiterentwicklung von Low-Code gesehen werden, sondern als anderen Ansatz der Softwareentwicklung [Sch20, vgl. Absatz 6.2]. Auf der Kehrseite dieses Ansatzes steht jedoch die Eingeschränktheit der Anwendung. Es können lediglich Elemente eingefügt werden, welche von der No-Code-Plattform vorgegeben sind. Diese Elemente können nur mit den Möglichkeiten bearbeitet werden, welche die Plattform vorgibt. Konkrete Designwünsche können daher sehr schlecht mit No-Code-Anwendungen umgesetzt werden. Darunter kann das Branding der Marke leiden.

3.4 "yeet" Grundlagen

3.4.1 Unternehmen vectorsoft AG

In Zusammenarbeit mit dem Unternehmen vectorsoft AG wurde diese Arbeit entwickelt. Das international erfolgreiche IT-Unternehmen, mit Sitz in Heusenstamm bei Frankfurt am Main, entwickelt und vertreibt das Programmentwicklungssystem "Conzept 16". Dieses System ist seit 1985 auf dem Weltmarkt und heutzutage sind weltweit über 25.000 Serverlizenzen im Einsatz.
"Conzept 16" bietet eine integrierte Datenbank sowie einen Frontend-Designer und besitzt eine eigene, an C/C++ angelehnte Programmiersprache. Damit ist eine effiziente Anwendungsentwicklung per Rapid Application Development (RAD) einschließlich SaaS (Software as a Service) vorhanden. Conzept 16 wird als "All-in-One Software-Entwicklungssystem für Software- und Systemhäuser"[AG, Absatz 2] beschrieben.
Die Low-Code-Software "yeet" soll die nächste große Software aus dem Hause der Entwickler sein. Diese befindet sich jedoch noch in der Entwicklungsphase.
Zusätzlich besitzt die vectorsoft AG einen Standort in der Schweiz. Von beiden Standorten aus bietet die Firma Beratung und Unterstützung für ihre Kunden.

3.4.2 Was ist "yeet"?

Wie im vorherigen Abschnitt erwähnt, soll mit "yeet" eine Low-Code-Plattform entwickelt werden. Diese Anwendung soll nicht nur den Bestandskunden von Conzept 16 eine weitere performante Software bieten, "yeet" soll auch auf dem modernen Low-Code-Markt einen Platz gewinnen und neue interessierte Nutzer anziehen. Zurzeit steckt die Software jedoch noch in der Entwicklung und bis diese mit aktuellen Anbietern mithalten kann, wird noch etwas Zeit benötigt.
"yeet" ist eine klassische Low-Code-Anwendung. Wie diese grundlegend funktionieren wurde in Kapitel 3.3 beschrieben. Eine Eigenschaft von "yeet" ist die Verwendung des Vue.js[3] Frameworks in Zusammenspiel mit der JavaScript Variante Typescript[4] sowie des Frameworks Quasar[5]. Durch diese Technologien ist eine moderne und leistungsfähige Entwicklung gewährleistet. Ebenso sind diese Technologien später wichtig, in Bezug auf die Technologieauswahl der Dokumentation.
Im Sinne einer Low-Code-Anwendung lassen sich mithilfe der Software schnell und einfach Geschäftsanwendungen realisieren. Einfache Websites, wie auch interne Anwendungen für komplizierte Prozessabläufe, durch "yeet" kann beides entwickelt werden. In Abbildung 3.3 Die Low-Code-Software bietet seinen Nutzern einen grafischen Designer, mit dem die visuellen Elemente, der zu entwickelnden Anwendung, erstellt werden können. Dieser ist in Abbildung 3.2 zu sehen. Außerdem wird eine Datenbank offeriert, in welcher alle wichtigen Daten abgespeichert sind. "yeet" beinhaltet viele Komponenten, wie einen "Button" oder eine "Table". Der Nutzer kann diese verwenden und in seine Anwendung einfügen. Funktio-

[3] https://vuejs.org/
[4] https://www.typescriptlang.org/
[5] https://quasar.dev/

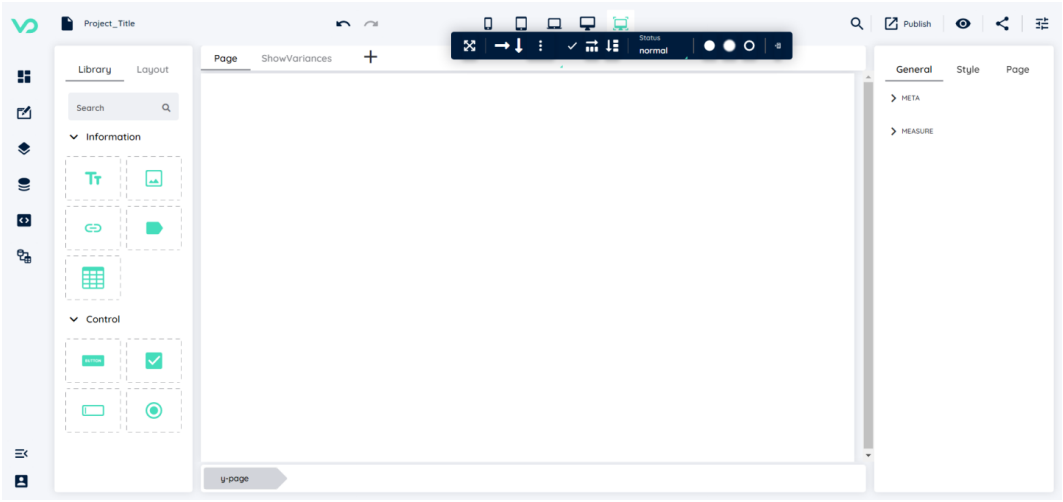

Abbildung 3.2: "yeet" Designer ,Quelle: Eigene Darstellung

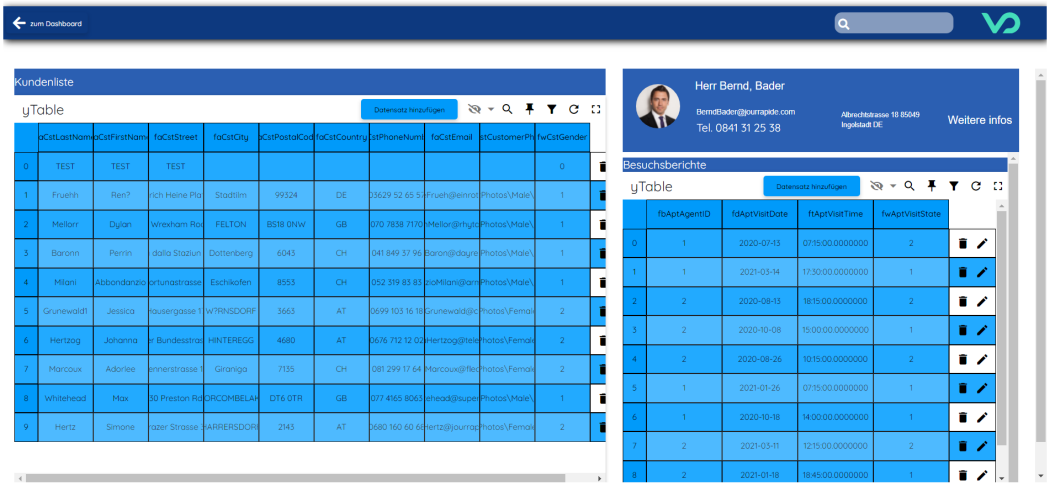

Abbildung 3.3: "yeet" Anwendungsbeispiel ,Quelle: Eigene Darstellung

nale Anpassungen dieser Komponenten - was bei anklicken des Buttons passiert - geschehen ebenso im Designer, ganz ohne Code. Durch diese vielen Anpassungsmöglichkeiten benötigen die Komponenten sowie der Designer eine gute Dokumentation. Die Low-Code-Software strebt nach dem Ziel möglichst intuitiv anwendbar zu sein, jedoch ist dies bei dieser Menge an Einstellungsmöglichkeiten nie vollständig erreichbar. Da eine gute Usability (Benutzerfreundlichkeit) im Vordergrund der Entwicklung steht, ist es wichtig eine ebenso optimierte Dokumentation bereitzustellen.

Kapitel 4

Durchführung des individuellen User-Tests

4.1 Anforderungen aus vorherigen Arbeiten

In Kapitel 2 wurden einige "verwandte Arbeiten" beschrieben und deren Ergebnisse zu dem Thema vorgestellt. Diese Arbeiten beziehen sich dabei jedoch meistens auf API- Dokumentationen und nicht auf Dokumentationen von Low-Code-Plattformen. Da die Software "yeet" der Firma vectorsoft AG jedoch eine Low-Code-Plattform ist und damit auch eine Dokumentation für eben diese Bedingungen braucht, muss erforscht werden, ob die bisherig gefundenen Anforderungen ebenfalls für Low-Code-Dokumentationen praktikabel sind.
Die verwandten Arbeiten wurden, trotz dieser Feststellung, in dem gleichnamigen Kapitel als solche präsentiert. Für den nachfolgenden Test wird vorerst angenommen, dass sich grundlegende Anforderungen beider Dokumentationsarten gleichen. Dies wurde festgelegt, damit mit den ausgearbeiteten Anforderungen dieser Arbeiten geforscht werden kann. Diese Anforderungen, sowie selbst erarbeitete Eigenschaften, wurden in dem User-Test/Interview und der Umfrage untersucht. Ob sich Anforderungen aus API-Dokumentationen mit denen der Low-Code-Dokumentationen gleichen, wird in Kapitel 6.3 erörtert.
In dieser Aufzählung werden dem Leser die Anforderungen aus Kapitel 2 nochmals gesammelt und zusammengefasst präsentiert.

- API Übersicht
- Anleitung für den Einstieg
- Beispielcode
- Video-Tuturials
- API-Referenzen dokumentieren
- API-Richtlinien dokumentieren

- Status- und Fehlercodes dokumentieren
- gut strukturiert
- leicht über Google auffindbar
- Verschiedene Niveaus
- IDE Integration
- Tutorialbereich
- Möglichkeit für Fragen, Antworten und Disskussionen
- Bewertungssystem
- Hintergrundwissen zu der API
- Negativbeispiele
- Einblick in den Sourcecode
- Szenario- und aufgabenbezogene Dokumentation
- aussagekräftige Dokumentation
- Genauigkeit, Vollständigkeit und Korrektheit
- Inhalte, die nicht das Offensichtliche wiederholen, z. B. was man von der Benutzeroberfläche lernen kann

Aus diesen Aspekten sowie selbst überlegten sinnvollen Anforderungen wurde der User-Test, sowie die Umfrage konzipiert, welche in den nächsten Abschnitten näher erklärt werden.

4.2 Definition User-Test

4.2.1 Was ist ein User-Test?

In einem User-Test oder User-Testing werden normalerweise Produkte oder Prototypen von zukünftigen Nutzern auf ihre Benutzerfreundlichkeit getestet. In einem User-Test stellt der Interviewer einer Testperson konkrete Aufgaben, welche umgesetzt werden müssen. Dies passiert meist mithilfe einer oder mehreren Softwares. Während der Teilnehmer die einzelnen Aufgaben ausführt, beobachtet der Forscher das Verhalten des Teilnehmers und hört auf sein Feedback [Mor19, vgl. Absatz 1]. Ein User-Test kann bei der Identifizierung von Problemen der Gestaltung des Produkts, dem Aufdecken von Möglichkeiten zur Verbesserung und auch zum Lernen über das Verhalten und die Vorlieben der Tester beitragen [Mor19, vgl. Absatz 3].
Eine weitere Vorgehensweise, welche oft bei User-Tests herangezogen wird, ist das "Laute Denken". Dies ist eine Methode, die verwendet wird, um kognitive Vorgänge erfassbar zu

machen, die sonst implizit und damit unausgesprochen bleiben."[Hof17, Absatz 1] Beim "Laut Denken" sprechen die Testpersonen alle Gedanken aus, welche ihnen beim bewältigen der Aufgabe in den Sinn kommen. Jede kleinste Information kann hilfreich sein. Durch diese Methode ist es einfacher Gefühle und Wahrnehmungen festzustellen. Laut Hofmann handele es sich bei den daraus resultierenden Daten um "eine Momentaufnahme von Gedanken während einer Handlung"[Hof17, Absatz 3].

4.2.2 Warum eignet sich ein User-Test für diese Arbeit?

Durch die oben genannten Eigenschaften, eignet sich der User-Test perfekt, um andere Dokumentationen von Low-Code-Plattformen zu analysieren. Die Testpersonen bekommen Aufgaben gestellt, welche sie innerhalb der Dokumentationen lösen müssen. Dabei können sehr gut die positiven oder negativen Resonanzen auf Funktionalitäten oder optische Merkmale festgestellt werden.

4.3 Definition Interview

4.3.1 Was ist ein Interview?

Bei einem Interview werden die Teilnehmer von einem Interviewer zu einem bestimmten Themengebiet befragt [Win00, vgl. Absatz 1]. Bei einem qualitativen Interview besitzt der Interviewer dazu einen Leitfaden mit thematischen Fragen, welche der Reihe nach abgearbeitet werden sollen. Der Unterschied zu einem quantitativem Interview besteht nun daran, dass die Teilnehmer auf diese Fragen frei antworten können. "Dadurch werden eine hohe Inhaltsvalidität und ein tieferer Informationsgehalt erreicht."[Win00, Absatz 1]. Bei einem quantitativen Interview gibt es zwar auch einen Leitfaden an Fragen, jedoch sind die Antwortmöglichkeiten für die Teilnehmer festgelegt. Diese können sich dann nur noch für eine Option daraus entscheiden.
Neben dem Interview wurde in dieser Arbeit ebenfalls eine Umfrage eingesetzt. Eine Umfrage ist eine Art des Interviews, bei der jedoch kein Interviewer nötig ist. Auch Umfragen können qualitativ oder quantitativ durchgeführt werden [Pfe20, vgl. Absatz 2]. In der vorliegenden Arbeit wurde die Methode der quantitativen Umfrage verwendet. Dies wurde aus dem Grund gemacht, neben den qualitativen Ergebnissen, ebenfalls quantitative zu erzielen. Ebenso wurde innerhalb der Umfrage öfters mithilfe der Likert-Skala gearbeitet. Diese ist ein Verfahren um persönliche Einstellungen zu Messen [Wü, vgl. Absatz 1]. Den Testpersonen wird dabei die Möglichkeit gegeben eine Gegebenheit nach einer Fünf-Punkte-Skala zu bewerten. In dieser Arbeit waren beispielsweise die Antwortmöglichkeiten "sehr gut", "eher gut", "mäßig", "eher schlecht" und "sehr schlecht" auszuwählen.

4.3.2 Warum eignet sich die gewählten Methoden für diese Arbeit?

Durch die Kombination des User-Tests mit dem qualitativen Interview ist es möglich viele Informationen durch einen einzigen Test zu generieren. Das Verhalten der Testpersonen so-

wie die ihre Einstellungen zu bestimmten Themen, können so gleichzeitig bestimmt werden. Wie genau dieser individuelle Test aufgebaut ist, wird im nachfolgenden Abschnitt 4.4 beschrieben.
Des Weiteren soll mit der Methode des Triangulationsdesigns gearbeitet werden. Bei dieser werden Ergebnisse qualitativer, sowie quantitativer Forschung zusammen ausgewertet. Bei diesem Ansatz der Analyse werden qualitative und quantitative Umfragen gleichzeitig erhoben und fließen gleichwertig in die Auswertung ein [NS15, vgl. Seite 7]. Speziell wird mit dem "Convergence Model" gearbeitet. Dieses Modell stellt das traditionelle Modell einer Triangulation mit gemischten Methoden dar. Das Ziel dieses Modells ist es, zu gültigen und gut begründeten Schlussfolgerungen über ein einzelnes Phänomen zu gelangen [NS15, vgl. Seite 7].
Durch Verwendung der Ergebnisse des User-Tests / Interviews zusammen mit den Ergebnissen der Umfrage, können somit in Kapitel 6 fundierte Anforderungen generiert werden.

4.4 Individueller User-Test

In den folgenden Unterkapiteln werden drei Low-Code Konkurrenzprodukte der Software "yeet" vorgestellt und deren Dokumentationen analysiert. Dies geschieht mithilfe eines selbst erarbeiteten User-Tests, welcher Teile eines qualitativen Interviews enthält. Zusätzlich zu dieser Forschungsmethode wurde ebenfalls eine quantitative empirische Methode durchgeführt in Form einer Umfrage, welche direkt nach dem Test auszufüllen war.

Aufgrund der aktuellen Pandemielage waren keine persönlichen Gespräche möglich. Der User-Test fand sowohl über das Programm Zoom und als auch mithilfe von Microsoft Teams statt. Die Bildschirm- und Audioaufnahmen wurden bei Zoom durch das dort integrierte und dafür vorgesehene Tool erzeugt. Bei Teams war es nötig, auf das externe Tool "OBS (Open Broadcaster Software)" zurückzugreifen.
Zoom ist eine Software für Videokonferenzen. Das US-amerikanische Unternehmen hinter der Anwendung heißt Zoom Video Communications, Inc. und wurde 2011 von Eric Yuan gegründet [Zooa]. Durch das kostenlose Feature "Lokale Aufzeichnung", welche den Teilnehmern die lokale Aufzeichnung von Meeting-Video und -Audio auf Ihrem Computer ermöglicht [Zoob, vlg. Absatz 1], eignet sich die Software perfekt für die Unser Tests.
Da viele Testpersonen aus der Firma vectorsoft AG kamen, wurde bei diesen Personen die Anwendung Teams von Microsoft verwendet, da diese bereits gekannt wurde. Teams wurde 2017 von der ebenfalls US-amerikanischen Firma Microsoft (MS) entwickelt [Mic]. MS Teams ist eine Plattform auf der es möglich ist Chats, Voice- und Videoanrufe, Notizen und Anhänge vereint in einem Programm zu haben. Mithilfe von Teams wurden Videoanrufe für die User-Tests ausgeführt. MS Teams bietet das Tool zum Aufnehmen von Anrufen nur als kostenpflichtige Funktion an. Daher wurde für diese Aufgabe ein anderes Programm (OBS) verwendet.
OBS, was als Abkürzung für "Open Broadcaster Software" verwendet wird, ist eine kostenlose und Open-Source Software für Videoaufnahmen und Live-Streaming[1]. Das Programm des

[1] https://obsproject.com/

Hauptentwicklers Hugh Bailey sowie vieler anderer Mitwirkenden besteht seit 2013 [OBS]. OBS wurde für das Aufnehmen der Microsoft Teams Anrufe verwendet. Damit konnten sowohl Audio- als auch Videoaufnahmen vom Bildschirm der Testpersonen erstellt werden.
Die Umfrage dauerte circa 45 Minuten und wurde durch einen Interviewer beziehungsweise Aufgabensteller begleitet. Dies wurde bewusst entschieden, da dadurch die Qualität des User-Tests erheblich steigt. In einem Artikel der Nielsen Norman Group, einer der weltweit führenden Konzerne in forschungsbasierter User Experience, heißt es, "Wenn man herausfinden will, wie Menschen ein Thema recherchieren, Angebote vergleichen und Entscheidungen treffen, ist es am besten, eine moderierte Studie durchzuführen, bei der ein Moderator anwesend ist (physisch oder aus der Ferne)."[Mor21, Absatz 3 - eigene Übersetzung]. Der Interviewer behandelte die zu stellenden Aufgaben, wie es in einem User-Test üblich ist, ohne Hilfestellungen oder Kommentare. Falls eine Aufgabe nicht geschafft wurde, mussten die Testpersonen eigenständig mitteilen, dass sie mit der nächsten Aufgabe weiter machen wollen. Bei den Fragen beziehungsweise dem Interview wurde die qualitative Interviewmethode verwendet. Es existierten Fragen, welche in einer bestimmten Reihenfolge gestellt werden sollten, jedoch war die Beantwortung dieser vollkommen frei durch die Testpersonen bestimmbar.
Die Umfrage wurde direkt im Anschluss an die User-Tests durchgeführt. Hierzu bekamen die Testpersonen einen Link zu einer Google-Umfrage gesendet und diese musste alleine beantwortet werden. Sie dauerte circa fünf Minuten. Das Gespräch wurde vorab beendet, damit der Tester frei antworten konnte.

