FOM Hochschule für Oekonomie & Management
Hochschulzentrum Düsseldorf

Seminararbeit

im Studiengang
Wirtschaftsinformatik - Business Information Systems
Bachelor of Science (B.Sc.)

zum Thema

Entwicklung eines widget-basierten Dashboards

verfasst von

Nassim Doukkali Attar, Jan Bendel & Frederik Reiff

Erster Prüfer/in	Olga Koch
Matrikelnummer	547484, 555541 & 544582
Abgabedatum	28-02-2023

Inhaltsverzeichnis

Abkürzungsverzeichnis **III**

Abbildungsverzeichnis **4**

1 Einleitung **5**
1.1 Motivation . . . 5
1.2 Zielsetzung . . . 5
1.3 Vorgehensweise . . . 6

2 Grundlagen **7**
2.1 Projekte und Projektmanagement . . . 7
2.1.1 Personas . . . 8
2.1.2 User Stories . . . 9
2.1.3 Kanban . . . 10
2.2 Technologien . . . 11
2.2.1 Rollenbasierte Zugriffskontrolle . . . 11
2.2.2 API . . . 11
2.2.3 JS Frontend Framework . . . 12
2.2.4 Vue.js und Nuxt.js . . . 13

3 Projektplanung **15**
3.1 Zielsetzung und Umfang . . . 15
3.2 Methodik . . . 16
3.2.1 Zielgruppe . . . 18
3.2.2 Personas . . . 19
3.3 Konzeption und Entwurf . . . 21
3.3.1 Prototyping . . . 21
3.3.2 Navigationskonzept . . . 22
3.3.3 Screen- und Interactiondesign . . . 25

4 Umsetzung **26**
4.1 Umsetzung: Frontend . . . 27
4.1.1 Erster Entwurf und Hierarchiekonzept . . . 27
4.1.2 Widgets . . . 30
4.1.3 Daten Management und Store . . . 33
4.1.4 Token-Authentifizierung . . . 37
4.1.5 Einstellungen und Themes . . . 40
4.2 Umsetzung: Backend . . . 43
4.3 Automatisches Deployment mit Github Actions . . . 49

5 Reflexion und Fazit **55**

Literatur- und Quellenverzeichnis **XVII**

Abkürzungsverzeichnis

API ..... Application Programming Interface

CSS ..... Cascading Style Sheets

DOM ..... Document Object Model

FTE ..... Full Time Equivalent

HTML ..... Hypertext Markup Language

ID ..... Identfikiation, Identfikitationsnummer

IT ..... Information Technology

JS ..... Javascript

JSON ..... Javascript Object Notation

RBAC ..... Role-Based Access Control/ Rollenbasierte Zugriffskontrolle

REST ..... Representational State Transfer

SEO ..... Search Engine Optimisation/ Suchmaschinenoptimierung

UX ..... User Experience

Abbildungsverzeichnis

1 Trello Board 17
2 Zielgruppe 18
3 Persona Anna 19
4 Persona Andi 20
5 Prototyp 22
6 Navigationsleiste 24
7 Screendesign 25
8 Erster Layoutentwurf für das Dashboard 28
9 Hierarchiekonzept 30
10 Unterschiedlicher Inhalt bei gleichem Design 31
11 Vorhandener Token im lokalen Speicher des Browsers 38
12 Relationaler Datenbankentwurf 44
13 Ablauf des Frontend Deployments 49
14 Ablauf des Backend Deployments 52

1 Einleitung

1.1 Motivation

Laut Statista nutzen bereits circa 90 Prozent der Europäer und Europäerinnen das Internet täglich, daher ist es unumstritten, das dies eine enorme Relevanz auf den Alltag vieler Menschen hat.[1] Für den Großteil der Menschen ist der Browser einer der wesentlichsten Mittel, um sich durch das Internet zu bewegen und sich Informationen zu beschaffen. Dennoch ist das Internet in den letzten Jahren sehr stark gewachsen, weshalb das Stichwort Organisation an Relevanz gewinnt. Hier setzt das Projektteam an, die Vision des Teams ist es, die Organisation und Effizienz bei der Navigation durch das Internet zu verbessern. Um dies zu erreichen, hat das Team sich dazu entschieden ein Dashboard, welches sich immer bei Browserstart öffnet, zu bauen. Auf diesem sollen alle relevanten Informationen in Form von Widgets sofort erkennbar sein. Hierzu gehören vor allem das Wetter, eine persönliche Linksammlung und aktuelle Nachrichten. Mit dieser Lösung erhofft sich das Team einen positiven Mehrwert für Menschen zu schaffen, in dem diese einen wesentlich verbesserten Einstieg in den Browser und somit in das Internet erhalten.

1.2 Zielsetzung

Das größte Ziel des Projektteams ist natürlich die in der Motivation bereits genannte Vision. Damit diese Vision erfolgreich erreicht werden kann, benötigt es konkrete und zeitlich terminierte Ziele, damit ein Projekterfolg gewährleistet werden kann und das Produkt, den Nutzenden, so schnell wie möglich Mehrwert liefern kann. Diese Unterziele sollen alle der SMART- Methodik entsprechen. Eremit Britta und Kai F Weber definieren SMART in ihrem Buch, Individuelle Persönlichkeitsentwicklung: Growing by Transformation , wie folgt. „S.M.A.R.T. ist ein Akronym und steht für Specific Measurable Accepted Realistic Timely“.[2] Zu Deutsch sollten die Ziele des Projektteams also spezifisch, messbar, akzeptiert und realistisch sein. Das Projektteam hat in einem gemeinsamen Workshop drei Ziele erarbeitet und gemäß der SMART-Methodik schriftlich festgehalten, damit diese während des Projekts als Leitplanken dienen und nach Abschluss für eine Reflexion genutzt werden können. Folgende Ziele wurden schriftlich festgehalten:

[1] vgl. Rabe, 2023.
[2] vgl. Eremit et al., 2016, S. 95.

1. Zum Projektabschluss, welcher am 6. Februar 2023 ist, soll ein Produkt vorhanden sein, welches alle benötigten Features und technischen Implementationen hat, damit es als ganzheitliches Dashboard bei dem Browserstart genutzt werden kann. Dies beinhaltet eine funktionierende Nutzerverwaltung, Rollenverwaltung, Todo- und Quicklinkverwaltung, sowie eine optisch ansprechende Benutzeroberfläche.

2. Das Projekt soll bis zum 31. Dezember 2022 komplett durch die strategische Planungsphase durch sein, mit dem Ergebnis, das konkrete Arbeitspakete vorhanden sind und diese unter der Gruppe verteilt wurden. In der Planungsphase soll jedes Mitglied der Gruppe in gleichermaßen an den Dokumenten mitgearbeitet haben, so dass der gesamte Input des Teams in die Konzeptionierung einfließt.

3. Während der Umsetzung wirken alle Mitglieder an der Implementierung und der Erstellung der Projektdokumentation mit, so dass bis zum 28. Februar 2023 sowohl das Projekt von Implementierungssicht abgeschlossen wurde, als auch alle Formen der Dokumentation vorhanden sind.

1.3 Vorgehensweise

Diese Projektarbeit ist chronologisch so geschrieben, dass Sie dem zeitlichen Verlauf des Projektes gleicht. Das Projekt und die damit einhergehende Arbeit sind in vier wesentliche Bereiche aufgeteilt, zunächst die Grundlagen, gefolgt von der Projektplanung und der Umsetzung, sowie abschließend der Reflexion des Projektes. Im Bereich der Grundlagen wird alles behandelt, was sowohl aus Sicht des Projektmanagements, sowie der Umsetzung relevant ist. Diese Inhalte waren für das Team bereits in der Vorbereitung des Projekts essentiell, um eine ordentliche Planung durchführen zu können. Mit Hilfe der Grundlagen kann das Team nun in die Projektplanung einsteigen, hier wird sowohl die methodische Planung, als auch die Ausrichtung des Projekts in Bezug auf Zielgruppen, Designfragen und Prototypen herausgestellt. Nach der Planung geht das Projekt in die Umsetzung über, welche durch das nach der Projektplanung folgende Kapitel abgebildet wird. Hier wird vor allem das Frontend betrachtet, damit der Bezug zu der strategischen Planung gut aufgearbeitet werden kann. Neben dem Frontend, wird aber auch das Backend und das automatische Deployment technisch betrachtet und die Implementierung an Hand von eigenem Quelltext erläutert. Um den Erfolg des Projektes zu bemessen wird in einer abschließenden Reflexion evaluiert, welche Aspekte besonders positiv

waren und welche Aspekte in Zukunft verbessert werden können. Abschließend wird das Projekt und dessen Planung, sowie Dokumentation mit einem Fazit vollendet.

2 Grundlagen

Die Planung und Umsetzung des Projektes erfordert einige Methoden des Projektmanagements, sowie technische Frameworks. Diese werden im Rahmen der Grundlagen definiert und erläutert, um ein klares Verständnis der genutzten Methodiken und der technischen Umsetzung zu schaffen. Zu welchem Zweck welche Methodiken und Technologien genutzt wurden, wird in Kapitel drei und Kapitel vier näher betrachtet.

2.1 Projekte und Projektmanagement

Ein Projekt ist laut DIN 69901, eine einzigartige und einmalige Unternehmung, die durch einen zeitlichen und finanziellen Rahmen gedeckelt ist.[3] Ein solches Projekt muss ausreichend geplant, geführt und kontrolliert sein. Nach selbiger DIN-Norm ist das Projektmanagement definiert als „Die Gesamtheit von Führungsaufgaben, -organisation, -techniken und -mitteln für die Initialisierung, Definition, Planung, Steuerung und den Abschluss von Projekten“.[4]

Das Projektmanagement wird in der Praxis durch Methoden beziehungsweise Frameworks unterstützt, welche Projektrollen und -ablauf definieren. Beispiele für solche Frameworks sind SCRUM, Wasserfall, Extreme Programming, Six Sigma und viele weitere.[5]

Ein typischer Projektablauf, laut DIN 69901, in der gröbsten Form sieht wie folgt aus, es existieren jeweils Phasen, welche eine Initialisierung, Definition, Planung, Steuerung und Abschluss abbilden. Jedoch muss beachtet werden, dass diese Schritte nicht iterativ durchgearbeitet werden müssen. Teilweise können Phasen zumindest teilweise parallel laufen und auch Iterationen in vorherige, bereits abgeschlossene Phasen sind durchaus möglich.[6] Das Project Management Institute kennt die folgenden 10 Managementbereiche[7]:

[3] vgl. Weber, 2021, S. 4.
[4] Weber, 2021, S. 4.
[5] vgl. Weber, 2021, S. 4.
[6] vgl. Weber, 2021, S. 23.
[7] vgl. Morris, 2000, Exhibit 6.

1. Integrierende Aufgaben oder auch Integrationsmanagement
2. Zeitmanagement
3. Personalmanagement
4. Qualitätsmanagement
5. Umfangsmanagement
6. Kostenmanagement
7. Stakeholdermanagement
8. Risikomanagement
9. Beschaffungsmanagement
10. Kommunikationsmanagement

2.1.1 Personas

In vielen Softwareprojekten wird heutzutage der Fokus auf den Anwendenden gelegt, da dieser logischerweise der Dreh- und Angelpunkt einer jeden Software ist, welche auf eine Benutzung durch Menschen ausgelegt ist.[8]

Aus dieser Überlegung heraus ergeben sich die Fragen, wer die potentiellen Anwender sind und was für diese wichtig ist? Um diese und weitere Fragen zu beantworten, lässt sich die nach Alan Cooper in 1990 formulierte Methode der Personas nutzen.[9]

Ein Persona ist eine fiktive Person, welche definiert wird, um die Bedürfnisse einer reellen Person oder einer Zielgruppe abzubilden. Der große Vorteil der Methode ist, dass der Fokus von den Programmierenden und der technischen Umsetzung weg und mehr in Richtung der Nutzenden verschoben wird.[10]

Personas werden in der Regel mit einigen Informationen, wie zum Beispiel dem Alter, dem Geschlecht, dem Beruf, der technischen Kompetenz, der Persönlichkeit, der Interessen und dem Bezug zu dem aktuellen Softwareprodukt beschrieben.[11]

[8] vgl. Holt et al., 2011, S. 40.
[9] vgl. Holt et al., 2011, S. 40.
[10] vgl. Schweibenz, 2004, S. 152.
[11] vgl. Schweibenz, 2004, S. 155.

Der Detailgrad einer Persona lässt sich jedoch, je nach Notwendigkeit für das aktuelle Vorhaben, anpassen.[12]

Personas werden oft in der agilen Softwareentwicklung, besonders im SCRUM-Modell genutzt, da sie gut geeignet sind, um sogenannte User Stories anzureichern.[13]

2.1.2 User Stories

User Stories sind eine, im agilen Umfeld, oft genutzte Methodik um Funktionalitäten zu schaffen, welche dem Nutzenden oder dem Auftragnehmenden einen Mehrwert bieten.

Eine übliche User Story besteht, laut Ron Jeffries, einem bekannten US amerikanischem Informatiker, aus 3 Teilen, einer Art Karteikarte, einem Gespräch und einer Bestätigung.

Die Karteikarte soll die Funktion einer groben initialen Beschreibung bieten, sie ist natürlich sehr sichtbar, da sie die gesamte User Story repräsentiert, jedoch ist sie nicht sehr detailreich.

Der Detailgrad der User Story wird durch das Gespräch gewährleistet. Gespräch, zu Englisch Conversation, bedeutet in diesem Fall, dass die zuvor definierte User Story durch einen gezielten Austausch, oft zwischen dem Kunden und dem Entwicklerteam, präzisiert werden.[14] Die User Story wird gezielt besprochen und kritische Rückfragen sind nicht nur erwünscht sondern notwendig. Wichtig ist hierbei zu beachten, dass User Stories und die besprochenen Inhalte des Gespräches nicht rechtlich bindend sind.[15]

Damit die Inhalte der Conversation nicht verloren gehen, muss eine Bestätigung, zu Englisch Confirmation, erfolgen. Der Sinn dieser ist es, die Nachfragen und besprochenen Themen durch Tests festzuhalten. Akzeptanzkriterien zu den jeweiligen User Stories werden ebenfalls in der Bestätigung festgelegt.[16]

Der Umfang der User Story ist durch die vorhergehenden drei Bestandteile gesichert. Jedoch ergibt sich ein weiteres Problem, nämlich der Umsetzungsaufwand. Es wäre fatal, wenn nicht bewusst ist, wie aufwendig die Umsetzung einzelner User Stories

[12] vgl. Holt et al., 2011, S. 40.
[13] vgl. Holt et al., 2011, S. 43.
[14] vgl. Cohn und Beck, 2019, S. 26.
[15] vgl. Cohn und Beck, 2019, S. 29.
[16] vgl. Cohn und Beck, 2019, S. 26.

wird. Denn auch wenn User Stories rein funktional sind, spielt der Aufwand in der Umsetzung, logischerweise, immer noch eine wichtige Rolle.