Die drei verwendeten Low-Code-Anwendungen wurden aus dem Grund ausgewählt, da sie alle eine deutsche Herkunft besitzen oder zumindest speziell für den deutschen Markt optimiert sind. Dies wurde so gewählt, um Konkurrenzprodukte aus dem eigenem Land als besseres Vergleichsmaterial heranziehen zu können. Des Weiteren wurde bei der Auswahl darauf geachtet, dass die drei Produkte unterschiedlich umfangreich sind. Das bedeutet die erste Low-Code-Plattform hat weniger Features als die Zweite und diese wiederum besitzt weniger Einstellungsmöglichkeiten als die dritte Plattform. Der Grund hierfür liegt bei den unterschiedlichen Dokumentationen, welche daraus entstehen. Für die Untersuchung sollte herausgefunden werden, ob Dokumentationen mit weniger Inhalt, auch weniger "überladen" wirken, als Dokumentationen, welche mit Informationen überfüllt sind.

Die Fragen und Aufgaben des User-Tests wurden aus den recherchierten Eigenschaften der verwandten Arbeiten sowie selbst erarbeiteten Fragen, welche für das Thema Low-Code wissenswert sind, zusammengesetzt. In Anhang A.1 sind die Fragen des User-Tests abgebildet. Die Umfrage im Anschluss diente dazu, zu den qualitativen Ergebnissen, welche interpretativ ausgewertet werden müssen, ebenfalls quantitative Ergebnisse zu erzeugen, welche einen statistischen Einblick in die Situation hervorbringen sollen. Das Ergänzen von Fragen oder Interviews im Nachhinein ist eine gängige Methode, welche beispielsweise auch bei Völzke [Vö12, Seite 34 f.] erwähnt wird. Diese Umfrage ist im Anhang A.2 zu finden.
Der User-Test / Interview bestand aus Aufgaben, welche die Testpersonen ohne Hilfe bewältigen mussten und Fragen zu bestimmten Umständen, die jeweils beantwortet werden sollten. Zudem wurden dabei die Methode des "Lauten Denkens" verwendet.

Als Testpersonen wurden zwei Hauptgruppen ausgewählt. Die eine Gruppe bestand aus

Personen, welche einen Beruf ausführen, in dem sie programmieren müssen. Dabei war es egal, welche Art von Programmierer sie waren. Das heißt es war nebensächlich, mit welcher Programmiersprache sie arbeiten, oder ob sie als Frontend oder Backend Entwickler tätig sind. Diese Gruppe bestand aus sieben Personen. Alle Testpersonen aus dieser Gruppe, bis auf eine Person, waren Mitarbeiter der Firma vectorsoft AG. Die andere Personengruppe bestand aus Testern, welche zwar noch nie programmiert haben, jedoch mit Computern gut umzugehen wissen. Ebenfalls aus dieser Gruppe waren zwei Teilnehmer Mitarbeiter von vectorsoft AG. Die Gruppe zählte insgesamt fünf Personen. Zwölf Personen wurden in diesem Test befragt, da dies als angemessene Größe betrachtet wurde, um valide Ergebnisse zu erhalten. Zwar stellt Nielsen[NL93] in einer Studie fest, dass schon fünf Testpersonen für ein aussagekräftiges Ergebnis reichen würden, jedoch erst ab fünfzehn Testpersonen sind die Ergebnisse zu 100% exakt. Eine Gesamtmenge von 12 Testpersonen wurde festgelegt, aufgrund dieser Tatsache und dem Umstand, dass der User-Test gemeinsam mit einem Interview vollzogen wurde.

Die beiden Gruppen wurden so ausgewählt, um beide Arten möglicher Low-Code-Benutzer abzudecken. Denn wie in Kapitel 3.3 erklärt, ist eine Low-Code-Software bekannt dafür, durch ein großes Spektrum an unterschiedlichen Benutzern verwendet zu werden. Daher ist es wichtig, in dem Test unterschiedliche Individuen miteinzubeziehen. Durch diese Unterteilung konnte später eine Differenzierung zwischen den Anforderungen von "Programmierern" und "Nicht Programmierern" getroffen werden.

Der Test lief wie folgt ab:
Zuerst wurde den Testpersonen kurz erklärt, welches Thema dieser Test behandelt. Es wurde erläutert, dass dieser Test über Dokumentationen von Low-Code-Softwares absolviert wird. Für die Personen, welche nicht wussten was unter solchen Dokumentationen zu verstehen ist, wurde dies ebenfalls kurz erklärt. Mehr Informationen erhielten die Testpersonen nicht. Danach wurde abgesprochen, ob das gesamte Gespräch inklusive des Bildschirmes aufgenommen werden darf, um es später einfacher auszuwerten. Dies wurde ebenfalls vorgenommen, damit sich der Tester voll und ganz auf die Fragen und Antworten konzentrieren konnte und nicht parallel noch mitschreiben musste.
Nachdem die grundlegenden Themen besprochen waren, konnte begonnen werden. Der Test verlief dreimal nach dem selben Prinzip. Angefangen mit "Ninox", dann "Simplifier" und zum Schluss folge die Dokumentation der Software "Appian". Eine genaue Beschreibung der Produkte erfolgt in Kapitel 4.5 - 4.7. Hierbei entstand die Problematik, dass Testpersonen bestimmte Aufgaben anders behandelten, wenn sie zum zweiten Mal gestellt wurden, auch wenn es auf einer anderen Website passierte. Jedoch war dieses Verhalten leicht zu erkennen, sodass durch konkretes Nachfragen, warum nun ein anderer Weg gewählt wurde, dieser Fehler eliminiert werden konnte.

Zu Beginn wurde die Aufgabe gestellt, die Dokumentation der jeweiligen Software zu finden. Dadurch konnte herausgefunden werden, nach welchen Schlagworten Tester in einer Suchmaschine suchen. Falls die Testpersonen bei dieser Aufgabe die Dokumentation nicht über die Hauptwebsite des Anbieters fanden, wurde dies vermerkt und im Test dazu aufgefordert die Dokumentation nochmals über diesen Weg zu suchen. Dies wurde getan, um die Einstellung der Tester zu der Auffindbarkeit der Dokumentation auf der Website zu erfassen.

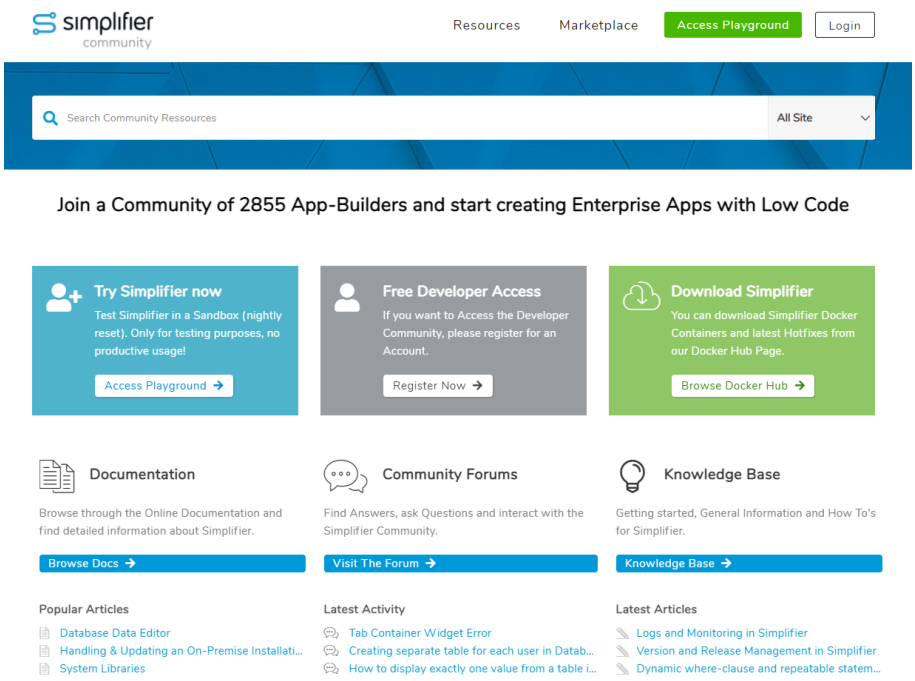

Abbildung 4.1: Beispiel einer "Hilfsstartseite" am Beispiel von `https://community.simplifier.io/`

Anschließend gab es mehrere Fragen zur Optik und Funktionalität der Startseite der Hilfeseite, um herauszufinden welche Merkmale den Testpersonen wichtig und erwähnenswert erscheinen. Diese Startseite gehört nicht explizit zu einer Dokumentation, jedoch waren die Erkenntnisse daraus äußerst hilfreich. Ein Beispiel einer solche "Hilfestartseite" ist in Abbildung **??** zu sehen.

Nach diesen allgemeinen Fragen wurden spezifischere Fragen zu den Kategorien der Startseite gestellt. Die Fragen waren wie folgt aufgebaut: "Wenn Sie die Software X installieren wollen würden, bei welcher Kategorie würden Sie zuerst nachschauen?". Die daraus gewonnenen Einsichten waren hilfreich um nachzuvollziehen, welche Informationen in einer Dokumentation erwartet werden und welche eher bei anderen Hilfsangeboten, wie zum Beispiel dem Forum, gesucht werden.

Die Testpersonen durften nun die eigentliche Dokumentation der jeweiligen Software öffnen. Sie bekamen die Aufgabe, etwas in der Dokumentation zu finden. Dabei wurden alle Eindrücke, welche durch das "Laute Denken" vernommen wurden, gesammelt und aufgeschrieben.

Wenn die Aufgabe erfüllt wurde, waren die Tester auf einer bestimmten Seite in der Dokumentation angelangt. Falls sie diese nicht gefunden haben, wurde ihnen der Link dorthin geschickt. Auf dieser Seite wurden erneut Fragen zum optischen Gesamteindruck gestellt. Natürlich konnten die Testpersonen alle anderen Gedanken, zum Beispiel zu bestimmten Funktionalitäten, ebenfalls nennen. Ziel dieser Fragen war es, einen Eindruck zu erhalten,

wie ein einzelner Dokumentationsartikel aufgebaut sein sollte. Je nach Anwendung wurden zusätzlich noch konkrete Fragen zu bestimmten Merkmalen gestellt, welche nur die jeweilige Anwendung vorweisen konnte. Zum Beispiel wurde bei der Dokumentation von Ninox nach dem Nutzen des angebotenen PDF-Downloads gefragt - ob dieser von den Testpersonen gebraucht werden würde oder ob sie auf ihn verzichten könnten. Als anderes Beispiel dient ebenso eine konkrete Frage, welche in Bezug auf Simplifier gestellt wurde. Bei dieser mussten die Testpersonen ihre persönliche Einstellung zu den Social Media Angeboten im Zusammenhang mit der Dokumentation darlegen.
Nach diesen Eindrücken sollten die Tester eine bestimmte Seite in der Dokumentation aufrufen, in welcher die Hervorhebung eines Codebeispiels zu sehen war. Ebenfalls zu diesem Element sollte der optische Eindruck beschrieben werden und positive oder negative Funktionalitäten erläutert werden. Durch diese Frage wurde deutlich, welche Einflüsse das Design auf die Beispiele von Codeabschnitten hat und auch, welche Funktionalitäten gewollt sind beziehungsweise welche vermieden werden sollten.
Die vorletzte Frage, handelte speziell von der Navigation der jeweiligen Anwendung. Somit konnten ebenfalls Eindrücke zu dieser gesammelt werden, welche andernfalls übersehen werden könnten.
Schlussendlich bekamen die Testpersonen die Aufgabe die Seite der "Frequently Asked Questions"(FAQ) zu finden und Ihre Meinung dazu zu äußern. Dadurch konnte festgestellt werden, wo ein FAQ erwartet wird, als Bestandteil einer Dokumentation oder als eigenständiger Abschnitt.

Direkt nach dem Test mussten die Testpersonen noch die Umfrage absolvieren, damit die Erinnerungen daran präsent blieben. Die Umfrage war ähnlich des User-Tests aufgebaut. Sie enthielt zu Beginn allgemeine Fragen zur Person, damit diese später besser einzuordnen waren. Anschließend wurden allgemeine Fragen über die drei Testseiten gestellt. Diese nützten dem Zweck sowohl generelle Eigenschaften, als auch statistische Werte zu erlangen. Die letzte Seite der Umfrage befasste sich mit allen Fragen, welche nicht direkt in den User-Test passten, jedoch als wichtig erschienen. Beispielsweise die Frage, ob eine Dokumentation nach Schwierigkeitsstufen einstellbar sein sollte.

4.5 Vorstellung Ninox

Ninox ist die erste Anwendung, welche für den User-Test ausgewählt wurde. Sie wurde von der gleichnamigen deutschen Firma Ninox Software GmbH entwickelt, deren Hauptsitz in Berlin ist. Sie ist eine Low-Code-Software, welche sich jedoch hauptsächlich mit Datenbanken auseinandersetzt. Das bedeutet, mit ihrer Hilfe ist es den Kunden möglich in sehr kurzer Zeit übersichtliche und funktionale Lösungen zu erschaffen, welche interne Geschäftsprozesse digitalisieren und vereinfachen sollen. Auf ihrer Startseite werben sie damit, Formulare, Berichte und Auswertungen einfach über Drag&Drop erstellen zu können, mit mehreren Mitarbeitern an einem Projekt gleichzeitig zu arbeiten und mit einer großen Vielzahl an kompatiblen Geräten verwendbar zu sein [Nin, vgl. Startseite]. Letzteres bezieht sich darauf, dass die Ninox Plattform auf jedem Gerät verfügbar sein soll, egal ob als App für Android oder IOS oder als Anwendung für Windows PC's oder Apples Mac's. [Nin, vgl. Plattformen].

Neben den Punkten, dass Ninox eine kleinere, deutsche Low-Code-Anwendung ist, wurde sie außerdem wegen der ordentlichen, klaren Dokumentation und dem modernen Gesamtbild der Website ausgewählt.

4.6 Vorstellung Simplifier

Simplifier ist ebenfalls eine Low-Code-Plattform, mit Firmenhauptsitz in Würzburg, Deutschland. Die Simplifier AG wurde 2019, aus einer Umfirmierung der Firma iTiZZiMO gegründet. Ihr Schwerpunkt liegt, laut eigenen Angaben, auf der "konfigurativen Erstellung (Low-Code) von integrierten Unternehmensanwendungen, die auf modernen und innovativen (Web)-Technologien basieren." [Sim, Absatz 1]. Simplifier wirbt damit, ihr Produkt einfach in bestehende Systemlandschaften, Maschinen und Anlagen integrieren zu können. Zudem sollen die erstellten Applikationen geräte- und betriebssystemunabhängig funktionieren und ohne großen Programmieraufwand entwickelbar sein [Sim, vgl. Absatz 1]. Simplifier ist ein Hauptkonkurrent der Low-Code-Software "yeet" und daher perfekt geeignet, um in dem Test analysiert zu werden. Die Anwendung besitzt eine klassische Dokumentation und eine ausführliche Hilfeseite, auf der alle angebotenen Hilfeleistungen von Simplifier zu finden sind.

4.7 Vorstellung Appian

Die Low-Code-Automatisierungsplattform Appian ist die umfangreichste der drei Applikationen. Ebenfalls ist es das größte Unternehmen, denn der deutsche Markt ist nur einer von vielen, auf dem sie tätig sind [App]. Appian Corporation, welche die Low-Code-Anwendung 2017 hervor brachte, ist ein US-amerikanisches Software-Unternehmen [Kon19]. Somit passt die Anwendung eigentlich nicht in das Auswahlkriterium, dass die zu analysierenden Softwares ebenfalls einen deutschen Ursprung haben sollen. Jedoch hat die Appian Corporation ein Tochterunternehmen in Deutschland, welches sich um die Wünsche des deutschen Marktes kümmert. Beispielsweise ist die Internetseite von Appian dem jeweiligen Land angepasst [App]. Aufgrund der vorgenannten Gründe wurde entschieden, dass Appian trotzdem als geeignet für den Test gilt, da die Analyse der Dokumentation sehr wichtige Erkenntnisse liefern könnte. Appian Corporation ist einer der 'Vorreiter', wenn es um Low-Code-Softwares geht. 2017 waren sie das erste Unternehmen, welches als reiner Low-Code-Anbieter an die Börse ging [BK21, vgl. Seite 8]. Seitdem Appian erschien, war die Anwendung bei dem Forrester Wave Ranking immer einer der "Strong Performers" [BK21, vgl. Seite 3 + 8].
Forrester Research, Inc. ist ein weltweit tätiges, unabhängiges Forschungs- und Beratungsunternehmen." [For, Absatz 1, eigene Übersetzung]. Diese Firma bietet regelmäßige Analysen zu den unterschiedlichsten Marktsegmenten und Anbietern, darunter auch zu Low-Code-Anwendungen. Dabei werden die Anwendungen immer in vier Kategorien eingeteilt, "Leader", "Strong Performer", "Contender" und "Challenger". Laut Forrester, sind "Leader" die zurzeit besten Anwendungen und "Challenger" die schwächsten in dem Segment [BK21, vgl. Seite 2].

Kapitel 5

Analyse der Tests

5.1 Vorgehensweise

Bei den Erhebungen der Daten des vorherigen Kapitels, wurde eine Mischform der Umfrage verwendet. Zum einen war es ein gängiger User-Test, bei dem der Testperson Aufgaben gestellt wurden, welche diese ohne fremde Hilfe lösen musste und dabei eigene Gedanken laut aussprechen sollte. Zum anderen wurde nebenbei ein qualitatives, semistrukturiertes Interview durchgeführt. Das heißt, den Testern wurden an bestimmten Punkten Fragen gestellt, welche diese frei beantworten konnten. Zum Ende gab es noch eine andere Art der Umfrage, nämlich einen Umfragebogen mit vorgegebenen Fragen und Antworten. Dieser soll jedoch getrennt von den anderen Umfragen analysiert werden, da er ein statistisches Ergebnis hervorbringt. Die gesamte Auswertung am Ende soll nach dem Triangulationsdesign geschehen und alle Ergebnisse werden zusammen ausgewertet.
Folglich behandelt dieses Kapitel die Analyse der verschiedenen Umfragen. Die Daten des User-Tests und des Interviews werden dabei zusammen ausgewertet. Dies geschieht zu Teilen nach der Methode der qualitativen Inhaltsanalyse von Kuckartz [Kuc18]. Die Vorgehensweise ist wie folgt: alle Daten aus dem User-Test und dem Interview wurden stichpunktartig in der Excel-Tabelle "Alle Aussagen User-Test" festgehalten. Diese ist in Anhang A.3 zu finden. Dabei wurden zu den verschiedenen Aufgaben Schlagworte erfasst, welche entweder von der Testperson zu der jeweiligen Aufgabe durch "Lautes Denken" geäußert wurden oder durch Verhalten, welches die Testperson hervorbrachte. Des Weiteren wurden zu den Fragen, welche die Tester beantworten sollten, ebenfalls Schlagworte aufgeschrieben. Lediglich sehr interessante Aussagen wurden wortwörtlich festgehalten und entsprechend in der Tabelle markiert. Die Erstellung der Tabelle erfolgte durch die erneute Auswertung der Videos, welche für diesen Zweck aufgenommen wurden.
Danach wurden erste systematische Gruppierungen vorgenommen. Bei den Aufgaben des User-Testes wurden gleiche Antworten oder Verhaltensweisen markiert und festgehalten. Die Ergebnisse der Fragen wurden in positive, negative und neutrale Antworten aufgegliedert. Zum Schluss wurden die Testpersonen nach "Programmierer" und "Nicht-Programmierer" eingeteilt.
Im letzten Schritt wird der Umfragebogen analysiert. Dieser wurde mit dem Tool "Google

Formulare" kreiert. "Google Formulare" ist eine kostenlose Anwendung, des Unternehmens Google Inc., mit dessen Hilfe sich unter anderem Umfragen erstellen lassen [Goo].
In den nachfolgenden Kapiteln wird der Begriff "User-Test" als Verkürzung für "User-Test / Interview" verwendet.