In agilen Modellen, wie zum Beispiel SCRUM, werden Story Points zur Aufwandsschätzung genutzt. Ein Story Point beschreibt hierbei nur die Gewichtung der User Stories, anhand des Aufwandes. Wichtig dabei ist, dass ein Story Point und somit auch die User Story selbst, meist nur in relativen Maßen gemessen werden können. Konkret bedeutet dies, dass Story Points die User Stories untereinander vergleichbar machen, jedoch ein Story Point nicht sinnvoll mit einer Zeitspanne oder FTE-Stunden zu bewerten ist.[17]

2.1.3 Kanban

Kanban ist eine Methodik mit japanischem Ursprung.[18] Der Gedanke derer es sein soll eine klare Übersicht über alle offenen Aufgaben zu verschaffen. Kanban nimmt oft die Gestalt einer Aufgabenleiste mit mehreren Spuren an. Oft zum Beispiel Backlog, In Bearbeitung, Fertig, in der gröbsten Form. Kanban zeichnet sich durch vier Charakteristika aus.[19]

1. Aufgaben werden nicht vom Management zugewiesen. Der einzelne Mitarbeitende entnimmt offene Aufgaben direkt aus dem Backlog heraus. Dieses System wird auch als Pull bezeichnet.
2. Es gibt ein Höchstlimit an gleichzeitig aktiv bearbeiteten Aufgaben.
3. Alle Aufgaben sind öffentlich durch alle Projektteilnehmenden einsehbar
4. Kontinuierliche Verbesserung wird praktiziert.

Hierbei ist wichtig zu nennen, dass Kanban, zum Teil auf Grund der oben genannten Eigenschaften, als ein agiles Vorgehensmodell gesehen wird. Gründe dafür sind auch, dass es keine klare Phasentrennung, wie zum Beispiel bei dem Wasserfallmodell gibt.[20]

[17] vgl. Bartonitz, 2018, S. 106.
[18] vgl. Epping, 2011, S. 23.
[19] vgl. Epping, 2011, S. 54f.
[20] vgl. Epping, 2011, S. 67.

2.2 Technologien

Damit ein technisch umzusetzendes Projekt auch in der Implementierung gelingt, ist es von hoher Relevanz, die richtigen Technologien einzubinden. Das Team hat genutzte Technologien mit hoher Sorgfalt im Vorhinein ausgewählt und sich hierfür konkreter mit deren Funktionsweise beschäftigt. Diese Technologien werden im Folgenden näher erläutert, da sie einen wichtigen Teil in dem Kapitel bezüglich der Umsetzung spielen.

2.2.1 Rollenbasierte Zugriffskontrolle

Im Bereich der serverseitigen Webapplikationen muss aus sicherheitsperspektivischen Gründen eine Zugriffskontrolle eingerichtet werden. „Role-Based Access Control“ (zu Deutsch: Rollenbasierte Zugriffskontrolle) ist einer der Mechanismen zur Steuerung von Berechtigungen für Nutzer und Nutzerinnen einer Web-Applikation.[21]

RBAC generalisiert in zwei Objekte, die Nutzer-Objekte und die Rollen-Objekte. Zwischen diesen beiden Objekten werden nun Beziehungen erstellt, hierbei ist es jedoch wichtig zu erwähnen, dass keine Beziehungen untereinander, also zum Beispiel Rolle zu Rolle, erstellt werden, sondern lediglich Nutzer zu Rolle Beziehungen. Das Konzept erlaubt es einem Nutzenden mehrere Rollen zu geben, aber auch eine Rolle mehreren Nutzenden zu geben, daher basiert das RBAC Konzept auf einer m:n, „Many to Many“ Beziehung.[22]

Durch den Einsatz von RBAC erhält die Systemlandschaft besonders unter Sicherheitsaspekten einen Vorteil. Der Zugriff auf Ressourcen ist mit klar definierten Vektoren beschrieben, die Rechteverteilung ist transparent, flexibel und leicht administrierbar. Besonders beim Einsatz im Unternehmensumfeld zeigen sich die Vorteile im Bereich von der Position und der damit einhergehenden Rolle eines Mitarbeitenden und den zu gewährenden Befugnissen.[23]

2.2.2 API

Application Programming Interfaces, sind eine, in der Anwendungsentwicklung, wichtige Komponente, die bei der Kommunikation von logisch oder gar physisch

[21] vgl. Harnal und Chauhan, 2020, p. 2.
[22] vgl. Wang et al., 2005, p. 427.
[23] vgl. Harnal und Chauhan, 2020, p. 2.

getrennten Systemen zum Einsatz kommen. Hierbei sind APIs eine Schnittstelle, die von außen durch Clients oder Server von zum Beispiel Geschäftspartnern, Externen oder der Allgemeinheit genutzt werden können. Durch diese APIs können dahinterliegende Daten durch kontrollierte Wege erreichbar und änderbar gemacht werden. Dabei ist es für den Nutzer oder die Nutzerin einer API nicht wichtig, wie die Schnittstellen quelltextseitig umgesetzt wurden, sondern lediglich, welche Informationen mitgegeben werden müssen, und welche Art von Antwort erwartet werden kann.[24]

Eine Unterform von APIs sind die sogenannten REST APIs, Representational State Transfer APIs. Diese erfüllen zusätzlich eine Reihe vom bestimmten Architekturanforderungen, die Schnittstellen häufig in der Praxis nutzen.[25] Roy Fielding legte hierfür die Grundsteine mit seinem sogenannten 'Pipe and Filter' Architekturtypen. Bereits in seinem Werk aus dem Jahr 1999 brachte Herr Fielding wesentliche Punkte vor, die bis heute in der Praxis maßgeblich Anwendung finden. Im Folgenden sind 3 Beispiele der Anforderungen an Schnittstellen gelistet, die einen wesentlichen Teil der Abgrenzung darstellen:

1. Client-Server: Ein Server stellt hier einen Service bereit und wartet auf Anfragen von einem Client. Sollte eine Anfrage eintreffen so beginnt der Server seine Arbeit und liefert eine Antwort oder einen Fehler.[26]

2. Layered System: Um Aufgaben logisch und strukturell zu trennen, muss die Schnittstelle mehrere Schichten, wie Load Balancer, Caching, und Proxies unterstützen.[27]

3. Client-Stateless-Server: Die Schnittstelle darf keine Sitzungen nutzen, die Anfrage muss alle Informationen mitgeben und darf somit keinen Kontext aus vorherigen Anfragen, wie bei einer Sitzung, nutzen. Gleiches gilt für Caching-Komponenten.[28]

2.2.3 JS Frontend Framework

Bei der Entwicklung größerer Webseiten oder Webanwendungen kann ein Entwickelnder auf wiederkehrende Herausforderungen, besonders im Bezug auf die Benutzeroberfläche, treffen. Des Weiteren ist ein Entwickelnder in der Regel mit der

[24] vgl. IBM, 2022.
[25] vgl. Red-Hat, 2022.
[26] vgl. Fielding, 1999, S. 14f.
[27] vgl. Fielding, 1999, S. 15.
[28] vgl. Fielding, 1999, S. 15f.