5.2 User-Test Analyse

Aus der Excel-Tabelle "Alle Aussagen User-Test", welche in Anhang A.3 zu finden ist, wurden die Ergebnisse herausgefiltert und in einer weiteren Tabelle "Anforderungen aus User-Test" gesammelt. Diese ist ebenfalls in Anhang A.3 zu finden. Um die Übersicht zu wahren, werden die Gruppen, in welche die Testpersonen vorher eingeteilt wurden, im nachfolgenden Abschnitt mit G1 und G2 bezeichnet. Dabei steht G1 für die Gruppe der "Nicht-Programmierer" und G2 für die der "Programmierer". Diese Einteilung wurde durchgeführt, um einen besseren Überblick über eventuelle Unterschiede in den Antworten der jeweiligen Gruppen zueinander zu haben. Des Weiteren wurden Zitate, welche von einzelnen Testpersonen stammen, mit beispielsweise "T4" gekennzeichnet. Die Zahl hinter dem "T" steht hierbei für die Nummer, welcher der jeweiligen Testperson in der Tabelle zugeordnet wurde. Alle Antworten der Tabelle "Alle Aussagen User-Test" wurden analysiert. Sobald in einer Gruppe eine Antwort mehr als zweimal vorkam, wurde sie in der Tabelle "Anforderungen aus User-Test" (ebenfalls zu finden unter Anhang A.3) gesammelt. Danach wurde diese Tabelle geprüft. Letztendlich wurde festgelegt, dass alle Antworten, welche sich in dieser Tabelle in beiden Gruppen glichen, als "must-have"-Anforderungen gelten. Alle anderen Antworten, die gesammelt wurden, wurden als "nice-to-have"-Anforderung angesehen. Daneben gibt es auch Kategorien, welche sich nicht eindeutig in Anforderungen gliedern lassen, jedoch wird bei diesen im folgenden Abschnitt immer erklärt, wie damit verfahren wurde.
Die erste Aufgabe, die Dokumentation zu finden, vollbrachten 75% der Testpersonen aus G1, indem sie sofort "xxx Dokumentation" oder "xxx Doku" googelten (wobei "xxx" stellvertretend für den jeweiligen Namen der Dokumentation steht). In G2 wandten zumindest 53% diese Methode an. Dabei fiel auf, dass es bei der ersten und zweiten Plattform Probleme mit den Suchergebnissen gab. Denn bei Ninox war es für zwei Tester verwirrend, dass die Dokumentation bei Google als "Handbuch" gelistet ist. Auf deren Website heißt es dann entweder "Dokumentation" oder "Benutzerhandbuch". Hier wurde sich eine einheitliche Benennung gewünscht. Bei Simplifier lag das Problem daran, dass über Google die falschen Websites angezeigt wurden. Erst wenn nach der Dokumentation auf Englisch gesucht wurde, wurde die Community Seite aufgelistet, auf der die Dokumentation gefunden wurde. Acht Testpersonen haben insgesamt bei der Google Suche auf den falschen Link geklickt, zwei Personen bei Ninox und sechs bei Simplifier. Drei Tester aus G2 gaben an, dass sie gerade bei Anwendungen, welche sie noch nicht kennen, lieber erst einmal über deren Website gehen, anstatt gleich zu googeln.

Die Testpersonen, welche bei der vorherigen Aufgabe die Dokumentation nicht über die jeweilige Website gesucht haben, sondern gleich "xxx Dokumentation" gegoogelt haben, wurden aufgefordert nochmals über die Website zu suchen. Fast alle Testpersonen (100% G1, 86% G2) suchten die Dokumentation erst in der Headernavigation der Website und

schauten dann in den Infos des Footers danach. Auffällig war, dass nur 44% aus G1 die Dokumentation im Header gefunden haben, 11% fanden sie im Footer und ebenfalls 44% fanden sie gar nicht. Aus G2 fanden sie zumindest 57% im Header, 5% im Footer und 38% fanden sie nicht auf der Website. Ninox war die einzige der drei Websites, welche die Dokumentation als eigenen Punkt unter dem Navigationspunkt "Hilfe" im Header aufgelistet hatte. Dadurch war Ninox auch die einzige Plattform, auf der die Dokumentation sofort gefunden wurde, zehn von elf Testern fanden sie dort schnell. Bei den anderen Anwendungen war die Dokumentation immer etwas "versteckter". Sie waren bei beiden unter dem Punkt "Community" zu finden, welcher jeweils auf eine andere Seite führte. Dort war die Hilfsstartseite, mit allen angebotenen Hilfeleistungen zu finden, wie zum Beispiel dem Forum, Blog oder der Dokumentation. Diese Strukturierung fanden sieben Testpersonen jedoch ziemlich verwunderlich. Dazu kam von T4 die Aussage "[Unter] Community, das ist komisch. Dokumentation hat nichts mit Community zu tun. Community ist Forum, Blog, etc. Eine Dokumentation hat einen direkten Produktbezug, ich hätte nie unter Community nach der Doku gesucht.". Auch andere Testpersonen fanden, dass die Dokumentation offensichtlicher auf der Produktwebsite eingebaut werden soll, "Dass die Doku immer so versteckt ist, ist unverständlich." [T11].

Als nächste Aufgabe war eine Frage zu beantworten, welche sich auf den Eindruck der Hilfsstartseite bezog. Die Hilfsstartseite ist in diesem Bezug die Seite, auf der die jeweilige Anwendung all ihre Hilfsangebote auflistet. Da alle drei Applikationen eine solche Seite besaßen, wurde dies in den Test miteinbezogen. Hier wurden in der Tabelle viele Adjektive gesammelt, welche die Testpersonen äußerten sowie generelle Anmerkungen zu verschiedenen Funktionalitäten der Startseite. Die Adjektive und Funktionalitäten, welche sich bei beiden Gruppen glichen, waren folgende:

- übersichtlich, simpel, aufgeräumt, klar aufgebaut, sauber
- Benennung muss immer gleich sein (bei Google, im Navigationsmenü, auf der Startseite, im Breadcrumb-Menü, etc.)
- Begrifflichkeiten müssen klar gewählt werden
- Suche ist wichtig - muss gut sichtbar sein
- Einstiegspunkte sind sinnvoll
- Startseite darf Benutzer nicht "erschlagen", darf nicht zu voll sein
- farblich und inhaltlich nicht überladen
- auf den ersten Blick alle wichtigen Infos enthalten - jedoch keine unnötigen
- Inhalte sinnvoll abtrennen

Bei einem Punkt trennten sich die Meinungen der beiden Gruppen. G1 war der Meinung, dass Animierungen oder Teaser zum Testen der Anwendung "nice-to-have" seien, jedoch nicht die Hauptaufmerksamkeit bekommen sollten. Tester der Gruppe G2 waren der Meinung,

dass jegliche Teaser eher störend seien und als uninteressante Werbung lieber weggelassen werden sollen.
Die nachfolgenden Punkte sind Themen, welche ausschließlich bei Gruppe G1 vorkamen:

- keine unnötigen/langweiligen Icons verwenden
- nicht zu viel Freiraum lassen
- Informationen wie "What's New", etc. sollten nicht die Hauptaufmerksamkeit bekommen und demnach nicht ganz oben stehen
- Live-Chat wäre "nice-to-have"

Nun folgen Punkte, welche ausschließlich bei Gruppe G2 vorkamen:

- Dokumentation sollte eigene Website haben
- Icons/Bilder sollten Inhalte der Kategorien verdeutlichen

Die nächste Aufgabe war etwas komplizierter. Den Testpersonen wurden zu jeder Hilfestartseite Fragen gestellt, wie zum Beispiel "Wenn Sie die Software installieren wollen würden, bei welcher der xx Kategorien würden Sie danach suchen?". Dabei mussten die Tester lediglich sagen, wie Sie vorgehen würden und dies nicht weiter ausführen. Die "xx" stehen hier beispielsweise für die Anzahl der Kategorien der jeweiligen Seite. Als Kategorien wurden hierbei die Suche, das Forum, der Blog, die Dokumentation, etc. gezählt, folglich alle Hilfsangebote, welche die Seite bereitstellte.
Zur Auswertung der Daten wurde in den jeweiligen Gruppen geprüft, ob es Übereinstimmungen bei den Antworten gab. Außerdem wurden immer die ersten zwei Antworten berücksichtigt, falls es mehr als eine gab. Die Prozentangabe wurde dann aus den Gesamtantworten der Gruppe ausgerechnet. Das heißt eine Gruppe mit nur sieben Teilnehmern, konnte jedoch 14 Antworten haben. Es wurde also zur Auswertung nicht die Teilnehmerzahl, sondern die Anzahl an Antworten genutzt. Außerdem wurden fünf Hauptantworten ausgewählt, auf die das Hauptaugenmerk gelegt wurde. Diese wären Dokumentation, Suche benutzen, Forum, "Knowledge Base o. Tutorial Videos o.ä." und Google. Alles andere wurde als Kategorie "Anderes" gewertet.
Um die Anwendung zu installieren, würden 55% der Tester aus G1 unter "Anderes" suchen, jeweils 18% würden die Suche benutzen oder bei der Kategorie "Knowledge Base o. Tutorial Videos o.ä." schauen. 9% würden googeln, jedoch kein Einziger der Gruppe G1 würde die Dokumentation benutzen. Bei der Gruppe G2 ist die Dokumentation die meistgewählte Kategorie mit 50%. Danach folgt "Anderes" mit 23%, "Knowledge Base o. Tutorial Videos o.ä." mit 15% und zum Schluss würden 8% die Suche benutzen. Das Forum würde keiner der Testpersonen zu dieser Frage benutzen.
Die nächste Frage ging um "Einsteiger-Content". Das heißt die Tester sollten Informationen zu ersten Schritten mit der Anwendung finden. Bei der Gruppe G1 war die Kategorie "Anderes" mit 57% wieder die größte Anlaufstelle. 33% würden jedoch bei "Knowledge Base o. Tutorial Videos o.ä." schauen. Jeweils 11% würde in der Dokumentation suchen oder die Suche benutzen. Keiner würde das Forum oder Google verwenden. Mit 33% liegt bei G2

"Knowledge Base o. Tutorial Videos o.ä." vorne. Danach würden die Testpersonen dieser Gruppe die Dokumentation selbst benutzen (30%), bei der Kategorie "Anderes" schauen (19%), die Suche benutzen (15%) und wenige (3%) würden das Forum konsultieren. Bei diesem Problem würde keiner der Tester Google benutzen.
Nun hatten die Testpersonen einen spezifischen Suchauftrag, und zwar eine Erklärung dazu finden, wie ein "Button" in die jeweilige Anwendung eingefügt werden kann. Aus Gruppe G1 würden 33% dazu das Forum befragen. Jeweils 22% würden die Dokumentation oder Suchen verwenden und jeweils 11% würden bei diesem Problem Google oder "Anderes" benutzen. "Knowledge Base o. Tutorial Videos o.ä." würde keine der Testpersonen gebrauchen. Bei Gruppe G2 verhält es sich etwas anders. Hier würden die Meisten (39%) die Dokumentation verwenden. Danach würden 25% bei "Anderes" nachschauen und 21% würden die Suche benutzen. 7% würden im Forum suchen und jeweils 4% würden bei "Knowledge Base o. Tutorial Videos o.ä." schauen.
Die letzte Frage handelte um einen Fehler oder Fehlercode, welchen die Testpersonen zu lösen versuchen sollten. 46% der Tester aus G1 würden dazu im Forum schauen. Je 18% würden die Suche benutzen, bei "Knowledge Base o. Tutorial Videos o.ä." suchen oder Google befragen. Kein Tester würde bei dieser Frage die Kategorie "Anderes" gebrauchen. Auch bei G2 würden die Meisten das Forum benutzen, und zwar 33%. Mit 26% war die zweit häufigste Antwort zu dieser Frage "die Suche benutzen". Danach würden die Tester in die Dokumentation schauen, Google benutzen oder bei "Anderes" schauen, mit jeweils 11%. Zuletzt würden 7% bei "Knowledge Base o. Tutorial Videos o.ä." suchen.

Nun folgten wieder mehrere Aufgaben, bei denen eine konkrete Frage im Vordergrund stand. Angefangen mit einer Frage, welche sich auf einen speziellen Dokumentationseintrag bezog. Auch hier wurden in der Tabelle wieder viele Adjektive notiert, welche die Testpersonen äußerten sowie generelle Anmerkungen zu verschiedenen Funktionalitäten des Dokumentationsartikels. Die Adjektive und Funktionalitäten, welche sich bei beiden Gruppen glichen, waren folgende:

- interaktive Componenten
- gutes Bild/Textverhältnis
- grundsätzliche Eigenschaften am Anfang / Anfangsbeschreibung
- verlinkte Schlagworte auf andere Dokumentationsartikel
- gute, große, hervorgehobene Bilder
- Seitenaufbau muss gleiche Struktur haben

Punkte, welche nur bei G1 aufkamen:

- Übersichtlichkeit
- gute Textgröße

Bei dieser Frage hatten die Testpersonen aus G2 deutlich mehr Anmerkungen:

- viele, reale Beispiele
- alle möglichen Informationen zu dem Thema aufzeigen
- ausführliche, klare Überschriften
- lieber gleich große Bilder, Funktion zum Bilder vergrößern eher störend
- nicht überladen
- nicht durch Navigation "eingeengt"

Die nächste Frage bezog sich auf die Hilfen am Ende der jeweiligen Artikel. Des Weiteren wurden drei Fragen gestellt, welche sich immer nur auf Features der aktuellen Dokumentation bezogen. Beispielsweise die erste Low-Code Plattform bot in ihrer Dokumentation unter jedem Artikel die Möglichkeit die gesamte Dokumentation als PDF herunterzuladen an. Dort wurden die Testpersonen gefragt, unter welchen Umständen sie so etwas nutzen würden oder ob sie solche Optionen überhaupt nutzen würden. Die Meinungen zu den unteren Hilfen, welche sich bei beiden Gruppen glichen, waren wie folgt:

- bei Forum Hilfen sowie bei "Related Artikels" ist es wichtig, dass sinnvolle Artikel angezeigt werden
- Hilfen dürfen nicht zu vollgepackt sein, Überschriften reichen

Teilnehmer der Gruppe G1 wünschten sich darüber hinaus noch:

- Kommentarfeld für Artikel

Ansonsten gab es keine Anmerkungen zu dieser Frage. Im Anschluss wurden die konkreten Fragen gestellt.
Ob Teilnehmer Social Media Angebote nutzen würden, welche am Ende von Simplifiers Dokumenationsartikel zu finden sind, beantworteten alle Teilnehmer aus beiden Gruppen mit "nein".
Simplifier bot des Weiteren eine Funktion an, mit welcher sich der Artikel bewerten ließ. Die aktuellen positiven Bewertungen wurden dabei am Anfang der Seite angezeigt. Die gesamten negativen, sowie positiven Bewertungen fanden sich nochmals auf den Bewertungs-Buttons selbst wieder. Ebenfalls hier war das Ergebnis eindeutig. Für die meisten Teilnehmer war so eine Bewertungsoption unnötig. Lediglich ein Teilnehmer aus G1 teilte mit, dass er so etwas benutzen würde und auch drauf achten würde, "wenn schon viele es schlecht bewerten, muss man es sich nicht durchlesen"[T3].
Die letzte Frage drehte sich um das Angebot von Ninox, sich die ganze Dokumentation als PDF herunterladen zu können. Drei von vier Teilnehmern aus Gruppe G1 fanden dieses Angebot nützlich und würden es wahrnehmen und auch vier von sieben Teilnehmern der Gruppe G2 waren derselben Meinung. Mehr als die Hälfte aus den jeweiligen Gruppen finden eine solche Option nützlich. Auf die Folgefrage, warum sie es nützlich finden, wurden Antworten wie, "falls die Website irgendwann down sein könnte"[T4 + T5], "[...] nützlich, gerade wenn

man ohne Internet unterwegs ist" [T10, T6, T9] oder "höchstens für STRG+F Suche"[T11, T12]. Eine Testperson (T4) merkte bei dieser Frage an, dass die Platzierung dieses Buttons jedoch "unlogisch" sei. Da die gesamte Dokumentation heruntergeladen wird, sollte dieser lieber am Anfang des Artikels auftauchen oder generell am Anfang der Dokumentation und nicht erst am Ende eines bestimmten Eintrags.

Die nächste Aufgabe bezog sich auf Beispiele, in denen Codezeilen gezeigt wurden. Dafür wurde den Testpersonen ein Link geschickt, welcher zu einem Artikel der jeweiligen Dokumentation führte, der solch ein Beispiel abbildete. Auf dieser Seite wurde nun wieder die Frage gestellt, ob es Anmerkungen zu dem gezeigten Beispiel gibt. Die Adjektive und Funktionalitäten wurden abermals als Stichpunkte in der Tabelle gesammelt und sortiert. Die einzige Aussage, welche in beiden Gruppen häufiger aufkam, war folgende:

- farbliches Highlighting ist wichtig

Bei den nachfolgenden Aussagen gab es keine Übereinstimmungen der beiden Gruppen mehr. Teilnehmer aus G1 fanden folgende Punkte nennenswert:

- gut abgetrennt, gut erkenntlich
- Erklärungen wichtig
- platzsparend
- extra Erweiterung für Codebeispiel
- übersichtlich
- gut verständlich

Anmerkungen von Teilnehmern aus G2 waren:

- Kommentare im Code selbst
- Formatierung wichtig
- gleiche Struktur wichtig
- gute Überschriften
- rauskopierbar
- interaktiver Code eher nice-to-have

Nach dieser Frage wurde die Aufgabe gestellt, sich die jeweilige Navigation der Seite nochmals genauer anzuschauen und auch zu dieser Anmerkungen zu tätigen. Adjektive und Funktionalitäten, welche bei beiden Teilnehmergruppen mehrfach genannt wurden, sind folgende:

- Hauptpunkte brauchen klare Namen

- Verschachtelungen mit nur einem Unterpunkt sind unnötig
- nicht zu viel auf einmal "aufklappen" können
- Navigation darf Seite nicht "einengen"/nicht zu viel Platz wegnehmen

Aus der Gruppe G1 gab es zu diesem Thema noch Anmerkungen wie folgt:

- Überpunkte sollten auch anklickbar sein
- Punkte sollten nicht zu gleichwertig sein
- Breadcrumb-Navigation o. andere Option zum zurückgehen wichtig

Aus Gruppe G2 gab es nur eine weitere Anmerkung:

- nicht zu viele Unterpunkte/Verschachtelungen

Bei dieser Aufgabe wurde am Ende eine Frage gestellt, auf die alle Teilnehmer eindeutig mit ja oder nein antworten sollten. Die Frage bezog sich auf die letzte Dokumentation von Appian und lautete folgendermaßen: "Die Navigation aufgeteilt in die Hauptnavigation links, sowie die Seitennavigation rechts zu haben, ist das praktisch und gut oder störend und schlecht?". Alle Testpersonen aus G1 empfanden diesen Ansatz als praktisch und gut. "Die rechte Navigation auf der Seite ist sehr gut, weil man gleich zu den wichtigen Punkten hinspringen kann"[T2]. Aus Gruppe G2 fanden nur zwei Personen die Navigation in dieser Form wirklich gut. Alle anderen empfanden es eher als umständlich, oft von links nach rechts schauen zu müssen [T4, T5, T7]. Auch wurde angemerkt, dass beide Navigationen oft zu viel sind und viel Platz wegnehmen [T2, T5, T12]. Gerade unter dem Gesichtspunkt, dass manchmal sehr wenige Unterpunkte vorhanden sind und es so den Testern als noch unnötiger auffiel "Bei so wenigen Punkten, warum ist nicht direkt alles links?"[T5].