Architektur konfrontiert, die oft angepasst werden muss, damit neue Features während der Entwicklungsphase mit integriert werden können.[29] Diese Probleme können durch den Einsatz eines Javascript Frontend Frameworks gelöst werden. Ein JS Frontend Framework konsolidiert mehrere Konzepte, Bibliotheken und Datenflüsse, um die Entwicklung von Anwendungen im Web zu vereinfachen und somit auch zu beschleunigen. Jedes Framework bietet eine eigene Struktur und Architektur, die mittels der eingebauten Schnittstellen von einem Entwickler oder einer Entwicklerin genutzt werden können.[30]

Besonders sollten Frontend Frameworks genutzt werden, wenn eine Webanwendung den simplen Raum verlässt und sich Richtung kompliziert und komplex entwickelt. Dies zeichnet sich besonders dadurch aus, dass die Anwendung Formulare und Datenbindung nutzt, sowie Routing und State-Management. Des Weiteren, sollten Javascript Frontend Frameworks genutzt werden, wenn Projekte mit größeren Arbeitsgruppen erstellt werden, da durch die Nutzung eines Frameworks einen gewissen Standard in die Applikation integriert, was den kurzfristigen Einsatz von Entwickelnden erst möglich macht.[31]

Bei der Wahl des JS Frontend Frameworks sollten mehrere Aspekte berücksichtigt werden, darunter gehören vor allem die Nutzeranforderungen, die Architekturanforderung, die Skalierbarkeit und Performanz.[32]

2.2.4 Vue.js und Nuxt.js

Vue.js ist ein Beispiel für ein Javascript Frontend Framework, welches speziell für die Entwicklung von Webapplikationen oder sogenannten Single-Page Apps entwickelt wurde.[33] Dabei liegt Vue.js den Fokus darauf, einheitliche Methoden einzuführen, um klassische Konzepte, wie Interaktivität, Datenzentralität und Komponenten nativ mit sicheren und standardisierten Lösungen zu vereinfachen. Hierbei nutzt Vue.js alle drei Komponenten einer Webseite, also HTML, CSS und JS und kombiniert diese so, dass die Grenzen zwischen den Komponenten verschwimmen.

```
<template>
    <div id="app">
      <button @click="count++">
```

[29] vgl. Vyas, 2022, p. 302.
[30] vgl. Pšenák und Tibenskỳ, 2020, p. 64.
[31] vgl. Persson, 2020, p. 39.
[32] vgl. Vyas, 2022, p. 307.
[33] vgl. Kyriakidis et al., 2016, Seite. 2.

```
        Count is: {{ count }}
      </button>
    </div>
</template>

<script>
import { createApp } from 'vue'

createApp({
  data() {
    return {
      count: 0
    }
  }
}).mount('#app')
</script>
```

Das Beispiel hier zeigt, wie die Grenze zwischen HTML und JS verschmilzt, es ist erkennbar, dass HTML in dem Template Teil genutzt wird. Das HTML enthält einen 'div', also einen Container, indem in diesem Beispiel ein Button-Element ist. Der Knopf enthält in seinem inneren Teil den Text '{{ count }}', dieser ist ein Verweis auf die Variable 'count', die in dem data() Objekt in Javascript definiert und initialisiert ist. Der User sieht nun direkt auf der Webseite eine Zahl anstelle des Textes '{{ count }}'. Wenn er nun den Knopf drückt, erhöht sich in dem data() Objekt der Zähler auf eins, welches der Nutzende direkt in dem Inhalt des Knopfes sieht, da '{{ count }}' nun eins anzeigt.[34] Durch dieses simple Beispiel wird direkt ersichtlich, welchen Vorteil das Vue.js Framework mit Hilfe der Konzepte der Reaktivität und der Datenbindung erzielt. Entwickelnde können diese Vorteile nutzen, um einen einheitlichen, leichten und effizienten Quelltext zu erzeugen.
Neben diesen Konzepten gibt es bei Vue.js auch die einfache Verwaltung von zentralen lokalen Datenspeichern, so dass Nutzerdaten als Beispiel immer an einem zentralen Ort erreichbar sind und keine Schnittstellen innerhalb des Frontends konstruiert werden müssen. Hierfür bietet Vue.js bis vor Kurzem den sogenannten Vuex Store an, welcher seit neustem offiziell von dem Pinia Store ersetzt wurde. Des Weiteren bietet Vue.js ein Template, welches eine optische, aber auch funktionale Trennung von Code vorsieht, so wird ein Vue.js Dokument, wie bereits angeschnitten, in drei Oberbereiche unterteilt. Dem Template als HTML-Container, dem Skript-Bereich

[34] vgl. You, 2022a.

für die Logik mittels Java- oder Typescript und dem Style-Bereich, in dem das für die Optik der Oberfläche relevante CSS oder SCSS steht. Der Skript-Bereich wird um Unterbereiche ergänzt. Hier gibt es unter anderem ein Daten Objekt für das Ablegen von lokalen Daten, ein Methoden Objekt, in dem alle Funktionen untergebracht sind und viele weitere optionale Blöcke.

All diese und weitere Funktionen, die Vue.js liefert, sorgen dafür, dass Vue.js eines der meistgenutzten Javascript Frameworks ist.[35] Ein Problem von Vue.js ist, dass die Übersetzung des Javascript-Quelltextes erst bei dem Nutzenden passiert, weswegen die Listung bei Suchmaschinen somit effektlos ist, da diese nach aktuellem Stand Javascript nicht vollumfänglich verarbeiten können.[36] Des Weiteren gibt es noch weitere Punkte, die Vue.js bewusst nicht unterstützt, dennoch sind einige dieser Punkte trotzdem essentiell damit ein Projekt mit oder auf Basis von Vue.js erstellt werden kann. Daher gibt es eine parallele Zusatzentwicklung von Vue.js, die Nuxt.js heißt. Nuxt.js wird auf der Basis von Vue.js betrieben, es ermöglicht jedoch, dass Web Applikationen statt bei dem Nutzenden, auf dem Server verarbeitet werden, entweder direkt bei jeder Anfrage mit dem 'Server-Side Rendering' oder einmalig zur Kompilation mittels der 'Static-Site Generation'. Nuxt.js ergänzt Vue.js ebenfalls mit weiteren Konzepten, wie Middlewares, Modulen und Plugins.[37]

3 Projektplanung

Nachdem nun die wesentlichsten Technologien erläutert wurden, wird in dem Folgenden Abschnitt die Projektplanung sowohl vom inhaltlichen Aspekt, als auch zeitlichen Verlauf her, betrachtet.

3.1 Zielsetzung und Umfang

Zu Beginn hatten alle Projektmitglieder eine etwas andere Vorstellung von der umzusetzenden Funktionalität und des Umfangs der Webseite, da noch nicht bekannt war, welcher Nutzen abgebildet werden soll und bis zu welchem Punkt die Funktionalitäten implementiert werden. Initial geplant war eine Art Todo-App, welche den Nutzenden erlauben sollte eigene Aufgaben zu definieren und diese zu verwalten. In einem gemeinsamen Workshop hat das Team sich dagegen entschieden, die Webseite

[35] vgl. Overflow, 2022.
[36] vgl. Botify, o. D.
[37] vgl. Kok, 2020, S. 10f.

in dieser Form umzusetzen, da es sich einig war, dass die gebotene Funktionalität nicht ausreicht und die Nutzung der Webseite nicht rechtfertigen würde. Im weiteren Verlauf der Diskussion hat sich herausgestellt, dass eine Webseite erstellt werden sollte, die viele alltäglich benötigte Funktionen abdeckt, wozu auch das Erstellen von Aufgaben zugehört. Letztendlich fiel die Entscheidung auf ein Widget-basiertes Dashboard, da somit benötigte Funktionen, wie eine Suchleiste, die den eingegebenen Text als Suche auf der Suchmaschine Google öffnet, das aktuelle Wetter, die aktuellen Nachrichten und schlussendlich selbstdefinierte Aufgaben anzeigt. Das Projektziel lautet somit: „Das Team möchte ein nützliches Dashboard anbieten, welches die Startseiten der üblichen Browser, wie Chrome oder Edge, revolutionieren soll“.

3.2 Methodik

Ein weiteres Problem ergab sich im Zuge des Projektmanagements, wie sollte das Projekt und die umzusetzenden Punkte getracked werden. Außerdem war unklar, wie die Aufgaben in Arbeitspakete aufgeteilt werden sollten. Möglich wäre zum Beispiel eine Unterteilung in Frontend und Backend oder eine modulare Unterteilung gewesen. Zuerst musste das Team sich jedoch für ein Vorgehensmodell entscheiden, da dieses die vorher genannten Punkte massiv beeinflussen könnte. Klassische Vorgehensmodelle, wie das Wasserfallmodell oder das V-Modell, waren für das geplante Vorhaben nicht geeignet, da sie klare und unveränderliche Anforderungen benötigen, welche nicht gewährleistet werden konnten.[38] Außerdem spielt die zeitliche Komponente und der mittlere Umfang des Projektes natürlich eine große Rolle. Mehrphasige Vorgehensmodelle mit dedizierten Analysephasen sind nach Meinung des Projektteams ungeeignet für eine kleine Webseite, vor allen Dingen will das Team schnell auf Nutzerfeedback reagieren können, was logischerweise bei vorher fest definierten Anforderungen nicht gewährleistet werden kann. Agile Vorgehensmodelle benötigen diese Art von starren und unveränderlichen Anforderungen nicht. Das agile Vorgehensmodell SCRUM entspricht zwar prinzipiell den Anforderungen, da es eine iterative Vorgehensweise bietet und Selbstorganisation ermöglicht. Jedoch sollten für SCRUM mehrere Rollen definiert werden und da das Projektteam in diesem Fall aus drei Personen besteht, wurde es als nicht sinnvoll erachtet, einen Product Owner und einen Scrum Master zu ernennen.[39] Auch wenn der vollständige SCRUM-Ansatz nicht für sinnvoll gehalten wird, sind die agilen Ansätze dennoch gut mit dem Projekt vereinbar. Ein Modell, welches diese agilen Eigenschaften eben-

[38] vgl. Vivenzio und Vivenzio, 2013, S. 6 f.
[39] vgl. Hohberger, 2018, S. 118 ff.

falls erfüllt, aber für kleinere Projekte mit sehr kleinen Projektteams geeignet ist, wäre ideal. Das Modell, das diese Anforderungen erfüllt, ist das bereits beschriebene Kanban Modell. Es erlaubt Transparenz und Selbstorganisation, sowie eine klare Abtrennung und Priorisierung der Aufgaben. Dieses Kanbanboard hat das Team durch ein Programm namens Trello abgebildet und in diesem mehrere Spuren definiert, diese lauten:

1. „Basics und Fundamentals“; Diese Spur diente der initialen Themensammlung für die Grundlagen
2. „Ideas and Backlog“; Für die weitere Ideensammlung und potentielle Anforderungen, die mit aufgenommen werden könnten
3. „Planned“; Für bestätigte Anforderungen, welche aber noch nicht begonnen wurden
4. „In Work“; Für User Stories, welche aktuell bearbeitet werden
5. „Review“; Für fertige User Stories, die aber nochmal durch das vier Augen Prinzip kontrolliert werden sollen
6. „Done“; Für komplett abgeschlossene und sozusagen „releasefähige“ User Stories

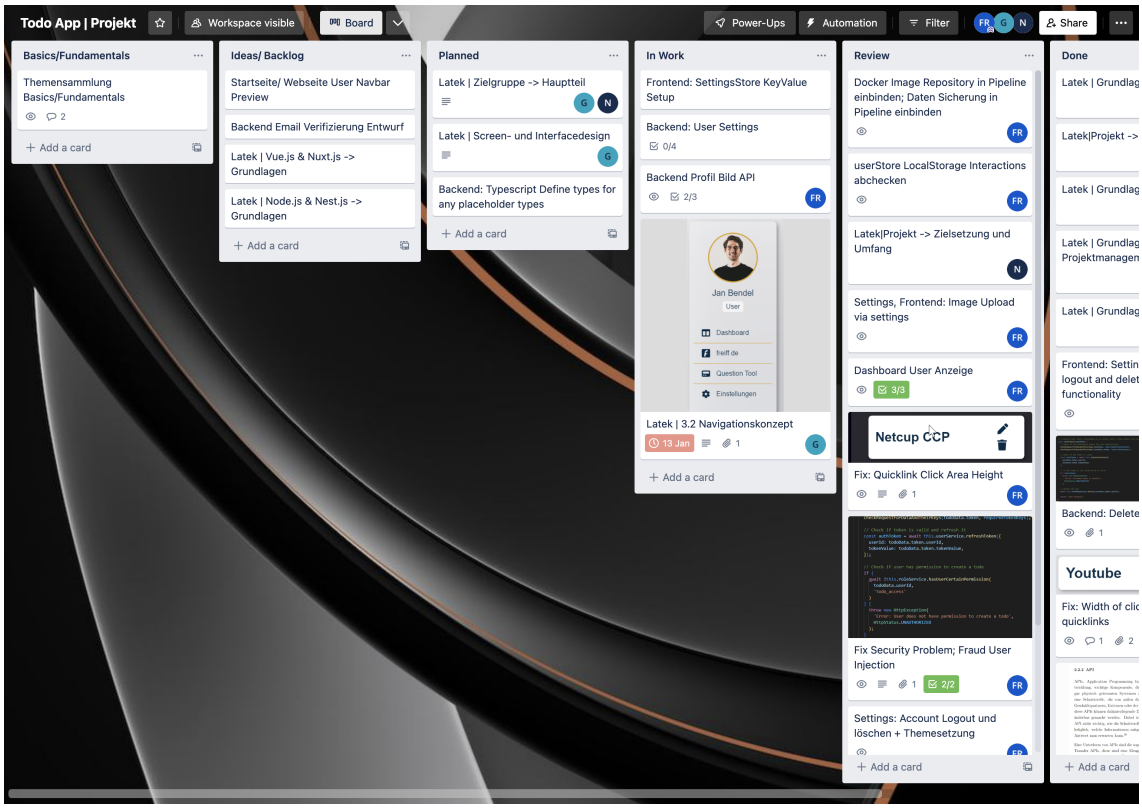

Abbildung 1: Trello Board

Dieses Board wurde sowohl für die Projektdokumentation, in Form dieser Arbeit, als auch für technische Aufgaben verwendet. Den jeweiligen Aufgaben sind einzelne Personen zugewiesen, diese gelten dadurch als „Single Point of Contact“, also alleiniger Ansprechpartner und Verantwortungsträger. Dadurch sind keine klaren Projektphasen definiert, sondern kleine modulare und vor allen Dingen funktionale Arbeitspakete, welche jeweils in ihrer eigenen Bearbeitungsphase sind.