Die letzte Aufgabe bezog sich auf das FAQ der jeweiligen Seite. Die Testpersonen wurden gebeten dieses zu finden. Als Hilfestellung hatten sie einen Begriff, welcher sich im FAQ wiederfand. Dennoch war es für viele Teilnehmer nicht leicht das FAQ zu finden. Es stellte sich heraus, dass die Meisten eine eigene Seite für das FAQ gesucht haben. Bei Ninox war dieses beispielsweise der letzte Punkt der Navigation der Dokumentation. An dieser Stelle hat es allerdings kaum einer vermutet. Nur eine Person (T4) meinte, dass das FAQ Bestandteil des Menüs in der Dokumentation sei [T4]. Der Rest (acht Testpersonen aus beiden Gruppen) war der Meinung, dass ein FAQ eine eigene Seite haben sollte und diese dann auf der Hilfsstartseite oder im Footer verlinkt sein sollte. "FAQ's suche ich immer unten"[T5]. Alle Adjektive und Funktionalitäten, welche die Teilnehmer beim Finden des FAQ's sonst noch geäußert haben, wurden in einer Tabelle gesammelt. Die sich in beiden Gruppen überschneidenden Punkte sind:

- FAQ braucht eigene Seite, z.B. unter der Hilfestartseite
- sollte im Footer zu finden sein

Teilnehmer aus Gruppe G2 hatten noch einen zusätzlichen Punkt:

- wenn in Suche "FAQ" gesucht wird, sollte dieser Überpunkt auch angezeigt werden und nicht nur Inhalte dessen

5.3 Umfrage Analyse

Die Ergebnisse des Umfragebogens wurden statistisch ausgewertet und werden in diesem Kapitel näher erläutert. Der Umfragebogen dient dazu, zusätzlich zu den qualitativen Aussagen des User-Tests und des Interviews, quantitative Ergebnisse zu erhalten, welche statistisch ausgewertet werden können. Dadurch, dass Resultate beider Varianten vorliegen, wird es später möglich sein, die Ergebnisse durch das Triangulationsdesign, welches in Kapitel 4.3.2 bereits erläutert wurde, zusammenzufassen und so eine aussagekräftigere Analyse zu bieten. Viele der Fragen konvergieren daher absichtlich mit den der zuvor gestellten Aufgaben.
Der Umfragebogen wurde von den Testpersonen direkt nach dem User-Test bearbeitet. Somit waren die Erinnerungen an die Websites und auch das allgemeine Stimmungsbild zu Dokumentationen noch sehr präsent.
Zu Beginn werden die allgemeinen Fragen analysiert. Dies waren die Fragen, welche keinen Bezug zu einer der Low-Code Anwendungen hatten. Diese Fragen wurden am Ende des Bogens gestellt.
Bewertet wurde wie folgt:
Die erste Frage lautete "Wenn Sie die Wahl hätten, welche Dokumentation würden Sie bevorzugen?". Antworten konnten die Testpersonen mit einer der drei Low-Code Anwendungen, "Ninox", "Simplifier" oder "Appian". Diese Frage war absichtlich so gewählt, auch wenn die Antwort eine sehr subjektive Aussage ist, um das allgemeine Stimmungsbild aller Befragten und deren Wahl "aus dem Bauch heraus" zu erhalten. Aus Gruppe G1 wählten 60% "Ninox" und jeweils 20% wählten "Simplifier" und "Appian". Aus Gruppe G2 wählte keiner die Dokumentation von "Simplifier". 57% aus dieser Gruppe wählten "Ninox" und die restlichen 43% "Appian".
Die nächste Frage dreht sich um die Auffindbarkeit der Dokumentation. Die Testpersonen konnten bis zu drei Optionen auswählen. Ziel dieser Frage ist es, herauszufinden, wo es den Testern am wichtigsten ist, die Dokumentation zu finden. Hierbei wurde eine Auswahl als "must-have"-Anforderung bewertet und keine Auswahl der Möglichkeit als "nice-to-have"-Anforderung. Alle Testpersonen geben an, dass eine gut Auffindbarkeit über Google und über die Hauptwebsite sehr wichtig ist. 60% der Teilnehmer aus G1 gaben zusätzlich an, dass die Dokumentation auch über die jeweilige Anwendung zugänglich sein sollte. Bei den Testpersonen aus G2 meinen dies sogar 72%. Demzufolge hatte die "Auffindbarkeit über Google", sowie die "Auffindbarkeit über die Hauptwebsite" eine allgemeine Zustimmung von 100%, während die "Auffindbarkeit über die Anwendung" nur eine allgemeine Zustimmung von 67% erhielt. Die Grenze, welche verwendet wurde um "must-have"- von "nice-to-have"-Anforderungen zu trennen wurde bei allen Fragen vorher auf größer 80% festgelegt. Die ersten beiden Punkte wurden somit als "must-have"-Anforderungen eingestuft, die letzte Option wurde zu einer "nice-to-have"-Anforderung.
Bei den nächsten beiden Fragen wurden die Antworten wie folgt ausgewertet. Die Auswahlmöglichkeiten "sehr wichtig" wurde als "must-have" angesehen, die Auswahlmöglichkei-

ten "eher wichtig" als "nice-to-have" und "eher unwichtig", sowie "sehr unwichtig" wurden als "unwichtig" gezählt. Die Auswahlmöglichkeit "egal" wurde keiner der Optionen zugeteilt.
Eine gute Suche finden 60% der Teilnehmer aus G1 "sehr wichtig", 20% meinen sie sei zumindest "eher wichtig". Aus der Gruppe G2 glauben 72% eine gute Suche sei "sehr wichtig". 14% finden eine gute Suche "eher wichtig", andere 14% meinen sie sei "unwichtig". Abschließend hat eine gute Suche eine allgemeine Zustimmung von 67% und ist somit eine "nice-to-have"-Anforderung.
Eine gute Struktur hingegen finden alle Teilnehmer "sehr wichtig". Diese hat eine allgemeine Zustimmung von 100% von beiden Gruppen erhalten und ist damit eine "must-have"-Anforderung.
Bei der nachfolgenden Frage wurden die Auswahlmöglichkeiten erneut unterschiedlich gewertet. Die Option "ja" wurde als "must-have" angesehen, die Option "nein" als unwichtig. Bei dieser Frage ging es darum, ob sich die Testpersonen eine Dokumentation nach "Leveln" aufgebaut wünschen würden. Das heißt, sie könnten sich z.B. am Anfang entscheiden, ob Sie den "Einsteiger-", "Normal-" oder "Experten"-Inhalt zu sehen bekommen. Die Antworten waren bei dieser Frage sehr gegensätzlich. Während alle "Nicht-Programmierer", also Teilnehmer der Gruppe G1, diese Frage mit "ja" beantworteten, wollten nur 30% der Programmierer, Teilnehmer der Gruppe G2, diese Möglichkeit angeboten bekommen. Die Restlichen 70% antworteten mit "nein". Dadurch ist die allgemeine Zustimmung bei diesem Thema bei 59%.
Die nächste Frage lautete "Wie oft finden Sie Lösungen zu Ihren Problemen in den Dokumentationen von Softwares?". Diese Frage sollte eine Antwort darauf geben, wie oft Personen ihre Erfahrung mit der Dokumenation, aufgrund der Tatsache, dass sie die Lösung ihres Problems in dieser finden, positiv bewerten. Bei dieser sollten nur die Testpersonen antworten, welche solch eine Erfahrung schon einmal erlebt hatten. Das waren zum größten Teil Personen der Gruppe G2, lediglich von einer Person aus G1 gab es eine Antwort. Die Option "gar nicht" wurde nie gewählt, daher wurde sie in der Tabelle nicht berücksichtigt. Die Antworten "sehr oft" und "eher oft" wurden in der Auswertung zu einer Antwort zusammengefasst - ebenso die Positionen "manchmal" und "eher weniger". Die eine Person aus G1 gibt an, "manchmal/eher weniger" eine Lösung in der Dokumentation zu finden. Von den Testern aus G2 geben die Meisten (56%) an, die Lösung "sehr oft/eher oft" in der Dokumentation zu finden. Das Gesamtergebnis daraus beläuft sich auf 56% der Personen, welche ihre Antworten "sehr oft/eher oft" in der Dokumentation finden.
Eine weitere Frage war, ob bei einer guten Dokumentation auf das Community-Forum für Fragen verzichtet werden kann. Die Teilnehmer konnten mit "ja" oder "nein" antworten. Aus Gruppe G1 wählte die Mehrheit mit 80% "nein". Ebenso aus Gruppe G2 waren die Meisten (72%) dagegen. Demnach ist die allgemeine Zustimmung zu dieser Frage 24%.
Die letzte Frage, "Wie wichtig ist Ihnen eine gute Dokumentation bei der Wahl einer Software?", wurde ebenfalls etwas zusammengefasst ausgewertet. Hierbei konnten sich wieder alle Teilnehmer beteiligen. Das Ergebnis war, dass alle Testpersonen aus beiden Gruppen "eher/sehr wichtig" als Antwort wählten. Die allgemeine Zustimmung, dass eine gute Dokumentation wichtig bei der Wahl einer Software sei, lag bei 100%.

Nun werden die Antworten der fehlenden Umfrageseiten analysiert. Diese bezogen sich

jeweils auf eine der drei Low-Code-Dokumentationen, welche sich vorher von den Testpersonen angeschaut wurden. Diese Analyse wird nicht so ausführlich sein, wie die davor, da es meist sehr eindeutige Ergebnisse gab. Dennoch werden interessante Befunde einzeln hervorgehoben.
Begonnen wird mit den Antworten zu Ninox. In Gruppe G2 hat Ninox eine durchweg positive Bewertung erhalten. Alle Testpersonen aus dieser Gruppe geben bei allen Punkten "eher gut" an. Auch in Gruppe G1 erhält Ninox bei fast allen Punkten die Wertung "eher gut", lediglich die Suchfunktion finden Teilnehmer dieser Gruppe "eher schlecht". Das Gesamtergebnis beläuft sich trotzdem auf "eher gut" in allen Punkten. Ninox Endbewertung mit den höchsten Prozenten ist demnach auch "eher gut" mit 50%.
Bei der nächsten Low-Code Dokumentation von Simplifier war die Meinung der Tester nicht einheitlich. In Gruppe G1 erhielt Simplifier überwiegend die Bewertung "mäßig". Bei der Auswahlmöglichkeit zur Strukturierung der Dokumentation wählten 60% "eher gut", während die Mehrheit mit 40% den Community-Einbezug auf "eher schlecht" schätzte. Aus Gruppe G2 gaben ebenfalls die meisten Tester die Bewertung "mäßig". Bei den Punkten "Optik" und "Suchfunktion", wählte die Mehrheit jedoch "eher gut" (bei "Optik" 42,8% und bei "Suchfunktion' sogar 71,4%). In der Gesamtwertung kommt Simplifier jedoch nur auf ein Ergebnis von "mäßig" mit 34,7% , als Bewertung mit den höchsten Prozenten.
Auch zu Appian gibt es unterschiedliche Meinungen in beiden Gruppen. Während Tester der Gruppe G1 diese Dokumentation als "eher gut" empfinden, meinen Teilnehmer der Gruppe G2 sie sei "mäßig". Gruppe G1 hat nur bei dem Punkt "Strukturierung" eine Mehrheit bei "mäßig" von 40%. Bei Gruppe G2 gibt es auch Ausnahmen. Hier findet die Mehrheit mit 42,8% den Gesamteindruck "eher gut" und sogar 42,8% finden Beispiele in dieser Dokumentation als "sehr gut". Dennoch ist der Gesamteindruck von Appian, ebenso wie der von Simplifier, "mäßig". Zwar gibt es hier nicht so ein Eindeutiges Ergebnis wie bei Simplifier, jedoch liegt die Mehrheit mit 19,45% bei "mäßig".

Kapitel 6

Erstellung der funktionalen Anforderungen

6.0.1 Was sind funktionale Anforderungen?

Anforderungen sind in dem Gebiet der Softwareentwicklung die Wünsche, die der Kunde an sein zu entwickelndes System besitzt. Diese Anforderungen dienen dann dem Entwickler, dieses System mit all seinen Spezifikationen entwickeln zu können. "Eine minimale Anforderungsformulierung beinhaltet eine Beschreibung der geforderten Anforderung, eine Beschreibung des zu Grunde liegenden Problems oder Bedürfnisses und ein Abnahmekriterium, gegen das die Erfüllung der Anforderung objektiv geprüft werden kann" [Bra16, Seite 3].
Des Weiteren können Anforderungen in zwei Kategorien eingeteilt werden, funktionale und nicht-funktionale Anforderungen. Der Unterschied liegt hierbei in der Qualität der Anforderungen. Während es bei einer funktionalen Anforderung reicht, die Anforderung zu beschreiben, muss bei einer nicht-funktionalen Anforderung ebenfalls beschrieben werden, mit welcher Qualität dies passiert [Bra16, vgl. Seite 3 f.]. Da diese Arbeit das erste Konzept für die Low-Code-Dokumentation der Software "yeet" sein soll, werden noch keine nicht-funktionalen Anforderungen definiert, sondern nur funktionale.
Diese Anforderungen zu konzipieren ist ein wichtiger Schritt, um das Konzept mit den relevanten Wünschen der Benutzer zu füllen.

6.0.2 "must-have-" und "nice-to-have-Anforderungen"

Die eigentliche Methode, die hinter dieser beliebten Einteilung von Anforderungen steht, ist die MoSCoW-Priorisierung. Diese wurde mit der agilen Projektmanagementmethode "Dynamic Systems Development Method" (DSDM), welche 1994 entwickelt wurde, erstmals angewendet und ist bis heute eine beliebte Methode Eigenschaften einzuteilen.[Sta97, vgl. xiii + Seite 28 f.]. Das Akronym MoSCoW steht dabei jedoch für vier Arten der Priorisierung "Must have","Should have", "Could have" und "Won't have"[Sta97, vgl. Seite 29]. In vielen Projekten wird heutzutage lediglich mit "must-have" und "nice-to-have" gearbeitet, wobei "nice-to-have" hierbei das "Should have" und "Could have" ersetzt [Conb, vgl. Absatz 8].

Aus vereinfachten Gründen wird ebenfalls in dieser Arbeit mit jener Methode gearbeitet.

6.0.3 Vorgehensweise

Aus den vorherigen Analysen lassen sich Merkmale herausfiltern, welche speziell durch die Analyse von Low-Code-Dokumentationen entstanden sind. Diese Eigenschaften wurden in den Tabellen, welche in Kapitel 5 erarbeitet wurden, gesammelt und dort in "must-have"- und "nice-to-have"-Anforderungen getrennt. Diese Unterscheidung wurde anhand der kontinuierlichen Erwähnung der Anforderung sowie der Erwähnung der Anforderung in beiden Gruppen getroffen. Das bedeutet, wurde eine Anforderung in beiden Gruppen von mehr als einer Person positiv vermerkt, wurde sie eine "must-have"-Anforderung. Wurde sie nur in einer der beiden Gruppen mehr als zweimal erwähnt, galt sie als "nice-to-have"-Anforderung. Zusätzlich wurden noch weitere Methoden zur Kategoriebestimmung verwendet.
Im folgenden Abschnitt werden die Anforderungen beider Kategorien gesammelt, definiert und analysiert. Ergebnisse aus dem User-Test / Interview, sowie der Umfrage werden hier gemeinsam ausgewertet. Um einen besseren Überblick zu wahren, werden die jeweiligen Anforderungen in Kategorien eingeteilt. Danach folgt ein Vergleich mit den Anforderungen aus Kapitel 4.1, in dem die Anforderungen der API-Dokumentationen vorgestellt wurden.

6.1 Must-Have-Anforderungen

6.1.1 Dokumentation allgemein

Daten der Umfrage, sowie Daten des User-Tests ergaben, dass die Dokumentation **"sehr gut über Google auffindbar"** sein sollte. Die Umfrage zeigte, dass es für alle Teilnehmer (100%) ein "must-have"-Feature ist. Dabei sollte es einer der erste Link sein, wenn "xxx Dokumentation" gegoogelt wird. Bei dem User-Test googelten auffällig viele Programmierer (48%), welche in Gruppe G2 waren, nicht gleich nach der Dokumentation, sondern versuchten über deren Website ein Ergebnis zu erhalten. Aussagen dazu waren unter anderem: "Offizielle Sachen haben meistens eine Doku. Wenn es um eine neue Anwendung geht, würde ich erst einmal über die Website gehen."[T9].
Daher ist die nächste wichtige Anforderung, dass die Dokumentation **"sehr gut über die Website der Anwendung zu finden"** sein muss. Ebenfalls bestätigt die Umfrage diese Erkenntnis, dort waren alle Teilnehmer mit 100% dafür. Da die meisten Testpersonen, zuerst in der Headernavigation und danach im Footer nach der Navigation gesucht haben, ist es ratsam dies ebenfalls zu berücksichtigen. Dokumentation sollte also über die Navigation im Header der Website zu finden sein und als Link auch noch einmal im Footer vorkommen. "Die wichtigsten Sachen sind meistens im Footer", meinte Testperson T2 dazu.
Eine **"gute Strukturierung"** der Dokumentation war 100% der Tester wichtig. Dies konnte durch die Umfrage als weitere "must-have"-Anforderung ermittelt werden. Die Strukturierung bezieht sich hauptsächlich auf die Navigation der Seite. Zusätzlich ist es ebenfalls wichtig eine gute Struktur auf den Dokumentationsseiten zu besitzen. Dieses Ergebnis der Umfrage stimmt mit der Anforderung **"Seitenaufbau muss gleiche Struktur haben"** übe-

rein.
Die nächste Anforderung behandelt das FAQ. Einerseits ist es Teil der Dokumentation, andererseits erwarteten es viele Testpersonen auf einer eigenen Seite (8 Testpersonen aus beiden Gruppen). Die Anforderung **"FAQ braucht eigene Seite"** gehört damit jedoch zu den "must-have"-Anforderungen einer Low-Code-Dokumentation.
"FAQ sollte im Footer zu finden sein", eine weitere Anforderung, welche das FAQ der Seite betrifft. Die Hälfte der Testpersonen suchte das FAQ im Footer der Seite (T10, T2, T4, T5, T7, T12).
Eine oft genannte Eigenschaft einer Dokumentation war **"Dokumentation nicht verstecken, besonders nicht unter Community"**. Viele Testpersonen (T10, T4, T5, T9, T3, T6, T7) waren bei Simplifier und Appian nicht in der Lage die Dokumentation über die offizielle Website zu finden. Dies lag immer daran, dass diese unter der Kategorie Community "versteckt"[T11] war. Die Meisten hätten die Dokumenation dort nie erwartet, eine Dokumentation hätte direkten Produktbezug (T4) und sei ein wichtiges Element der Software, welches direkt auf der Hauptwebsite zu finden sein sollte (T4).
Die Anforderung **"Suche in der Dokumentation sollte nur Inhalte der Dokumentation anzeigen oder so einstellbar sein"**, bezieht sich auf die generelle Suche der Dokumentation. Die Testpersonen fanden es verwirrend, wenn bei einer Suche innerhalb der Dokumentation Ergebnisse aus dem Forum oder Blog der Software angezeigt wurden. Viele bemerkten dies nicht gleich und klicken aus versehen auf solch ein Suchergebnis (T2, T3, T5, T6, T9). Daher erfolgte die Anmerkung "bei Suche sollte man Häkchen setzten können, wo man suchen will"[T8].
Die letzte allgemeine Anforderung an eine Dokumentation knüpft an diesen Punkt an. **"Nicht zu viele Unterschiedliche Seiten für Dokumentation und Hilfe haben"**, diese Anmerkung wurde häufig erwähnt, nachdem unwissend auf einen falschen Link geklickt wurde und dieser aus der Dokumentation heraus führte. Die Testpersonen landeten meist in Foren oder anderen Community-Seiten der Software. Dies führte zu negativen Einstellungen zu der Dokumentation. "Die Strukturierung ist nicht so gut, dass man als auf unterschiedlichen Seiten ist und nicht alles in einer hat"[T7].