3.2.1 Zielgruppe

Um die Nutzenden in Gruppen aufteilen zu können hat das Team zuerst eine Grobanalyse möglicher Nutzergruppen vorgenommen. Dies geschah nach demographischen Merkmalen, sowie gewissen Charakteristika. Außerdem wurden einige potentielle Nutzergruppen aus der Betrachtung entfernt, da wichtigere Nutzergruppen identifiziert wurden. Hierbei hat sich das Team die Kennwerte des Alters, des Berufs, und des Grades an Organisiertheit in den Fokus genommen. Diese Faktoren hat das Projektteam jeweils bewertet, um eine Person für die Zielgruppe zu identifizieren.

Target Group

Zielgruppe	Nicht Zielgruppe
• Organisierte Menschen • Junge Menschen/ Studenten • Menschen mit hoher Internetnutzung • kleinere Unternehmen • Personen die viel visualieren/ präsentieren • Gewohnheitsmenschen, die ihre gleichen Websites jeden Tag aufsuchen • "Bastler" die gerne customisieren	• Nicht Tech-affine Menschen • Unorganisierte Menschen • ältere Menschen(50+) • Default User der nur vorinstallierte Programme nutzt • Menschen ohne mobiles Endgerät oder PC • mittlere bis große Unternehmen, die dedizierte Tools kaufen

Abbildung 2: Zielgruppe

Wie auf der Zielgruppenanalyse zu sehen, sieht das Team besonders junge und organisierte Menschen als Teil der Zielgruppe. Ältere, nicht mit dem Internet vertraute und weniger organisierte Menschen, oder gar Menschen ohne Endgerät, jedoch nicht. Das Geschlecht hält das Team für weniger relevant, da das Internet heutzutage vollumfänglich genutzt wird.[40] Doch welchen Nutzen erhofft sich das Team nun aus dieser Analyse? Zum einen soll der Fokus in Richtung Zielgruppe im Design und in

[40] vgl. Harwardt, 2020, S. 17.

der Entwicklung gelegt werden. Das Team könnte zum Beispiel auf klassische Designs, die eher an Personen in höheren Altersgruppen gerichtet sind, verzichten und modernere Designs wählen. Außerdem könnte auf die finanzielle Lage der Nutzenden geachtet werden, da Studierende oft ein geringes Einkommen besitzen. Daher wäre ein Freemium-Modell denkbar, durch welches einige Premium Funktionalitäten durch ein Abonnement freischaltbar sind. Damit potentielle Nutzende dargestellt werden können, werden die Ergebnisse der Zielgruppenanaylse im Folgenden genutzt, um Personas aufzustellen.

3.2.2 Personas

Um zu garantieren, dass das Produkt den Anforderungen der Kunden gerecht werden kann, hat das Team Personas möglicher Nutzer und Nutzerinnen erstellt. Durch einen gemeinsamen Workshop wurden zwei Personas aus der durch das Team definierten Zielgruppe identifiziert. Zuerst eine junge Studentin und im Anschluss ein junger Unternehmer.

Die Persönlichkeiten der Personas werden durch die Faktoren Organisation, Motivation, Geduld und Computerwissen dargestellt. Der Grad der Ausprägung wird dabei durch einen Schieberegler von niedrig bis hoch geregelt. Zusätzlich wurden die Freizeitaktivitäten, Rolle und Aktionen betrachtet. Die Rolle meint hierbei in welcher Form das Produkt genutzt wird und wie es weiterempfohlen werden könnte. Mit Aktionen sind die Schritte gemeint, die Nutzende ausführen müssen, um die Website nutzen zu können.

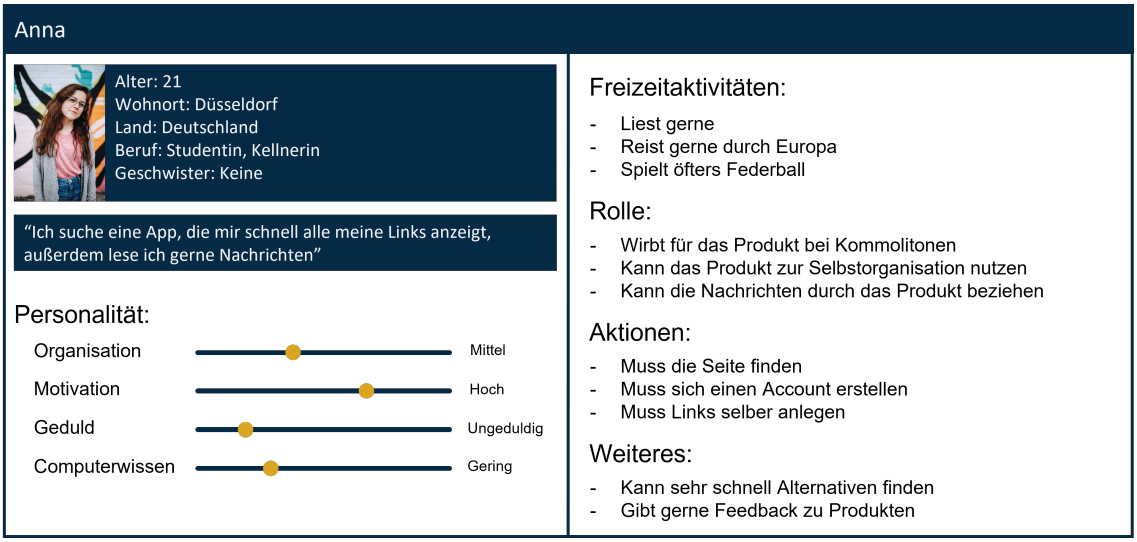

Abbildung 3: Persona Anna

Die erste Persona Anna ist 21 Jahre alt und lebt in Deutschland, sie studiert und kellnert nebenbei und hat oft einen stressigen Alltag. Trotzdem meistert sie diesen Alltag problemlos und ist sehr motiviert und hilfsbereit. Ebenso hat Anna eifrige Ziele und ist sehr organisiert. Wenn sie mal Freizeit hat, verbringt sie diese gerne mit dem Lesen ihrer Lieblingsbücher. Anna kann das Produkt durch direktes Empfehlungsmarketing weitervertreiben, da sie viele Kommilitonen und Kommilitoninnen hat und sich bereits mit vielen angefreundet hat. Ein Risiko ist, dass Anna viele Konkurrenzprodukte bereits kennt. Damit Anna das vom Team entwickelte Dashboard nutzen kann, muss sie nur auf die Webseite aufmerksam werden und sich registrieren.