6.1.2 Startseite

Durch den User-Test konnten einige Anforderungen zur Startseite einer Dokumentation gesammelt werden. Welche davon "Must-Have"-Features sind, folgen in der kommenden Aufzählung. Die Startseite soll **"übersichtlich, aufgeräumt, simpel und klar aufgebaut"** sein. Dies deckt sich mit den Ergebnissen der Umfrage, bei der die Low-Code-Plattform "Ninox" die beste Bewertung für die "Optik" mit 50% allgemeiner Zustimmung hatte. Denn Ninox ist die optisch modernste und simpelste der drei Plattformen. Sie ist sehr übersichtlich aufgebaut und sieht allgemein sehr aufgeräumt aus. Bemerkungen aus der Umfrage zu dieser Dokumentation stimmen dem gleichermaßen zu. "Sehr übersichtlich und modern"[T6] und "War die am einfachsten zu findende und übersichtlichste Dokumentation" [T12] wurde kommentiert. Es ist jedoch auch ein Fakt, dass Ninox die Low-Code-Anwendung mit dem wenigsten Features und Umfang ist. Somit kann eine übersichtliche Dokumentation einfacher erzeugt werden. Es werden nicht so viele Informationen benötigt, welche in der

Dokumentation Platz finden müssten.
"Benennung muss immer gleich sein",bei Google, im Navigations-Menü, auf der Startseite, im Breadcrumb-Menü, etc. Diese Anforderung bezieht sich auf eine Einheitliche Benennung der Dokumentation in unterschiedlichen Instanzen. Es wurde von zwei Testpersonen [T8 + T10] bei der Dokumentation von Ninox bemängelt, dass es "verwirrend" sei, wenn die Dokumentation mal "Benutzerhandbuch", "Handbuch" oder "Hilfe" genannt wird. Da es zwei Personen der Gruppe G1 aufgefallen ist, kann angenommen werden, dass gerade "Nicht-Programmierer" eher auf solche Aspekte achten. Hier sei eine "klare und wiederholende Strukturierung wichtig."[T10].
Mit der Anforderung **"Begrifflichkeiten müssen klar gewählt werden"** sind die Kategorienamen der Hilfsangebote gemeint. Das heißt, die Testpersonen haben es bevorzugt, wenn eine Kategorie wie beispielsweise das "Forum", auch klar und deutlich danach benannt wird. Die Dokumentation der Anwendung Simplifier bot zusätzlich zu den Überschriften, noch kleine Zusammenfassungen, welche ebenfalls erläuterten, was hinter dieser Kategorie zu finden ist. Jedoch wurde nur von zwei Testern [T11 + T9] erwähnt, dass diese Unterüberschriften hilfreich waren.
Eine Suche fanden alle Testpersonen hilfreich und sinnvoll. **"Suche ist wichtig - muss gut sichtbar sein, sollte aber nicht zu groß sein"**. Viele Testpersonen [T3, T11 + T5] fanden es gut, dass die Suche immer präsent auf der Startseite hervorsticht. Lediglich ein Tester [T8] fand, dass sie trotzdem nicht zu groß und auffallend sein sollte.
"Startseite darf Benutzer nicht erschlagen, darf nicht zu voll sein". Bei der Dokumentation von Simplifier fanden 2 Testpersonen [T6, T7] bereits, dass die Startseite zu voll oder zu "unübersichtlich"[T7] sei. Jedoch meinte ein Großteil der Testpersonen es sei "noch im Rahmen"[T11]. Bei der Dokumentation von Appian meinten jedoch schon 4 Tester sie würde einen "erschlagen"[T2]. Daher sollte die Startseite einer optimierten Low-Code-Dokumentation auf keinen Fall zu voll mit Informationen sein.
Ebenfalls darf die Startseite einer Dokumentation **"farblich und inhaltlich nicht überladen"** sein. Diese Anforderung erwähnten mehrere Personen aus beiden Gruppen, insgesamt fünf Tester. Diese Anforderung überschneidet sich wieder mit den Umfrageergebnissen, bei denen Ninox im Gesamteindruck mit 75% und bei der Optik mit 50% die beste der drei Anwendungen war.
Die nächste Anforderung wäre: **"Soll auf den ersten Blick alle wichtigen Infos beinhalten."** Dieser widerspricht sich etwas mit dem Vorherigen, denn dort wünschen sich die Testpersonen eine Seite welche "inhaltlich nicht überladen sein" soll. Der schmale Grat von "so viele Informationen, wie nur möglich enthalten" und "den Endbenutzer nicht erschlagen" muss hier irgendwie eingehalten werden. Des Weiteren wurden hier positive Anmerkungen getätigt, wie "viel Input, mit wenig scrollen"[T3]. Ebenso wurde häufiger vermerkt, dass die verschiedenen Kategorien gut voneinander abgetrennt sein müssen, "Oberpunkte schön in eigene Boxen getrennt"[T10].

6.1.3 Dokumentationseintrag

Bei einem Dokumentationseintrag sind **"Beispiele"'** eines der Wichtigsten "must-haves". Damit sind sich viele Teilnehmer einig [T9, T12, T8, T5, T10]. "Gerade beim Program-

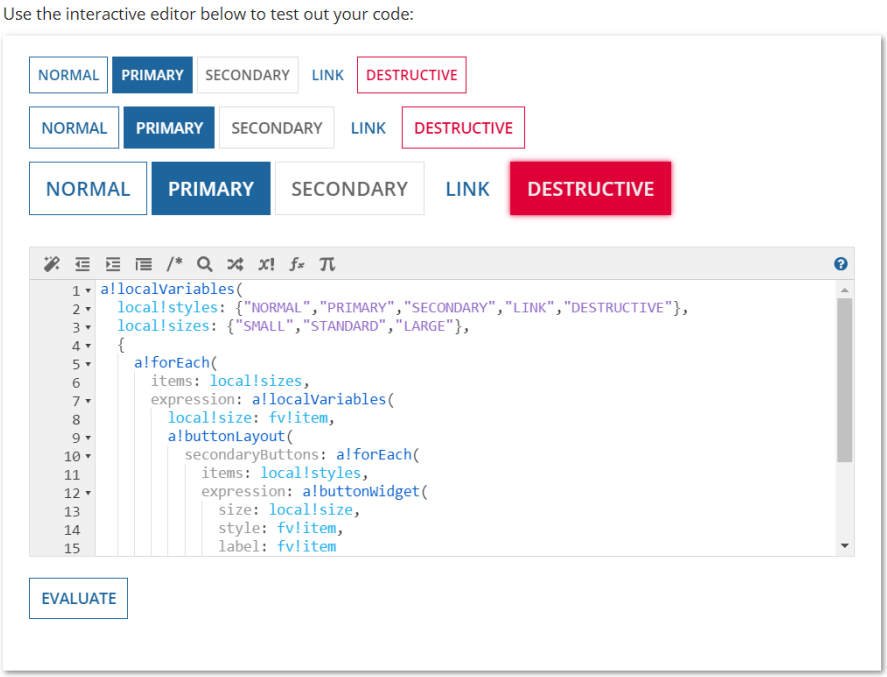

Abbildung 6.1: Beispiel eines interaktiven Code Editors, Quelle: https://docs.appian.com/suite/help/21.3/Button_Component.html

mieren oder generell, wenn man Software erstellt, sind Beispiele sehr, sehr hilfreich" [T9]. Darunter zählen neben Codebeispielen auch Bilder, auf denen das vorher Erklärte ebenfalls visuell veranschaulicht wird. Gerne gesehen waren dabei Bilder von der Anwendung selbst, "Screenshots von Anwendung selbst ist Top - reale Beispiele sind immer besser" [T12].
Die **"interaktiven Componenten"**, welche bei der Low-Code-Software "Appian" sichtbar hervortraten, waren als Beispiele sehr beliebt. Dort wurde nicht einfach ein Bild von einem "Button" gezeigt, sondern deren Komponente "Button" wurde real eingebunden, sodass damit interagiert werden konnte. Veranschaulicht wird dies in Abbildung 6.1. Des Weiteren wird hier mit einem Codeeditor gearbeitet, welcher direkt mit den angezeigten Komponenten verbunden ist. Das heißt, würde hier etwas verändert werden, würde dies direkt die Buttons beeinflussen. Dass die Beispiele von Appian generell von den Testpersonen bevorzugt wurden, ergibt sich auch aus den Ergebnissen der Umfrage. Der größte Anteil der Tester (41,7%) stimmte hier ab, dass die Beispiele bei Appian "sehr gut" sind.
Die Dokumentationsseite sollte auch ein **"gutes Text-Bild-Verhältnis"** besitzen. Dies bezieht sich darauf, dass Tester es gut fanden, wenn es nicht nur Textabschnitte am Stück gab, sondern in passenden Abständen auch erklärende Bilder dabei waren, "gut, wenn man Bilder hat, reiner Text ist immer schwierig zu lesen oder Leute dazu zu begeistern"[T5].
Einige Testpersonen (T4 ,T8 ,T5) finden eine **"Anfangsbeschreibung"** wichtig und erforderlich. Eine Person [T5], merkte außerdem an "gut wären schnelle Infos für Erfahrene und oben einen Link für erweiterte Infos für Neulinge". Hier wurde sich eine Unterteilung

gewünscht, welche Dokumentationsartikel in verschiedene Erfahrungs-Stufen gliedert. Dies deckt sich jedoch nicht mit den Ergebnissen der Umfrage. Hier waren die Meinungen sehr gespalten, was eine Einteilung nach "Leveln" anging. Da alle Testpersonen aus G1 dies bevorzugen würden, jedoch nur 30% der Tester aus G2 dafür gestimmt haben, liegt die allgemeine Zustimmung nur bei 59%. Damit ist diese Anforderung nur knapp ein "nice-to-have" Feature geworden.
Was die meisten (T12, T7, T6, T5, T11, T8) nützlich fanden, waren die Links in der Dokumentation von Simplifier, welche zu anderen Artikeln der Dokumentation führten. Über diese Links navigierten sich sogar 2 Tester (T8+T9) zu einer speziellen Seite der Dokumentation bei einer anderen Aufgabe. Dadurch wurde die Anforderung **"verlinkte Schlagworte auf andere Dokumentationsseiten"** zu einem "must-have"-Feature.
Ein Tester merkte bei der Dokumentation von Ninox zu den Bildern negativ an, "sieht aus wie mit Windows, Shift+S ausgeschnitten und reinkopiert" [T2]. Auch andere (T2, T10, T8, T4, T6, T9) gaben an, dass die Bilder bestmöglich hervorgehoben, gut erkennbar und groß genug sein sollten. Die Anforderung **"Bilder gut hervorgehoben und gut erkennbar"** ergab sich daraus.
"Seitenaufbau muss gleiche Struktur haben", eine weitere Anforderung an Low-Code-Dokumentationen. Hiermit ist der kontinuierliche Aufbau einer Seite gemeint. Die Tester fanden es verwirrend, dass es zum Beispiel bei Dokumentationsseiten von Ninox Unterschiede in der Struktur gab. "Bilduntertitel mal da, mal nicht? - Text unter Bildern manchmal kursiv, manchmal nicht? - alles sehr verwirrend und anstrengend", dies und noch Anderes meinte beispielsweise Tester T8 zu dem Artikel von Ninox. Verdeutlicht, wird diese Anforderung auch damit, dass eine gute Gesamtstruktur 100% der Tester in der Umfrage "sehr wichtig" fanden.

6.1.4 Untere Hilfen in Dokumentationseintrag

Zu den unteren Hilfen zählen alle Angebote, welche eine Dokumentation in dem Artikel zusätzlich hatte, um dem Endnutzer zu weiterzuhelfen. Da sich diese bei den ausgewählten Dokumentationen ausschließlich unten am Ende des Artikels wiederfanden, wurde die Kategorie entsprechend benannt.
Bei angebotenen Hilfen wie, passende Foreneinträge zu dem Thema oder andere vorgeschlagene Artikel der Dokumentation, war es den Testpersonen beider Gruppen sehr wichtig, dass nur sinnvolle Beiträge angezeigt werden. Daraus entstand die Anforderung **"Forum Hilfen - wichtig, dass sinnvolle Artikel angezeigt werden"**, sowie **"Related Articles - wichtig, dass sinnvolle Artikel angezeigt werden"**. Viele Testpersonen (T8, T4, T6, T7) haben schon einmal schlechte Erfahrungen mit irreführenden Vorschlägen solcher Hilfen gemacht, daher diese Skepsis.
Viele Testpersonen finden die Option, die Dokumentation als **"PDF-Dokument"** herunterzuladen, sehr nützlich und würden dies auch in Anspruch nehmen (T8, T11, T10, T4, T6, T9, T12). Bei dieser Anforderung müsste jedoch noch die perfekte Platzierung dieser Option evaluiert werden. Denn wie ein Tester anmerkte, sollte die Möglichkeit die gesamte Dokumentation als PDF herunterzuladen nicht am unteren Ende einer bestimmten Dokumentationsseite vorhanden sein [T4].

Bei jeglicher Art der Hilfen sollte jedoch drauf geachtet werden, dass der Endnutzer nicht von den Angeboten "erschlagen"[T11] wird. Daraus resultiert die letzte "must-have"-Anforderung der Hilfen, **"Hilfen dürfen nicht zu vollgepackt sein - Überschrift reicht"**. Die Dokumentation von Appian hatte als angebotene Hilfe eine sehr ausführliche Auflistung, welche die "Related Patterns" inklusive kurzer Beschreibung des Artikels anzeigte. Diese fanden jedoch alle Testpersonen zu voll, "nicht gut, hätten sie lieber rausgelassen"[T7].

6.1.5 Codebeispiel in Dokumentationseintrag

Die einzige "must-have"-Anforderung, welche beide Gruppen als wichtig erachten, ist **"farbliches Highlighting"** bei Codebeispielen. Darauf wurde bei Appian geachtet und dies ist auch in der Umfrage zu erkennen. Denn obwohl Appian eine durchschnittliche Bewertung von "mäßig" in der Umfrage erhalten hat, wurden von 41,7% aller Tester die Beispiele als "sehr gut" eingestuft. Alle anderen Anforderungen wurde lediglich gruppenintern genannt und wurden somit als "nice-to-have"-Anforderung gewertet.

6.1.6 Navigation der Dokumentation

Bei der Navigation gab es einige Anforderungen, welche in beiden Gruppen genannt wurden. **"Hauptpunkte (brauchen) klare Namen"**, diese Eigenschaft bezieht sich auf die Namen der Überschriften sowie Unterpunkte, welche im Navigationsmenü aufgelistet sind. Es sollte vermieden werden " verallgemeinerte Formulierungen" [T8] zu benutzen.
"Verschachtelungen mit nur einem Unterpunkt sind unnötig", diese Anforderung wurde ebenfalls von mehreren Testern aus den beiden Gruppen (T9, T11, T2, T5) genannt. Dort wurde kritisch hinterfragt, falls ein Überpunkt nur einen Unterpunkt hatte. Dies wurde von zwei Testern (T9, T5) sogar als "unnötig" angesehen.
Die nächste Anforderung, welche sich auf die Navigation der Dokumentation bezieht ist **"nicht zu viel auf einmal 'aufklappen' "**. Damit ist die Funktionalität eines Navigationsmenüs gemeint, welche sich auf das ein- und ausklappen der einzelnen Menüpunkte bezieht. Oft fanden es Testpersonen zu "erschlagend" (T2, T9, T3, T12), wenn die gesamte Navigation auf einmal angezeigt wird. Im besten Fall sollten die Unterpunkte erst vom Nutzer selbst aufgeklappt werden, sobald dieser sie benötigt.
Abschließend ist die letzte Anforderung, **"Navigation darf Seite nicht einengen/nicht zu viel Platz wegnehmen"**. Diese Anforderung wurde von einigen Testpersonen (T3 ,T2, T7) besonders bei der Dokumentation von Appian erwähnt. Dies wurde dadurch hervorgerufen, dass die Navigation auf der linken sowie der rechten Seite der Dokumentation angezeigt wurde. Dabei wurden Überpunkte links angezeigt und Unterpunkte, beziehungsweise die Inhalte der Seite, rechts. Manche Tester wiesen darauf hin, dass die zusätzlich vorhandene obere Leiste zum Gefühl der "Platzangst"[T2] führte. Alle Tester aus G1 und drei aus G2 (T6, 9, T12) merkten jedoch an, dass diese zusätzliche Navigation auf der rechten Seite nützlich sei und zu einer besseren Übersicht der Seite führt. Ebenso kann eine rechte Navigation helfen, eine überladene Navigation links zu vermeiden (T4). An dieser Stelle erfordert es wieder eine gute Kombination zu kreieren.

6.2 Nice-To-Have-Anforderungen

Die "nice-to-have"-Anforderungen werden im nachfolgenden Abschnitt nicht so detailliert beschrieben wie die "must-have"-Anforderungen zuvor. Trotzdem werden wichtige Zitate oder Erkenntnisse kurz vorgestellt.

6.2.1 Dokumentation allgemein

"Nach Level aufgebaut", bei dieser Anforderung waren sich die beiden Testgruppen nicht einig. Programmierer stimmten in der Umfrage größtenteils dagegen, während "Nicht-Programmierer" zusammen dafür stimmten. In dem User-Test wurde von manchen Testern jedoch innerhalb der Anforderung der "Anfangsbeschreibung" eine Unterteilung in verschiedene Niveaus gewünscht. Dadurch wurde die Anforderung eine "nice-to-have"-Anforderung, wobei diese Thematik noch weitere Forschung benötigt.
Die Anforderung **"Auffindbarkeit über Anwendung"** hatte in der Umfrage lediglich eine allgemeine Zustimmung von 67% und fällt damit unter die "nice-to-have"-Anforderungen.
Ebenso hatte die Anforderung **"gute Suche"** nur eine allgemeine Zustimmung von 67%. Diese Tendenz zeigte sich auch im User-Test, denn viele Tester meinten solche Suchen sind meist nicht zielführend, "Google Suche viel viel schneller und besser als Seitensuche"[T7].
Aus der Umfrage geht hervor, dass der Gesamteindruck von Ninox den Testpersonen am besten gefallen hat. Ninox besitzt eine sehr moderne, aufgeräumte Dokumentation. Durch seinen geringen Umfang ist die Dokumentation nicht groß und "erschlägt" den Nutzer nicht mit Informationen. Die Anforderung **"Gesamtbild wie Ninox"** bedeutet, dass bei der Entwicklung auf die gerade genannten Eigenschaften geachtet werden muss. In der Umfrage wurde bei diesen Fragen mithilfe der Likert-Skala abgestimmt. Da diese und die kommenden Anforderungen nie bei einer Mehrheit von "sehr gut" waren, sondern nur bei "eher gut", wurden sie lediglich als "nice-to-have"-Anforderungen gewertet.
Die folgenden Anforderungen stammen ebenfalls aus der Umfrage und können konform zu der zuvor vorgestellten Anforderung behandelt werden. Eine **"Optik wie Ninox"**, eine **"Suchfunktion wie Simplifier"**, die **"Strukturierung wie Ninox"**, **"Beispiele wie Appian"** und ein **"Community Einbezug wie Ninox"**. Bei der Erstellung einer neuen Dokumentation kann an diesen Anforderungen ein Beispiel genommen werden.
Weitere Erkenntnisse aus dem User-Test waren:

- Dokumentation sollte einen Abschnitt am Anfang besitzen, welcher die Installation beschreibt. Zusätzlich sollte dieser jedoch noch auf andere Wege auffindbar sein.
- In der Dokumentation wird kein Einsteiger Content erwartet, jedoch sollte es auf der generellen Hilfeseite einen Abschnitt dafür geben.
- Die Dokumentation ist Hauptanlaufpunkt für spezifische Suchen, ebenso wichtig sind dafür jedoch auch die Suche generell sowie das Forum.

Die Testpersonen wünschten sich außerdem folgende Eigenschaften als "nice-to-have"-Feature.
"Auf Hilfeseite über Suchen FAQ auch direkt FAQ vorgeschlagen bekommen", die

Testpersonen haben häufig versucht das FAQ über die angebotene Suche zu finden. Durch die Eingabe "FAQ" wurde jedoch nie die Startseite des FAQ's angezeigt. Dies war eine Anforderung, welche sich von vielen gewünscht wurde.
Die Anforderung **"Cards/Bilder/Icons der Hilfskategorien sollten anklickbar sein"** wurde bei der Hilfsstartseite von Ninox sowie Simplifier entdeckt. Viele Testpersonen klickten dort intuitiv auf die Card, das Bild oder das Icon, welches die Dokumentation darstellen sollte. Dieses Element war bei den beiden Dokumentationsseiten nicht mit einem Link versehen und der Klick darauf führte zu nichts. Um diese unnötigen Klicks zu vermeiden und die Usability zu erhöhen, wurde diese Anforderung erstellt.
Zuletzt sollte die **"Suche in der Dokumentation (sollte) nicht überladen mit Infos sein"**. Viele Testpersonen haben wegen der Missachtung dieser Anforderung schon schlechte Erfahrungen mit der Suche in Dokumentationen gemacht. Aussagen wie "Ich würde die Suche auf einer Website nie verwenden, dann lieber direkt googlen, das geht schneller"[T7], waren häufig zu hören.

6.2.2 Startseite

Für die Hilfsstartseite der Dokumentation wurden folgende Anforderungen herausgearbeitet. Es sollten **"keine unnötigen/langweiligen Icons verwenden"**, dies wurde bei jedem der drei Dokumentationen kritisiert. Obwohl Ninox in der Umfrage in der Kategorie Optik am besten bewertet wurde, wurde oft kritisiert, dass zu viel Freiraum auf dessen Startseite besteht. Daraus entstand die Anforderung **"nicht zuviel Freiraum"**. Auf der Dokumentationsstartseite von Appian wurde sehr oft angemerkt, dass die Informationsblöcke "What's new..." nicht im Vordergrund stehen sollten, wie es zurzeit der Fall ist, daher erfolgte die "nice-to-have"-Anforderung **"Informationsblöcke sollten nicht Hauptaufmerksamkeit bekommen"**. Ein **"Live-Chat"** wurde sich von zwei Testpersonen gewünscht. Ob solch ein Chat nur für die Dokumentationsstartseite bedeutsam ist oder ob dieser auch innerhalb der Dokumentation einen positiven Mehrwert bietet, müsste weiter erforscht werden. Weiterhin könnte dabei ebenfalls der Gebrauch eines Chatbots sowie generelle KI-Technologien innerhalb einer Dokumentation analysiert werden.
Die folgenden beiden Anforderungen sind sehr trivial. Zum einem sollte darauf geachtet werden **"Einzelne Kategorien gut voneinander** [zu] **trennen"** und zum anderen sollten **"Icons/Bilder (sollten)** [den] **Inhalt der Kategorie verdeutlichen"**.
Die Anforderung **"Dokumentation sollte eigene Website haben"** ist ebenfalls selbsterklärend. Weiterhin ist hier zu beachten, dass diese Website nicht unter der Kategorie "Community" der Software zu platzieren ist. Dies führte während des Tests mehrfach für Unverständnis und Verwirrung unter den Testpersonen.

6.2.3 Dokumentationseintrag

Neben den "interaktiven Beispielen" welche sich als "must-have"-Anforderung durchgesetzt haben, ist die Anforderung **"viele Beispiele, reale Beispiele"** lediglich eine "nice-to-have"-Anforderung geworden. Die Testgruppe G2 der Programmierer deutete jedoch öfter heraus, wie relevant Beispiele für eine Dokumentation sind.

Die Anforderung **"darf nicht überladen wirken"** stimmt mit den Ergebnissen der Umfrage überein. Die Dokumentation von Ninox wurde vom Gesamteindruck, von der Optik sowie von der Strukturierung am besten bewertet. Wie bereits erwähnt setzt sich dort Ninox mit seinem übersichtlichen, aufgeräumten und modernen Gesamtbild durch.
"Nicht durch Navigation 'eingeengt' ", diese Anforderung wurde häufig innerhalb der Dokumentation von Appian hervorgebracht.
Eine Seite der Dokumentation sollte weiterhin **"übersichtlich"** sein, eine **"gute Textgröße"** besitzen sowie **"ausführliche klare Überschriften"** aufweisen.

6.2.4 Untere Hilfen in Dokumentationseintrag

Während die meisten Testpersonen bei den unteren Hilfen auf Interaktionen durch "Social Media" oder ähnliches verzichten würden, wurde sich dort öfters ein **"Kommentarfeld für den Artikel"** gewünscht.

6.2.5 Codebeispiel in Dokumentationseintrag

Viele Anforderungen, die Codebeispiele der Dokumentation behandeln wurden ausschließlich von der Gruppe G2 gestellt. Dies liegt wahrscheinlich daran, dass diese Personen durch ihre Arbeit schon viele unterschiedliche Einbindungen von Quellcode in eine Website oder Dokumentation gesehen haben. Dadurch haben sie mehr Vergleiche mit denen sie arbeiten können. Aus diesem Grund sind die folgenden Anforderungen "nice-to-have"-Anforderungen geworden.
Die Anforderung **"Kommentare im Code selbst"** bedeutet, dass sich gewünscht wurde innerhalb des gezeigten Codeabschnittes zusätzliche Kommentare eingebaut zu haben. Dies dient dazu, beim Kopieren des Abschnittes in die eigene Anwendung, nicht auf Erklärungen zu diesem Code verzichten zu müssen.
Die nächste Anforderung hat ebenfalls etwas mit der Funktion des Kopierens zu tun. Einfach **"rauskopierbar"** soll der Quellcode sein. Bilder, von denen der Code abgeschrieben werden muss sind sehr schlecht für den Workflow der Nutzer.
Erklärungen nur im Codeabschnitt selbst, wurden allein nicht gerne gesehen. Ein Codebeispiel sollte niemals ohne Erklärung dem Nutzer vorgestellt werden. Die Anforderung **"Erklärungen"** entstand daraus.
Des Weiteren soll der Code **"gut verständlich"** geschrieben sein und keine unnötigen, oder nicht zum dem Thema passenden, Zeilen enthalten.
Im Gegensatz zu den interaktiven Beispielen, wurde **"interaktiver Code"** als "nice-to-have"-Anforderung angesehen. Einerseits sei es "ganz cool, dass man was ausprobieren kann und Änderungen gleich sieht"[T6], andererseits würde die "Gefahr besteht, dass aus versehen was gelöscht wird" [T6] und dadurch Fehler entstehen, welche zu Verwirrung führen können.
Oft angemerkt wurde jedoch, wenn eine **"extra Erweiterung"** für die Anzeige des Codebeispiels vorhanden war. Diese Anforderung wurde häufig von Personen der Gruppe G1 angemerkt, dadurch würden die Codeabschnitte viel verständlicher und anschaulicher wir-

ken. Programme wie Codepen[1] oder GithubGist[2], bieten solche Erweiterungen an.
Nachfolgende Anforderungen beziehen sich auf die optischen und inhaltlichen Gegebenheiten, die ein Codebeispiel besitzen sollte:

- "Formatierung wichtig"
- "gleiche Struktur"
- "klare Überschriften"
- "übersichtlich"
- "platzsparend"
- "gut abgetrennt, gut erkenntlich"

6.2.6 Navigation der Dokumentation

Die "nice-to-have"-Anforderungen der Navigation besitzen folgende Eigenschaften. Die **"Überpunkte sollten auch anklickbar sein"**, dies war ein unscheinbarer, aber oft genannter Punkt unter Testern der Gruppe G1. Dort führte es zu Verwirrung und war "nervig", wenn es zu den übergeordneten Kategorien keine eigene Seite gab und der Punkt nur zur Gliederung der Navigation diente.
Die **"Punkte nicht zu gleichwertig gestalten"** war auch eine Anforderung, welche in G1 angemerkt wurde. Dies meint eine gute Strukturierung der Navigation zu haben, indem geeignete Über- und Unterpunkte verwendet werden.
Dennoch lautet eine weitere Anforderung, dass es **"nicht zu viele Unterpunkte"** sein sollen. Ein gutes Mittelmaß zu erzeugen sollte in der Navigation als oberstes Ziel angesehen werden. Gerade Kategorien mit nur einem Unterpunkt sollten vermieden werden.
Die letzte Anforderung an die Navigation war **"Breadcrumb o.ä. zum Zurückgehen "**. Dies wurde häufig bei Appian gewünscht, wenn Tester auf einer bestimmten Seite waren und nicht zur vorherigen Seite zurück fanden. Obwohl jeder Browser über einen "zurückgehen" Button verfügt, wurde es als negativer Aspekt angesehen, wenn eine Dokumentation solch ein Feature nicht besaß.

6.3 Vergleich der Anforderungen

Nun ist der Zeitpunkt die Annahme zu überprüfen, welche zu Beginn des Kapitels erstellt wurde. In Kapitel 4.1 wird angenommen, dass sich grundlegende Anforderungen beider Dokumentationsarten gleichen. Um diese Annahme zu bestätigen, werden im Nachfolgenden die soeben erforschten Anforderungen, welche Anforderungen für Dokumentationen von Low-Code-Softwares sind, mit den Anforderungen der verwandten Arbeiten verglichen, welche alle Anforderungen von API-Dokumentationen behandelten.

[1] https://codepen.io/
[2] https://gist.github.com/

Adjektive aus den verwandten Arbeiten wie "Genauigkeit, Vollständigkeit und Korrektheit" sowie "gut strukturiert" können mit der Anforderung "Gute Strukturierung" und "Soll auf den ersten Blick alle wichtigen Infos beinhalten" verglichen werden. Explizit wurde jedoch nie in den Tests die Genauigkeit, Vollständigkeit und Korrektheit der Dokumentation erwähnt. Dies lässt mehrere Schlussfolgerungen zu. Entweder ist diese Anforderung so banal, dass von den Testern davon ausgegangen wird, dass die offizielle Dokumentation einer Software sowieso korrekte und vollständige Informationen beinhaltet, oder die Tester haben es nie erwähnt, da sie es als unnötige Anforderung wahrnehmen. Die dritte Möglichkeit bestünde darin, dass sie es lediglich vergessen haben zu erwähnen, beziehungsweise, dass die Fragen nie so gestellt wurden, dass es hätte erwähnt werden können.
Die Forderung nach "Beispielcode" findet sich ebenfalls in den Anforderungen wieder. Es ist mit eine der wichtigsten Anforderungen an eine Dokumentation, da es in fast allen verwandten Arbeiten genannt wurde. Ebenso wurde während der User-Tests von vielen Testpersonen immer wieder betont wie entscheidend gute Beispiele für die gesamte Dokumentation sind. Ein Teilbereich davon, die Negativbeispiele wurden in dem Test nie erwähnt, werden daher wahrscheinlich als nicht sehr relevant eingestuft.
"Video-Tutorials" oder ein "Tutorialbereich" ist mit Kategorien wie "Knowledge Base" oder "Tutorial-Videos" aus dem User-Test zu vergleichen. Auch diese waren gerne gesehen und ein beliebter Punkt, wenn es um Eigenschaften einer Dokumentation geht.
Ebenfalls eine "Anleitung für den Einstieg" wurde sich von den Testern gewünscht. Diese spiegelt sich in der Anforderung "Anfangsbeschreibung" wider.
Nach API spezifischen Anforderungen, wie "API Übersicht", "API-Referenzen" oder "API-Richtlinien", wurde in dem Test nicht explizit gefragt. Jedoch kann angenommen werden, dass auch Low-Code-Anwendungen einen Abschnitt benötigen, indem solche Informationen zu finden sind.
"Leicht über Google auffindbar" ist eine Anforderung aus den verwandten Arbeiten, welche exakt mit den Ergebnissen des Tests und der Umfrage übereinstimmt. Es ist demnach auch eine Eigenschaft, in der sich API- und Low-Code-Dokumentationen gleichen.
Eine "Möglichkeit für Fragen, Antworten und Diskussionen" zu haben, ist eine weitere Anforderung der verwandten Arbeiten. Diese Eigenschaft war den meisten Testpersonen jedoch nicht wirklich relevant. Es wurde häufig angemerkt, dass eine Frage in einem Forum zu stellen eine "Verzweiflungstat" [T4] sei. Dennoch wurde von manchen Testern der Wunsch geäußert eine Kommentarfunktion zu den einzelnen Artikeln zu erhalten.
Das "Bewertungssystem" der verwandten Arbeiten bezieht sich auf die Beurteilung von Kommentaren in Diskussionen. Diese Eigenschaft konnte nicht getestet werden, da für die meisten Kommentarfunktionen eine Anmeldung in der entsprechenden Dokumentation nötig ist.
Aus der Umfrage konnte weiterhin festgestellt werden, dass eine Integration der Dokumentation, in die Anwendung selbst, eine "nice-to-have"-Anforderung sein muss.
Aufschlussreich ist die nächste Anforderung, welche in der Arbeit von Huesmann et al. [HZH19] genannt wird. "Verschiedene Niveaus" innerhalb einer Dokumentation wurde sich dort von Entwicklern gewünscht. Zur Überprüfung dieser Anforderung gab es in der Umfrage eine passende Frage dazu. Das Ergebnis ergab, dass lediglich "Nicht-Programmierer" an solch einer Umsetzung interessiert wären, während der Großteil der Programmierer dagegen

war.

Abschließend kann gesagt werden, dass grundlegende Anforderungen, welche sich auf die Struktur, Beispiele oder die Korrektheit der Dokumentation beziehen, sich in beiden Fällen gleichen. API spezifische Anforderungen bestehen ebenfalls bei einer Low-Code-Dokumentation, sind jedoch nur ein Teilaspekt. Low-Code-Dokumentationen zeichnen sich dadurch aus, dass sie viel mehr Bereiche abdecken müssen.
Ein weiterer Punkt, welcher bedacht werden muss, wenn es darum geht die Anforderungen zu vergleichen, ist die Durchführung der Tests an sich. Denn nicht alle Anforderungen, welche die verwandten Arbeiten erforscht haben, wurden dort berücksichtigt. Beispielsweise wurde keine konkrete Antwort darüber erhalten, ob sich die Testpersonen spezielles Hintergrundwissen zu der Low-Code-Plattform wünschen.
Ebenso muss Berücksichtigt werden, dass jede verwandte Arbeit andere Forschungsmethoden verwendet hat und dadurch unterschiedlichen Ergebnisse entstanden. Ähnliche Anforderungen wurden ebenso in die Liste aufgenommen wie Anforderungen, welche nur in einer verwandten Arbeit vorkamen. Dennoch gilt die These, dass sich grundlegende Anforderungen beider Dokumentationsarten gleichen, als bestätigt.

Kapitel 7

Auswahl der Technologien

7.1 Ausgewählte Technologien

Die gegenwärtig Arten von Dokumentationstechnologien wurden in Kapitel 2.2 erläutert. Im Internet wurde nach den zurzeit führenden Dokumentationssoftwares beziehungsweise, API-Dokumentationssoftwares gesucht. Aufgrund der hohen Masse an Technologien, wurden zufällig ausgewählte webbasierte Anwendungen sowie lokal installierte Anwendungen getestet. Bei den meisten reinen API-Dokumentationssoftwares wurde jedoch schnell klar, dass diese nicht den Anforderungen einer Low-Code-Software entsprechen. Trotzdem wurde ihre Analyse in der Technologiematrix fertiggestellt, damit diese bei diesen Punkten nicht unvollständig blieb.

7.2 Technologiematrix

7.2.1 Was ist eine Technologiematrix?

Der Begriff Technologiematrix ist in diesem Sinne eventuell etwas irreführend. Gemeint ist nicht die Inputmatrix, welche aus der mathematischen Welt bekannt ist, sondern eine eigens konzipierte. Die Art der Matrix, mit der in dem nachfolgenden Kapitel gearbeitet wird, ist eine Anforderungs-Technologie-Matrix. Bei dieser wurde geprüft, ob ausgewählte Technologien den erarbeiteten Anforderungen des letzten Kapitels entsprechen können.

7.2.2 Warum eignet sich eine Technologiematrix für diese Arbeit?

Gerade um eine große Anzahl an Dokumentationsprogrammen zu testen eignet sich eine Technologiematrix mit den Eigenschaften, wie im vorherigen Abschnitt beschrieben, optimal. Sie dient dazu einen Überblick darüber zu bekommen, welche Programme über welche Anforderungen verfügen. Aufgrund dieser Daten kann dann eine Empfehlung zu einer Technologie, welche den Anforderungen optimal entspricht, herausgearbeitet werden.

Die Technologiematrix wurde mithilfe der Anforderungen sowie der unterschiedlichen Anwendungen erstellt. Die Anwendungen wurden darauf geprüft, ob sie die jeweiligen Anfor-

derungen erfüllen können. Dabei wurden Informationen von der Website, der Dokumentation und andere Quellen des Programmes miteinbezogen. Ebenfalls wurde, falls es möglich war, bei jeder Applikation die Testversion aktiviert. Das bedeutet, die Dokumentations-Anwendung bot meistens eine "xx Tage Testversion" an. Diese wurde benutzt um nach den Anforderungsfeatures der Matrix zu suchen. Wenn eine entsprechende Funktion vorhanden war, mit der die Anforderung erfüllt werden konnte, wurde dies in der Matrix mit einem "+" gekennzeichnet. Wurden Funktionen gefunden, mit dem die Anforderung teilweise erfüllt werden konnte, wurde dies mit einem "~" gekennzeichnet. Final wurden Anforderungen, welche durch dieses Programm nicht umsetzbar waren, mit einem "-" gekennzeichnet.
Auch wurden einige der Anforderungen aus dem letzten Kapitel zusammengefasst. Zum Beispiel Anforderungen, welche durch optische Anpassungen zu erzeugen sind, wurden durch die Anforderung "optisch und strukturell vielseitig anpassbar" ersetzt.
Ebenso wurden, in Absprache mit der Firma vectorsoft AG noch ein paar Anforderungen ergänzt. Beispielsweise die Übersetzung in verschiedene Sprachen, die allgemeine Bedienung und der Kostenfaktor. Bei der Anforderung "Bedienung" konnte nicht rein objektiv gewertet werden. Um die Bedienung einer Software richtig einschätzen zu können bedarf es viel mehr Routine und Vertrautheit damit, als das es bei dieser Analyse möglich gewesen wäre. Trotzdem wurde das allgemeine Gefühl beim Testen der Software verwendet, um diese Kategorie zu bewerten. Ebenso wurde die Informationssuche miteinbezogen. Das heißt die Website und Dokumentation der Software wurde mitberücksichtigt.
Die Matrix ist in der im Anhang A.4 zu finden.

7.3 Technologieauswahl

Aus der Technologiematrix zogen zwei Anwendungen die Aufmerksamkeit auf sich. Zum einen "readme" und zum anderen "Flare". Im Nachfolgenden werden die beiden Applikationen vorgestellt sowie auf die Stärken und Schwächen dieser, und auf die geforderten Anforderungen an eine Dokumentation der Software "yeet", eingegangen.