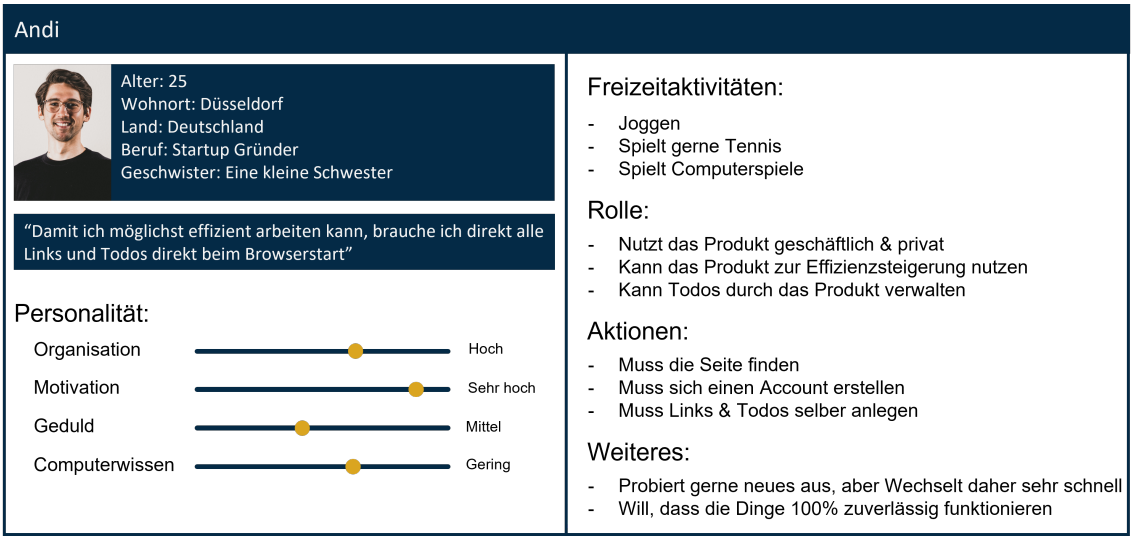

Abbildung 4: Andi

Die zweite Persona, Andi, ist 25 Jahre alt und lebt ebenfalls in Deutschland. Er ist ledig und hat sein eigenes Startup gegründet. Seine Freunde bezeichnen ihn als wahren Workaholic, es gäbe wohl keine Minute an der er nicht an die Arbeit denken würde. Die Arbeit in seinem Startup erfüllt ihn voll und ganz. Er arbeitet komplett alleine und ist organisiert, er möchte nämlich schon innerhalb von 5 Jahren in die Frührente. Er als Solounternehmer möchte natürlich ungern teure Lizenzen für Tools kaufen und verlässt sich lieber auf kostenlose Alternativen. Er nutzt die Webseite sowohl privat, als auch für Kundenaufträge. Jedoch könnte Outlook eine ähnliche Funktionalität abbilden. Er muss die Webseite ebenfalls finden und sich registrieren, um sie zu nutzen.

Die Personas haben die aktuellen Stärken und Schwächen der Webseite klar aufgezeigt und die Zielgruppe weiter verschärft. Das Team sollte sich hauptsächlich an

deutsche Nutzer und Nutzerinnen wenden. Diese sollten relativ jung und organisiert sein. Doch gibt es viel Konkurrenz von globalen Großkonzernen mit weitverbreiteten Tools. Damit die Konkurrenz an Relevanz verliert, hat das Team sich dazu entschieden auf Lokalität zu setzen. Eine Integration der Tagesschau API hat sich perfekt angeboten, um den Nutzenden mehr Komfort zu bieten und gleichzeitig etwas zu schaffen, dass aktuell keiner der bekannten Anbieter implementiert hat.

3.3 Konzeption und Entwurf

Das Kapitel Konzeption und Entwurf beschäftigt sich sowohl mit dem Aufbau der Webapplikation, als auch mit der Entwicklung vom anfänglichen Prototypen bis hin zum interaktiven Design. Zudem wird beschrieben, wie und mit Hilfe von welchen Konzepten und Methodiken das Team aus einer anfänglichen Idee einen vollumfänglichen Entwurf erstellt hat. Beginnend mit der Methodik des Prototypings, darauf folgend die Konzepte der Navigation bis hin zum Screen-und Interactiondesign bietet das Kapitel einen Einblick in die unterschiedlichen Entwicklungsschritte zu einem ersten Webapplikationsdesign.

3.3.1 Prototyping

Zu Anfang ist sich das Team noch nicht im Klaren gewesen, wie genau eine mögliche Webapplikation im Bereich der Webtechnologien später einmal aussehen kann und welche Funktionalitäten diese abdecken könnte. Es konnte also zu diesem Zeitpunkt noch nicht mit Sicherheit gesagt werden, welche Navigationskonzepte oder welches Screen-/ Interactiondesign sich für das spätere Endprodukt bewähren würden. Aufgrund dieser Tatsache hat es sich angeboten im Zuge der Konzeption und des Entwurfs die Methodik des Prototyping als Ideen- und Entscheidungsfindung zu verwenden.

Prototyping ist nicht nur ein nützliches Tool, welches zur ersten Einschätzung des Produktes beziehungsweise Produkterweiterungen genutzt werden kann, sondern auch um beurteilen zu können, welche Wirkung das Produkt auf den User hat. Diese Methodik beschränkt sich daher nicht nur auf die Bewertung der einzelnen Funktionalitäten, sondern auch auf die User Experience (UX). UX beschreibt im Grunde genommen die Erfahrungen des Users auf die Webapplikation in bestimmten Szenarien, aber auch im Ganzen.[41] Im Rahmen der Entwicklung

[41] vgl. Christoforakos und Diefenbach, 2017, S. 329 ff.

des Widget-basierten Dashboards baut die User Experience auf Empfindungen der einzelnen Teammitglieder auf, mit Hilfe welcher die Funktionsfähigkeit der Elemente der Webapplikation in den unterschiedlichsten Stadien des Prototyps beurteilt wurden.