7.3.1 "readme"

"Readme" ist ein Programm der gleichnamigen Firma ReadMe. Die Firma wurde 2014 gegründet und der Hauptsitz ist in San Francisco, US [Cra, vgl. Absatz 2]. "Readme" bietet Unternehmen die Möglichkeit, Dokumentationen zu erstellen und gleichzeitig produktive Communitys aufzubauen. Die Lösung hilft beim Aufbau von Entwicklerportalen, die Support, Tutorials, thematische Leitfäden und API-Erkundung kombinieren. Mit "readme" ist es ganz einfach, die Dokumentationen auf dem neuesten Stand zu halten, und die gemeinschaftsorientierten Funktionen fördern die Akzeptanz [Cra, vgl. Absatz 1].
Im Test überzeugte "readme" neben den vielen Anforderungen, welche die Anwendung abdeckt, besonders durch das moderne Design sowie den Community-Einbezug. Durch die Analyse der Tests in Kapitel 5 ist deutlich geworden, dass solch ein modernes Design, wie auch Ninox es hat, durchaus gewünscht ist.
Ebenfalls ein moderner Ansatz von "readme" ist, dass es eine webbasierte Anwendung ist. Dies ermöglicht die einfache Bearbeitung durch verschiedene Personen, egal an welchem

Endgerät. Es kann von überall aus aufgerufen werden, lediglich eine Verbindung zum Internet muss bestehen. Da solch eine Dokumentation jedoch meistens während der Arbeitszeit in der Firma geschrieben oder bearbeitet wird, sollte dies kein Problem darstellen.
Die Anwendung achtet sehr darauf eine gute Usability zu besitzen. Dies führt dazu, dass ein neuer Nutzer sich zwar einarbeiten muss, jedoch wird dies viel schneller erfolgen, als bei anderen Dokumentationstechnologien.
"Readme" bietet des Weiteren einige Vorlagen, welche den Einstieg in das Programm vereinfachen. So wäre zum Beispiel eine Startseite einfacher zu gestalten, da sie in ihren Grundlagen schon vorhanden ist. Dennoch ist eine individuelle Gestaltung durchaus möglich. "Readme" bietet die Möglichkeit eigene CSS, JavaScript oder HTML Codes einzubauen, um dadurch die Seiten nach individuellen Kriterien umzusetzen. Diese Features sind jedoch nur in der kostenpflichtigen Version vorhanden.
Grundlegende Eigenschaften, wie beispielsweise die Suche, sind ebenfalls schon nach den Anforderungen optimiert, welche durch die vorherigen Kapitel deutlich geworden sind. Dies alles bietet die Möglichkeit sehr schnell und einfach eine optimierte Dokumenation zu erzeugen.
Ein weiterer Vorteil von der Dokumentationsanwendung "readme" ist die eingebaute "Diskussions" - beziehungsweise "Fragen"-Seite. Die Low-Code-Plattform "yeet" besitzt zwar eine Community, jedoch noch kein richtiges Forum für Fragen. Es gibt zurzeit nur die Möglichkeit sich direkt an den Support zu wenden. Eine Alternative dazu würde die "Diskussions"-Seite in der Dokumentation bieten. Dort könnten Nutzer ihre Frage oder Anmerkung stellen und andere Nutzer oder Mitarbeiter könnten in die Diskussion einsteigen. Dies entlastet den Support und stärkt zugleich die Community. Ebenfalls kann der Umfrage aus Kapitel 5.3 entnommen werden, dass die Dokumentation allein nicht reicht, um die Fragen der Nutzer vollkommen zu beantworten. Die Behauptung "Bei einer guten Dokumentation braucht man keine Community Foren für Fragen" wurde von 80% der "Nicht-Programmierer" sowie 72% der "Programmierer" mit einem "nein" beantwortet. Bei dem Diskussionsfeature von "readme" besteht dazu die Möglichkeit Fragen oder Diskussionen als "beantwortet" zu kennzeichnen und ihr für eine bessere Übersicht einen bestimmten Tag zu vergeben, oder sie zum FAQ hinzuzufügen.
Die Anwendung bietet zudem ein Feature um die allgemeinen Statistiken der Dokumentation einzusehen. Dort werden Informationen gesammelt, wie zum Beispiel die meisten Seitenaufrufe, die häufigste Suchanfrage, etc. Dies kann sehr nützlich sein, um die Dokumentation mit diesen Informationen noch weiter den Kundenwünschen anzupassen. Der Artikel über Tabellen wird häufig gesucht? Warum dann nicht auf der Startseite darauf verweisen.
Eine andere Antwort darauf wäre ein "Rezept" dafür zu gestalten. Diese Eigenschaft gibt es bei "readme" noch nicht sehr lange und sie ist ein neuer Ansatz auf dem Markt. Ein "Rezept" soll dem Nutzer Arbeitsschritte erklären, um zu einem bestimmten Ziel zu kommen. Beispielsweise könnte ein "Rezept" für das Erstellen von Tabellen kreiert werden. Der Nutzer hätte, wie in einem richtigen Rezept, die einzelnen Arbeitsschritte nacheinander aufgelistet und kurz erklärt. Dazu wird ein Beispiel davon gezeigt. Dieser Ansatz deckt sich sehr mit der Anforderung der "Beispiele' beziehungsweise "interaktiver Beispiele", welche als "must-have"-Anforderung besteht. Bisher funktioniert dieses Feature nur mit reinen Codebeispielen, ist also für eine Menge der Low-Code-Beispiele unbrauchbar. Dennoch ist es

eine Funktion, welche "readme" bisher von seiner Konkurrenz abhebt.
Ein weiteres Feature von "readme" ist die Importierung von Github-Files. Zurzeit bestehen schon zu manchen Eigenschaften von "yeet" Teildokumentationen auf Github. Bisher nur für die Entwickler einsehbar. Durch die Importfunktion von "readme" könnten diese Informationen von "yeet" mühelos importiert werden.
"Readme" besitzt einen eingebauten Markdown Editor. Personen, welche damit umgehen können und lieber damit arbeiten, können einfach in diesen Modus wechseln.
Ein negativer Punkt von "readme" ist, dass es schnell passieren kann, dass die eigene Dokumentation einer anderen, mit "readme" erstellten Dokumentation sehr ähnlich sieht. Dies ist kein großer negativer Punkt, da es die Usability der Dokumentation nicht beeinträchtigt, trotzdem schadet es dem Branding der Marke "yeet". Wenn sich jedoch speziell um die Erstellung einer optimierten Dokumentation gekümmert wird und dort genug Zeit investiert wird, sollte es nicht passieren, dass die Dokumentation "undesignte" Elemente enthält.
Ein weiteres Problem ist das Framework "React"[1], mit dem "readme" hauptsächlich geschrieben ist. Dies könnte zu Problemen mit dem Einbinden der "yeet" Komponenten führen, da "yeet" größtenteils mit "Vue.js"[2] und "Quasar"[3] umgesetzt wurde. Dies müsste vor Beginn der Arbeit getestet werden.
Die Dokumentationssoftware "readme" bietet einen modernen webbasierten Ansatz einer Anwender-Dokumentation. Durch "readme" könnte in kürzester Zeit eine optimierte Low-Code-Dokumentation entstehen, welche nahezu alle erforschten Anforderungen abdeckt. Ein großer Pluspunkt von "readme" wäre die Möglichkeit, neben einer reinen Anwender-Dokumentation, ebenfalls andere Features wie ein Diskussionsforum oder ein FAQ zu erstellen. Dies würde helfen eine aktive Community aufzubauen.
Beispielbilder von "readme", sind innerhalb des Dokumentationskonzepts in Anhang A.5 zu finden.

7.3.2 "Flare"

Die Software "Flare" wird von der Firma MadCap Software, Inc. angeboten. MadCap Software ist eine international anerkannte Firma mit einigen verschiedenen Softwareprodukten. Die Firma wurde 2005 gegründet, ihr Hauptsitz ist in Kalifornien, US. Eine der angebotenen Anwendungen ist die Software "Flare", welche dazu dient technische Dokumentationen zu erstellen.
Ebenso "Flare" konnte in der Technologiematrix durch die Erfüllung fast aller Anforderungen hervorstechen. Mit am meisten beeindruckte im Test der Umfang der Anwendung.
Zum Gegensatz zu "readme" ist "Flare" eine Software zum herunterladen und zur lokalen Anwendung. Dadurch lässt sich die Anwendung ohne Internet bedienen und eine Abhängigkeit davon kann reduziert werden. Dennoch ist das Hosting auf einem Server kein Problem. MadCap bietet sogar von sich aus ein "Gesamtpaket" der Software "Flare" sowie einem Cloud-basiertem Content-Management-Systems an.
Bei den Tests fiel das altmodische Design häufig auf. Neben der grafischen Oberfläche,

[1] https://reactjs.org/
[2] https://vuejs.org/
[3] https://quasar.dev/

welche nicht mehr den Ansprüchen der heutigen Zeit gerecht wird, sehen ebenfalls alle von MadCap Flare bereitgestellten Themes veraltet aus. Gerade im Vergleich mit "readme" wirkt Flare nicht zeitgemäß. Natürlich lässt sich auch mit "Flare" eine moderne Dokumentation entwickeln. Der Arbeitsaufwand für solch eine Dokumentation liegt hier jedoch deutlich höher, da kein richtiges Grundgerüst vorhanden ist. Dennoch bietet "Flare" durch seine vielen Anpassungsmöglichkeiten die Möglichkeit eine einzigartige Dokumentation zu schaffen, ganz im Corporate Design von "yeet".
Ein großer Vorteil dieser Dokumentationslösung sind die vielen Features, die "Flare" enthält. So gibt es beispielsweise ähnlich wie bei "readme" die Möglichkeit eigenes CSS, JavaScript oder HTML einzubauen, Statistiken und Analysetools, Multilingualität oder die Importierung von vielen anderen Dateiformaten, wie Microsoft Word und Excel. "Flare" beinhaltet jedoch noch weitere Tools, welche für die Dokumentation von "yeet" nützlich sein könnten. Die Anwendung von MadCap ist dazu ausgelegt worden, eine Dokumentation auf mehrere Arten zu veröffentlichen zu können. Das heißt eine Umwandlung dieser in eine optimierte PDF Datei oder auch andere Formate ist kein Problem.
Die Option einen Chat in die Dokumentation einzufügen besitzt "Flare" ebenso. Dies kann für vectorsoft AG eine einfache Möglichkeit sein den Support zu entlasten und die Bindung zur Community zu stärken. Des Weiteren wirbt "Flare" mithilfe von "micro content"-Features wie Chatbots, Machine Learning und generell KI-Anwendungen zu realisieren. Solche Features in einer Dokumenation zu haben würde "yeet" eindeutig von der Konkurrenz abheben. Jedoch existiert noch keine Forschung dazu, ob diese Ansätze positive Einflüsse auf eine Dokumentation haben. Daher muss vorher durchdacht werden, ob dieser zeitliche und kostenintensive Aufwand sich lohnen würde.
Eine weitere Besonderheit von MadCap "Flare" ist die Möglichkeit interaktive Tutorials oder eLearning-Kurse zu erstellen. Diese würden gerade Anfängern helfen, die Anwendung schneller zu verstehen. Es würde die Erstellung von "Einsteiger Content" erleichtern. Ein weiterer Ansatz wäre die Anforderung "nach Leveln aufgebaut" durch dieses Feature umzusetzen. Die Meinung zu dieser Anforderung war in der Umfrage sehr unterschiedlich, sodass ein Kompromiss ein guter Ansatz wäre. Die Unterteilung in verschiedene Niveaus könnte daher nicht für die gesamte Dokumentation gelten, sondern nur in den, mithilfe von "Flare" erstellten, Tutorials angewendet werden.
Ein Nachteil, welcher unter anderem aus den vielen Features entsteht, ist die Bedienbarkeit der Anwendung selbst. Die Software ist auf den ersten Blick sehr unübersichtlich. Sich in diese einzuarbeiten ist anspruchsvoller und zeitintensiver als bei anderen.
Die Dokumentationstechnologie "Flare" der Firma MadCap bietet eine Vielzahl an Features in einem langjährig weiterentwickeltem und performanten System. Die Einarbeitung in die Software ist aufwendig, jedoch wird dies durch die unzähligen Einstellungsmöglichkeiten belohnt. Eine Dokumentation mit dieser Software zu erstellen würde länger dauern, jedoch wären alle wichtigen Anforderungen erfüllt. Eindeutige Vorteile von "Flare" sind die Freiheiten beim Designen, die simple Erzeugung eines PDF Dokumentes und die Importierung der vielen Dateiformate.
Beispielbilder von "Flare", sind innerhalb des Dokumentationskonzepts in Anhang A.5 zu finden.

Abschließend ist zu beachten, dass bei dieser Technologieauswahl nicht alle vorhandenen Dokumentationstechnologien getestet wurden. Da noch keine offiziellen "Rankings" (Einstufungen) über Dokumentationstechnologien bestehen, wurden zufällig ausgewählte Softwares in den Test miteinbezogen. Somit besteht die Chance, dass eine Technologie auf dem Markt ist mit der die geforderten Anforderungen noch besser umgesetzt werden können. Trotzdem sind die ausgewählten Technologien sehr gute Möglichkeiten mit denen die gewünschte Low-Code-Dokumentation für die Anwendung "yeet" umgesetzt werden kann. Die Wahrscheinlichkeit, dass eine noch optimiertere Technologie "readme" oder "Flare" sehr ähnlich sein wird, ist demnach sehr groß.
Die vorgestellten Dokumentationstechnologien sind beide Technologien, welche von Firmen vertrieben werden. Sie sind dadurch zwar kostenintensiver, jedoch auch effektiv und effizient zu warten. Diesen Aspekt erwähnen auch Dagenais B. und Robillard M.P in ihrer Arbeit in Kapitel 2. Ihre Forschung ergab, dass von öffentlichen Dokumentationssoftwares, aufgrund der hohen Wartungskosten und der geringeren Autorität der Dokumentation irgendwann zu einer besser kontrollierbaren Dokumentationsinfrastruktur gewechselten wurde.
Durch die kurzen Analysen der jeweiligen Anwendungen, konnten weitere positive und negative Eigenschaften in Bezug zu der Software "yeet" extrahieren werden. Jede Dokumentationstechnologie hat seine Stärken und Schwächen. Welche davon für die finale Auswahl der Technologie überwiegen, liegt letztendlich in der Entscheidung der Firma.

Kapitel 8

Finales Konzept

8.1 Ergebnisse

Die Ergebnisse dieser Arbeit lassen sich gut in einem Dokumentationskonzept zusammenfassen. Dieses ist in Anhang A.5 zu finden.
Das Konzept beinhaltet den Anleitungstyp, welcher erstellt werden soll. Dieser ist in dem Fall eine Anwender-Dokumentation. Es beinhaltet weiterhin eine Empfehlung für die Technologie sowie das Format in dem die Dokumentation umgesetzt werden soll. Diese Entscheidung wurde im vorherigen Kapitel 7 näher erläutert.
Die gesammelten Anforderungen an die Low-Code-Dokumentation aus Kapitel 6 finden sich kurz beschrieben, nach Relevanz sortiert und aufgelistet im Konzept wieder.
Dieses Konzept soll der Firma vectorsoft AG dazu dienen in naher Zukunft eine optimierte Anwender-Dokumentation für ihr neues Produkt yeet zu entwickeln. Die wichtigsten Eigenschaften für eine Low-Code-Dokumentation sind durch diese Arbeit erkenntlich geworden und somit kann sich bei der Entwicklung darauf fokussiert werden. Während der vorliegenden Arbeit wurden mit unterschiedlichen wissenschaftlichen Methoden und Analysewerkzeugen gearbeitet, dadurch liegen fundierte und aussagekräftige Ergebnisse vor.
Die Anforderungen an eine Low-Code-Dokumentationen, welche auf die Software "yeet" zugeschnitten sind - die zu Beginn der Arbeit fehlten - konnten wissenschaftlich erforscht werden. Dies erfolgte Anhand des User-Tests / Interview und der Umfrage.
Ebenso konnten Technologien empfohlen werden, welche sich sehr gut eignen die Anforderungen von "yeet" umzusetzen. Des Weiteren wurden diese ausgewählten Technologien nochmals speziell auf "yeet" bezogen analysiert, um den späteren Entscheidungsprozess zu erleichtern. Die Technologien können gewählt werden, oder nur als Beispiel für eine optimale Dokumentationstechnologie für die Software "yeet" fungieren.
Die These, dass sich Anforderungen von Low-Code-Dokumentationen und API-Dokumentationen im grundlegenden gleichen, konnte ebenso beantwortet werden. Durch die vielen Überschneidungen der Eigenschaften war dies ein eindeutiges Ergebnis.
Durch diese Arbeit wurde deutlich gemacht, wie wichtig Anwender-Dokumentationen in der heutigen Zeit sind und auf welche Eigenschaften des gesamten Produktes sie relevante Auswirkungen haben.

Kapitel 9

Zusammenfassung und Ausblick

9.1 Zusammenfassung

Die vorliegende Arbeit hatte das Ziel ein webbasiertes Dokumentationskonzept für die Low-Code-Plattform "yeet" der Firma vectorsoft AG zu erarbeiten.
Zunächst wurden dafür verwandte Arbeiten sowie der Stand der Technik zur heutigen Zeit recherchiert und analysiert. Grundlegende Anforderungen an API-Dokumentationen konnten dabei gesammelt werden und ein erster Überblick über moderne Dokumentationssysteme geschaffen werden. Die Bedeutsamkeit und Verbundenheit einer Dokumentation mit der Usability eines Produktes war dabei ein wichtiger Faktor. Es konnte die Feststellung getroffen werden, dass es noch keine spezifischen Anforderungen für die Dokumentation von Low-Code-Plattformen gibt. Derzeitige Forschungen zu diesem Thema beziehen sich lediglich auf die Anforderungen an API-Dokumentationen. Daraufhin wurde eine These aufgestellt, dass sich Anforderungen an API- und Low-Code-Dokumentationen im Grundprinzip gleichen. Diese These wurde erstellt, um die erforschten Anforderungen an API-Dokumentationen im weiteren Verlauf der Arbeit verwenden zu können. Weiterhin konnte diese These innerhalb der folgenden Kapitel bestätigt werden.
Aufgrund des Wissens aus den verwandten Arbeiten und dem Stand der Technik konnte ein individueller User-Test mit nachfolgender Umfrage konzipiert werden. Mithilfe von zwölf Testpersonen aus unterschiedlichen Zielgruppen - es wurde besonders darauf geachtet, viele mögliche Low-Code-Nutzer abzubilden - wurde der Test durchgeführt. Dabei wurden Dokumentationen anderer Low-Code-Plattformen betrachtet und nach den zuvor recherchierten Informationen ausgewertet. An dieser Stelle ist anzumerken, dass ein Großteil der Testpersonen Mitarbeiter der Firma vectorsoft waren. Daher waren diese Personen eventuell nicht gänzlich unvoreingenommen, wenn es um die Bewertung von Dokumentationen der Konkurrenz ging. Dennoch wurde auch hier darauf geachtet, eine möglichst große Diversität unter den Testteilnehmern vorhanden war. So gab es neben Entwicklern auch Marketing&Sales-Manager oder Büromanager. Dennoch wäre eine anschließende Forschung zu diesem Thema mit zufälligen Testpersonen empfehlenswert.
Die gründliche Analyse dessen sowie der Umfrage war ein weiteres wichtiges Kapitel dieser Arbeit. Dadurch konnten viele Eigenschaften herausgestellt werden. Um die Anforderungen

nach Relevanz sortieren und auswerten zu können, wurde eine eigene Methode entwickelt. Die Personas der Testpersonen wurden in Gruppen von "Programmierern" und "Nicht-Programmierern" eingeteilt. Durch Analyse der mehrfach genannten Eigenschaften konnten danach die Anforderungen selektiert und gruppiert werden.
Die "must-have"- und "nice-to-have"-Anforderungen, welche aus den Analysen hervorgingen, waren grundlegende Bestandteile des Konzeptes. Denn diese sind nun speziell als Anforderungen von Low-Code-Dokumentationen zu bezeichnen. An dieser Stelle konnte auch die vorher gestellte These größtenteils bestätigt werden, auch wenn es diverse Unterschiede zwischen Low-Code- und API-Dokumentationen gibt.
Anhand der erarbeiteten Anforderungen konnte im nächsten Schritt die Technologieauswahl stattfinden. Es wurden mehrere Dokumentationstechnologien mit einer extra dafür erstellten Matrix verglichen. Im Ergebnis waren zwei Technologien herausstechend, "readme" der Firma ReadMe und "Flare" von MadCap Software. Diese wurden anschließend kurz vorgestellt und im Hinblick auf den Nutzen als Dokumentationstechnologie für "yeet" analysiert. Die Gesamtheit der derzeit existierenden Dokumentationstechnologien, konnte in dieser Arbeit nicht analysiert werden. Die gefundenen Empfehlungen sind zwar die Technologien, welche den Anforderungen am besten entsprechen, jedoch besteht die Möglichkeit, dass auf dem Markt eine Technologie existiert, welche noch größere Übereinstimmungen aufweist. Dennoch ist zu sagen, dass durch diese Arbeit ein genereller, auf "yeet" zugeschnittener, Überblick über bestehende Technologien für Dokumentationen geschaffen werden konnte.
Abschließend konnte ein Dokumentationskonzept für die neue Low-Code-Plattform "yeet" erstellt werden. Es besteht aus allen zuvor erarbeiteten Informationen und enthält Anforderungen sowie Technologieempfehlungen speziell für "yeet" ausgelegt.

9.2 Ausblick

Der nächste Schritt für die Firma vectorsoft AG wäre die Umsetzung des Dokumentationskonzeptes. Vorher müsste erörtert werden, für welche Technologie sich letztendlich entschieden wird. Diese Arbeit, zusammen mit dem Konzept, liefern dazu die passenden Informationen und Empfehlungen. Die Entscheidung wird zusätzlich von Faktoren wie zur Verfügung stehenden finanziellen Mitteln und subjektiven Einschätzungen zu den vorgestellten Dokumentationstechnologien beeinflusst werden. Die "Mini-Prototypen", welche im Konzept zu finden sind und innerhalb der Recherche zu den Technologien entstanden, werden der Entscheidungsfindung ebenfalls weiterhelfen.
Sobald dieser Schritt gemacht wurde, hat die Firma alle nötigen Informationen zur Hand, um eine optimale und an das Produkt "yeet" angepasste, Low-Code-Dokumentation zu erstellen. Synchron zur Entwicklung der Low-Code-Plattform kann mit der Erstellung der Dokumentation begonnen werden. Dies ist ein wichtiger Schritt, um die Dokumentation als wertvolles Marketinginstrument zu sehen und nicht nur als "nötiges Übel", wie in Kapitel 2 erläutert.
Die Low-Code-Plattform "yeet" wird sich dadurch schon zu Beginn von seiner Konkurrenz abheben können und sehr weit fortgeschritten in Bezug auf die Nutzerfreundlichkeit sein.
Da während der Arbeit auffiel, dass es kaum andere wissenschaftliche Arbeiten zu dem

Thema Low-Code-Dokumentation gibt, wäre dies ein Ansatz für weitere Forschung. Anforderungsforschungen zu API-Dokumentationen gibt es bereits sehr viele, jedoch wurden keine Arbeiten zu den Anforderungen von Low-Code-Dokumentationen veröffentlicht. Die Anforderungen, welche durch diese Arbeit ausfindig gemacht wurden, haben einen sehr starken Bezug zu "yeet", beziehungsweise der Firma vectorsoft AG. Aus diesem Grund scheint es sinnvoll, weitere Untersuchungen in dieser Richtung vorzunehmen.

9.2.1 Dokumentation und Community

Während der Recherchen fiel häufig auf, dass die Dokumentation sehr oft mit der Community einer Software in Verbindung gebracht wurde. Beispielsweise wurde die Dokumentation in den User-Tests oft nicht von den Nutzern gefunden, da sie eher auf der Communityseite der Anwendung angegliedert war, statt auf der Hauptwebsite. Dies fiel im Test als negativer Punkt auf. Das neue Phänomen, dass Dokumentation und Community einer Software in Kombination gebracht werden, bedarf in Zukunft weiterer Forschung. Kann die Community noch besser in die Dokumentation miteinbezogen werden? Wie stark und in welcher Form wird in Communityseiten auf die Dokumentation verwiesen? Dies sind Fragen, die es zu beantworten gilt. Ob die Kombination von Community und Dokumentation einen positiven Einfluss auf die gesamte Dokumentation hat, muss weiter erforscht werden.
Für die Low-Code-Plattform "yeet" wird dies ebenso ein Thema sein, mit dem sich auseinandergesetzt werden muss. Gerade für "yeet" besteht die Fragestellung, ob es ausreicht, eine reine Einbindung der Community über die Dokumentation, beispielsweise mit Diskussionsforen und Artikelkommentaren, vorzunehmen - oder wird eine richtige eigene Website für die Nutzer benötigt, um eine aktive Community zu erhalten.

9.2.2 Dokumentation und KI

Um noch einen Schritt weiterzudenken, wären Chatbots eine potenzielle Erweiterung der Dokumentation. Auf einigen Websites existieren sie bereits und bieten dem Nutzer eine einfache Möglichkeit Antworten auf Problemstellungen zu erhalten.
Neben Chatbots gibt es ebenfalls richtige Chats, in denen direkt mit einem Mitarbeiter der Firma gesprochen und nach Hilfe gesucht werden kann. vectorsoft AG bietet derzeit eine Hilfshotline, zu den Geschäftszeiten sowie jederzeit die Möglichkeit, eine E-Mail an den Support zu senden. Neben der Hotline könnte es die Möglichkeit geben, einen Chat zu betreiben, in dem der Nutzer kleinere Fragen stellen kann, ohne direkt anrufen oder eine E-Mail versenden zu müssen. Konkurrenzanbieter wie Mendix[1] oder ServiceNow[2] folgen diesem Konzept schon.
Auch Chatbots sind heutzutage schon so gut entwickelt, dass sie als "First-Level-Support" dienen können. Oftmals finden sie sich auch in Kombination mit einem richtigen Chat wieder. Bei einfachen Fragen kann der Chatbot antworten, wird es komplizierter wird ein richtiger Mitarbeiter konsultiert.
Im Zusammenspiel mit Dokumentationen gibt es noch selten Chats oder Chatbots, welche

[1] https://www.mendix.com/
[2] https://www.servicenow.de/

für Fragen genutzt werden können. Es wäre ein interessanter Ansatz, zu erforschen, ob solche Chats ebenso einen Mehrwert für Dokumentationen bieten. Der Kosten-Nutzen-Faktor spielt in diesem Fall ebenfalls eine nicht zu vernachlässigende Rolle. Eine Einbindung solcher Chats oder Chatbots kann sehr aufwendig und zeitintensiv sein. Vorausgesetzt die Untersuchungen zu Chatbots in einer Dokumentation würden positive Erkenntnisse liefern, wäre es eine Innovation für die vectorsoft AG, welche sie von der Konkurrenz abheben würde.

Anhang A

Dokumente

Unter folgendem Link können alle relevanten PDF Datein abgerufen werden, auf die im Laufe dieser Arbeit verwiesen wird. Ebenso die erstellten Excel-Tabellen sowie Transkriptionen der User Tests sind dort zu finden: https://drive.google.com/drive/folders/1WQ62BCUB0d-mBFAYlzdCY8h1z7Zefuy-?usp=sharing. Alternativ befinden sich alle relevanten Dokumente noch einmal auf der hinterlegten CD.

Abbildungsverzeichnis

Glossar

GUI	Graphical User Interface, eine grafische Benutzeroberfläche, welche Software und Technik mit dem Menschen verbindet. [Rau17, vgl. Absatz 1]
CHM	Compiled HTML Help oder Compiled Help Module(s), ein Dateiformat was zu Erstellung von Hilfsdateien unter Windows benutzt wird.[Wik20, vgl. Absatz 1]
VDI	Verein Deutscher Ingenieure e.V., Herausgeber von Richtlinien zum aktuellen Stand der Technik [e.V, vgl. Absatz 1]
RESTful	Representational State Transfer, "ein Architekturstil für eine Anwendungsprogrammschnittstelle (API), die HTTP-Anfragen für den Zugriff auf und die Nutzung von Daten verwendet"[Gil20, Absatz 1, eigene Übersetzung]
SOAP-Webservices	Simple Object Access Protocol, ein Netzwerkprotokoll und Schnittstelle, über die ein Gerät den Dienst eines Servers in Anspruch nehmen kann [ion19, vgl. Absatz 1 + 2]
Messaging-API	Eine Programmierschnittstelle, die es einer Anwendung ermöglicht, Nachrichten und angehängte Dateien über ein E-Mail- oder Textnachrichtensystem zu senden und zu empfangen [pcm, vgl. Absatz 1]
SPI	Service Provider Interface, ein Reihe von Schnittstellen oder abstrakten Klassen, die ein Dienst definiert. Sie stellt die Klassen und Methoden dar, die für Ihre Anwendung verfügbar sind.[BES, vgl. Absatz 4]
KMU	Kurzform von "Kleine und mittlere Unternehmen"
RAD	Rapid Application Development, eine Methode, die sich auf eine schnelle Entwicklung durch häufige Iterationen und kontinuierliches Feedback konzentriert [Chi20, vgl. Absatz 1]
WYSIWYG	"What You See Is What You Get", stellt fest, wie die Inhalte innerhalb eines Editors dargestellt werden, eine "Echtbilddarstellung"[Cha, Absatz 1]
SaaS	Software-as-a-Service, "ist eine Cloud-basierte Methode zur Bereitstellung von Software für Nutzer"[clo, Absatz 1]
IDE	Integrated Development Environment (Integrierte Entwicklungsumgebung), eine Sammlung von Tools für das entwickeln von Software[Lub17, vgl. Absatz 1]
DSDM	Dynamic Systems Development Method, eine agile Methode, welche sich auf den gesamten Projektlebenszyklus bezieht [Cona, vgl. Absatz 1]

Literaturverzeichnis

[AG] AG vectorsoft: *Vectorsoft - Über Uns*. Online: https://www.vectorsoft.de/unternehmen/ueber-uns, . – Aufgerufen am: 2021-10-07

[AJ18] Ashby, Dennis ; Jensen, Claus T.: *APIs For Dummies®, 3rd IBM Limited Edition*. Bd. 3. Auflage. John Wiley & Sons, Inc., 2018 https://www.ibm.com/downloads/cas/GJ5QVQ7X. – ISBN 9781119457138

[App] Appian: *Startseite - Appian Website*. Online: https://appian.com/, . – Aufgerufen am: 2021-09-20

[BES] Ben Elhaj Slimene, Abdelbaki: *Implementing Plugins with Java's Service Provider Interface*. Online: https://reflectoring.io/service-provider-interface/, . – Aufgerufen am: 2021-10-11

[Bie06] Biesterfeldt, Jakob: Usability von technischer Dokumentation. In: *Tagungsband UP06* (2006)

[BK21] Bratincevic, John ; Koplowitz, Rob: The Forrester Wave™: Low-Code Development Platforms For Professional Developers, Q2 2021. (2021)

[BM07] Birn, Lukas ; Müller, Claudia: Steigerung des Kundennutzens durch eine kollaborativ erstellte Anwenderdokumentation. (2007)

[Bra16] Braun, Michael: Nicht-funktionale Anforderungen. In: *Juristisches IT-Projektmanagement Lehrstuhl für Programmierung und Softwaretechnik Ludwig-Maximilians-Universität München* (2016)

[Cas16] Castrillejo, Victoria: *Was ist ein Wiki?* Online: https://wikis.fu-berlin.de/pages/viewpage.action?pageId=402489828, 2016. – Aufgerufen am: 2021-10-10

[Cha] Chananewitz, Arne: *WYSIWYG*. Online: https://www.loewenstark.com/wissen/wysiwyg/, . – Aufgerufen am: 2021-10-11

[Chi20] Chien, Christine: *What is Rapid Application Development (RAD)?* Online: https://codebots.com/app-development/what-is-rapid-application-development-rad, 2020. – Aufgerufen am: 2021-10-11

[clo] cloudflare.com: *Was ist Software-as-a-Service (SaaS)?* Online: https://www.cloudflare.com/de-de/learning/cloud/what-is-saas/, . – Aufgerufen am: 2021-10-11

[Cona] Consortium, Agile B.: *What is DSDM?* Online: https://www.agilebusiness.org/page/whatisdsdm, . – Aufgerufen am: 2021-10-11

[Conb] Consulting, Coley: *MoSCoW Prioritisation*. Online: https://www.coleyconsulting.co.uk/moscow.htm, . – Aufgerufen am: 2021-09-27

[Cra] Craft: *ReadMe*. Online: https://craft.co/readmeio, . – Aufgerufen am: 2021-10-03

[DR10] Dagenais, Barthélémy ; Robillard, Martin P.: Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors. In: *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*. New York, NY, USA : Association for Computing Machinery, 2010 (FSE '10). – ISBN 9781605587912, 127–136

[e.V] e.V., VDI: *VDI-Richtlinien: Standards setzen – auf dem aktuellen Stand der Technik*. Online: https://www.vdi.de/richtlinien, . – Aufgerufen am: 2021-10-11

[Fir] Fireball, Daring: *Markdown: Main*. Online: https://daringfireball.net/projects/markdown/syntax, . – Aufgerufen am: 2021-10-05

[For] Forrester: *Forrester - Company*. Online: https://investor.forrester.com/company, . – Aufgerufen am: 2021-09-20

[Gil20] Gillis, Alexander S.: *REST API (RESTful API)*. Online: https://searchapparchitecture.techtarget.com/definition/RESTful-API, 2020. – Aufgerufen am: 2021-10-11

[Git] Github: *Where the world builds software*. Online: https://github.com/about, . – Aufgerufen am: 2021-10-03

[Goo] Google: *Google Formulare verwenden*. Online: https://support.google.com/docs/answer/6281888?visit_id=637679927877883420-603265873&rd=1, . – Aufgerufen am: 2021-09-23

[Hof17] Hofmann, Maria: *Die Erhebungsmethode des Lauten Denkens QUASUS. Qualitatives Methodenportal zur Qualitativen Sozial-, Unterrichts- und Schulforschung*. Online: https://quasus.ph-freiburg.de/2217-2/, 2017. – Aufgerufen am: 2021-09-16

[HZH19] Huesmann, Rolf ; Zeier, Alexander ; Heinemann, Andreas: Eigenschaften optimierter API-Dokumentationen im Entwicklungsprozess sicherer Software. In: *Mensch und Computer 2019 - Workshopband*. Bonn : Gesellschaft für Informatik e.V., 2019

[IJRJ18] Inzunza, Sergio ; Juárez-Ramírez, Reyes ; Jiménez, Samantha: API Documentation. In: Rocha, Álvaro (Hrsg.) ; Adeli, Hojjat (Hrsg.) ; Reis, Luís P. (Hrsg.) ; Costanzo, Sandra (Hrsg.): *Trends and Advances in Information Systems and Technologies*. Cham : Springer International Publishing, 2018. – ISBN 978–3–319–77712–2, S. 229–239

[ion19] ionos.de: *SOAP: Das Netzwerkprotokoll erklärt*. Online: https://www.ionos.de/digitalguide/websites/web-entwicklung/soap-simple-object-access-protocol/, 2019. – Aufgerufen am: 2021-10-11

[Kon19] Konrad, Alex: *Meet Matt Calkins: Billionaire, Board Game God And Tech's Hidden Disruptor Forbes Magazine*. Online: https://www.forbes.com/sites/alexkonrad/2019/04/29/meet-matt-calkins-billionaire-board-game-god-and-techs-hidden-disruptor/#2ae5b5106e77, 2019. – Aufgerufen am: 2021-09-20

[Kot11] Kothes, Lars: *Grundlagen der Technischen Dokumentation - Anleitungen verständlich und normgerecht erstellen*. Springer, 2011

[Kuc18] Kuckartz, Udo: *Qualitative Inhaltsanalyse. Methoden, Praxis, Computerunterstützung - 4. Auflage*. Beltz Juventa, 2018

[Lub17] Luber, Stephan Stefan und A. Stefan und Augsten: *Was ist eine IDE?* Online: https://www.dev-insider.de/was-ist-eine-ide-a-600703/, 2017. – Aufgerufen am: 2021-10-11

[Mar] Markdown: *Markdown: Syntax*. Online: https://markdown.de/, . – Aufgerufen am: 2021-10-05

[Mic] Microsoft: *Facts About Microsoft*. Online: https://news.microsoft.com/facts-about-microsoft/, . – Aufgerufen am: 2021-09-20

[Mor19] Moran, Kate: *Usability Testing 101*. Online: https://www.nngroup.com/articles/usability-testing-101/, 2019. – Aufgerufen am: 2021-10-09

[Mor21] Moran, Kate: *How to Test Content with Users*. Online: https://www.nngroup.com/articles/testing-content-websites/, 2021. – Aufgerufen am: 2021-10-10

[Nin] Ninox: *Startseite - Ninox Website*. Online: https://ninox.com/de, . – Aufgerufen am: 2021-09-18

[NL93] Nielsen, Jakob ; Landauer, Thomas K.: A Mathematical Model of the Finding of Usability Problems. New York, NY, USA : Association for Computing Machinery, 1993 (CHI '93). – ISBN 0897915755, 206–213

[NS15] Ndanu, Mwaniki C. ; Syombua, Mue J.: Mixed methods research: The hidden cracks of the triangulation. In: *Name: General Education Journal* 4 (2015), Nr. 2

[OBS] OBS: *OBS Github - Contributors*. Online: https://github.com/obsproject/obs-studio/graphs/contributors, . – Aufgerufen am: 2021-09-20

[pcm] pcmag.com: *messaging API*. Online: https://www.pcmag.com/encyclopedia/term/messaging-api, . – Aufgerufen am: 2021-10-11

[Pfe20] Pfeiffer, Lena Franziska und G. Franziska und Genau: *Umfrage als wissenschaftliche Methode für die Bachelorarbeit*. Online: https://www.scribbr.de/methodik/umfrage-wissenschaftliche-methode/, 2020. – Aufgerufen am: 2021-10-09

[Rau17] Rauch, Gedeon: *Was ist eine GUI?* Online: https://www.dev-insider.de/was-ist-eine-gui-a-651868/, 2017. – Aufgerufen am: 2021-10-11

[RD11] Robillard, M.P. ; DeLine, R.: A field study of API learning obstacles. In: *Empir Software Eng 16*, 2011, 703–732

[RK19] Rymer, John R. ; Koplowitz, Rob: The Forrester Wave™: Low-Code Development Platforms For AD&D Professionals, Q1 2019. (2019)

[RR16] Richardson, Clay ; Rymer, John R.: Vendor Landscape: The Fractured, Fertile Terrain Of Low-Code Application Platforms. (2016)

[SC98] Strauss, Anselm ; Corbin, Juliet: *Basics of qualitative research techniques*. Citeseer, 1998

[Sch20] Schenk, Philip: *Low-Code vs. No-Code - Für Unternehmen*. Online: https://www.02100.io/ratgeber/low-code-vs-no-code, 2020. – Aufgerufen am: 2021-10-07

[SIDRP20] Sahay, Apurvanand ; Indamutsa, Arsene ; Di Ruscio, Davide ; Pierantonio, Alfonso: Supporting the understanding and comparison of low-code development platforms. In: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2020, S. 171–178

[Sie14] Siemens: *Anwenderdefinierte Dokumentation bereitstellen - Funktionshandbuch A5E33344700-AA*. Online: https://www.spshaus.ch/files/inc/Downloads/Lernumgebung/Kurse/TIA-PRO3/providing_user-defined_documentation_dede.pdf, 2014. – Aufgerufen am: 2021-10-07

[Sim] SimplifierAG: *Über uns - Simplifier Website*. Online: https://simplifier.io/ueber-uns/, . – Aufgerufen am: 2021-09-18

[Soc04] Society, IEEE C.: *SWEBOK Guide to the Software Engineering Body of Knowledge*. Erhältlich über: https://craft.co/readmeio, 2004

[Spi17] Spichale, Kai: *API-Design : Praxishandbuch für Java- und Webservice-Entwickler.*. Bd. 1. Auflage. dpunkt.verlag, 2017 https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=1620178&site=ehost-live. – ISBN 9783864903878

[Sta97] Stapleton, Jennifer: *DSDM, dynamic systems development method: the method in practice*. Cambridge University Press, 1997

[Vö12] Völzke, Katja: *Lautes Denken bei kompetenzorientierten Diagnoseaufgaben zur naturwissenschaftlichen Erkenntnisgewinnung*. Kassel Univ. Press, 2012

[Wik20] Wikipedia: *CHM (Dateiformat)*. Online: https://de.wikipedia.org/wiki/CHM_(Dateiformat), 2020. – Aufgerufen am: 2021-10-11

[Win00] Winter, Stefanie: *Qualitatives Interview*. Online: http://nosnos.synology.me/MethodenlisteUniKarlsruhe/imihome.imi.uni-karlsruhe.de/nqualitatives_interview_b.html, 2000. – Aufgerufen am: 2021-09-27

[WSJSS13] Watson, Robert ; Stamnes, Mark ; Jeannot-Schroeder, Jacob ; Spyridakis, Jan H.: API Documentation and Software Community Values: A Survey of Open-Source API Documentation. New York, NY, USA : Association for Computing Machinery, 2013 (SIGDOC '13). – ISBN 9781450321310, S. 165–174

[Wü] Wübbenhorst, Klaus: *Likert-Skalierung*. Online: https://wirtschaftslexikon.gabler.de/definition/likert-skalierung-40986, . – Aufgerufen am: 2021-10-10

[Zooa] *Annual Report Zoom Video Communications 2019 - Form 10-K*. Online: https://www.sec.gov/ix?doc=/Archives/edgar/data/1585521/000158552120000095/zm-20200131.htm, . – Aufgerufen am: 2021-09-20

[Zoob] Zoom: *Lokale Aufzeichnung - Zoom Website*. Online: https://support.zoom.us/hc/de/articles/201362473-Lokale-Aufzeichnung, . – Aufgerufen am: 2021-09-20