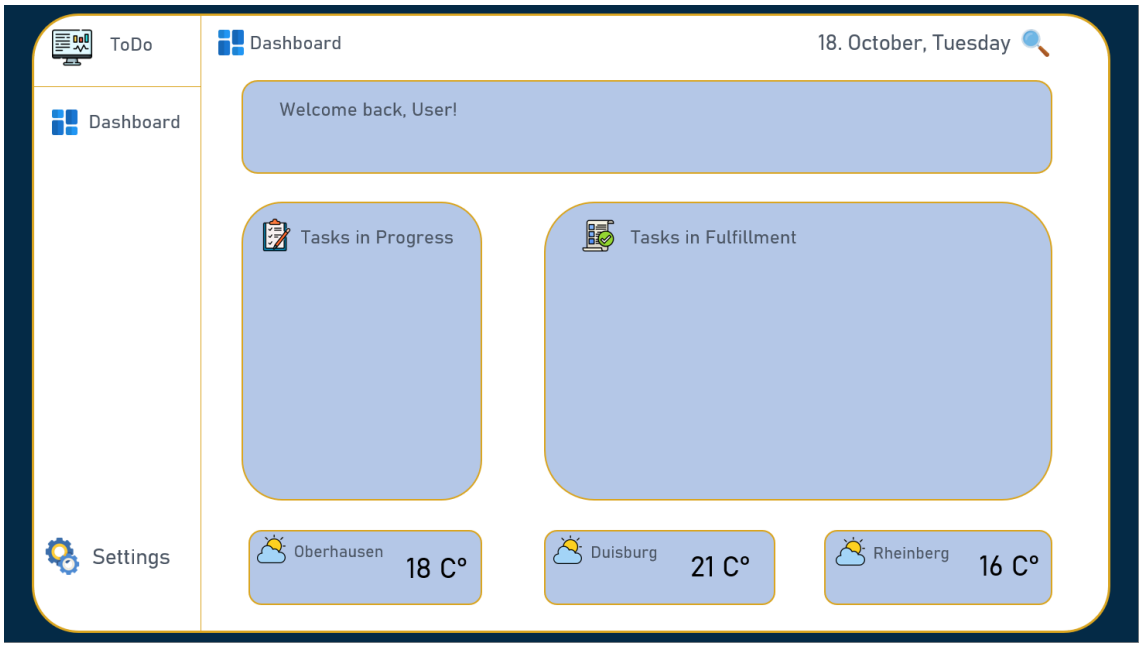

Abbildung 5: Prototyp

In Abb.5 ist einer der ersten Prototypen des Dashboards zu erkennen, bei welchem es noch nicht möglich war mit diesem zu interagieren. Zu diesem Zeitpunkt nutzte das Team ihn hauptsächlich zur anfänglichen Ideenfindung und Evaluation. Im weiteren Verlauf der Seminararbeit werden die unterschiedlichsten Stadien des Prototyps veranschaulicht bis hin zur Umsetzung der Webapplikation.

3.3.2 Navigationskonzept

Neben den Komponenten des Backend und der Datenebene, welche einen Großteil der Applikation ausmachen, beschäftigen sich Navigationskonzepte mit der Umsetzung bestimmter Designideen im Frontend, welche letztendlich dazu führen sollen, dass der Nutzende sich konsistent durch die Applikation navigieren kann.[42]
Im Laufe der Zeit haben sich die unterschiedlichsten Arten von Navigationskonzepten etabliert, wobei es sich hierbei nicht nur um Konzepte handelt, welche für die Webseitenavigation verwendet werden, sondern vielmehr Methodiken, welche ihren

[42] vgl. Blas et al., 2020, S. 14.

Ursprung deutlich früher in der Geschichte haben, wie zum Beispiel das Navigationskonzept der Bostoner U-Bahn.[43] Um gewährleisten zu können, dass das Bostoner U-Bahn Netz von allen verständlich interpretiert und korrekt genutzt werden kann, folgt das Konzept mehreren Prinzipien, welche versuchen dem Nutzenden die Information simpler und noch deutlicher darzustellen.
Zum einen wird auf die Nutzung von prägnanten Farben bei den verschiedenen U-Bahn Linien gesetzt, um diese besonders kenntlich zu machen.[44] Zum anderen werden auch verschiedene visuelle Hierarchieebenen verwendet, welche sich beispielsweise in unterschiedlicher Größe der Elemente widerspiegeln. Zu den bekanntesten visuellen Hierarchien, welche bei den Navigation verwendet werden, gehören:

1. Relative Größe: Bei der relativen Größe handelt es sich um die bewusste Nutzung unterschiedlicher Elementgrößen damit der Nutzende die Wichtigkeit des Elements erkennen kann.

2. Platzierung/ Position: Unterschiedliche Positionen werden bestimmten Betrachtungshäufigkeiten zugeordnet und besitzen somit eine höhere Wahrscheinlichkeit früh erkannt zu werden. Somit ist es sehr üblich, dass es bei Sprachen, welche von links nach rechts gelesen werden, die linke Seite mit einer hohen Wahrscheinlichkeit als Erstes betrachtet wird. Diese Beobachtung ist unter anderem mit für die bekannte links gebundene Navigationsleiste verantwortlich.

3. Farbnutzung und Kontrast: Mit einer durchdachten Farbnutzung und mit Hilfe unterschiedlichster Kontraste können Elemente hervorgehoben und Beziehungen zwischen diesen aufgezeigt werden. Das menschliche Auge ist in der Lage diese markanten Unterschiede in kürzester Zeit zu erkennen und ermöglicht somit eine schnelle und klare Navigation.

4. Bewegung: Das Interesse des Nutzenden kann einerseits durch bewegende Elemente auf sich gezogen werden und andererseits hilft es zudem beim Navigieren durch die Applikation, da sich der Nutzende mit Hilfe von diversen Bewegungen der Elemente beim Drüberfahren mit der Maus zurecht finden kann.[45]

[43] vgl. Fleming und Koman, 1998, S. 1 ff.
[44] vgl. Fleming und Koman, 1998, S. 2 ff.
[45] vgl. Fleming und Koman, 1998, S. 2-5.

Abbildung 6: Navigationsleiste

Nachdem die Navigationskonzepte und die vier Hierachieebenen theoretisch erläutert worden sind wurden diese auf die Todo-App übertragen. Zunächst wurde die Position festgelegt, bei welcher das oben genannte Prinzip der links gebundenen Navigationsleiste angewandt worden ist. Aufgrund der Tatsache, dass die App in deutscher Sprache verfasst wurde, welche von links nach rechts gelesen wird, ist die Häufigkeit erhöht, dass der User seine Aufmerksamkeit zunächst auf die linke Hälfte des Bildschirms richtet.[46] Des Weiteren wurde die Schriftfarbe deutlich dunkler gewählt, als die Hintergrundfarbe, um einen Kontrast zu erzeugen, welcher dem Nutzenden ermöglicht die unterschiedlichen Optionen in der Navigationsleiste klar und deutlich auszumachen. Darüber hinaus kann durch Bewegung der einzelnen Elemente in der Navigationsleiste, sobald sich der Nutzende mit der Maus über diesen befindet, schnell ausgemacht werden an welcher Stelle sich der Navigierende befindet. Durch die kombinierte Anwendung dieser Konzepte ermöglicht die Navigationsleiste eine möglichst effiziente Bedienung und klare Anwendung der Webapplikation.

[46] vgl. Fleming und Koman, 1998, S. 2.

3.3.3 Screen- und Interactiondesign

In den Anfängen der Erstellung der Webapplikation hat sich das Team Gedanken über das Screendesign und über das Interactiondesign gemacht. Screendesign beschäftigt sich mit dem Aufbau der Webseitenelemente, beziehungsweise mit der Anordnung dieser auf dem Bildschirm.[47]

Abbildung 7: Screendesign

Aufgrund der Tatsache, dass es sich um ein widget-basiertes Dashboard handelt, hat sich das Team für vier kachelförmige Rechtecke entschieden. Die Rechtecke besitzen abgerundete Ecken, damit diese besser zum Gesamtdesign passen und bilden den Grundaufbau der Widgets. Diese befinden sich rechts neben der Navigationsleiste und nehmen den restlichen Teil des Bildschirms ein. Das Team hat sich hierbei für einen gewissen Grad an Transparenz bei dem Design der Rechtecke entschieden, um einen Kontrast zu erzeugen, welcher es ermöglicht, dass sich die Widgets vom Hintergrund abheben und als Bedienelemente direkt erkannt werden können.[48] Jedes Widget besitzt den gleichen Aufbau, aber unterscheidet sich im Inhalt und der Funktion. Hierbei ist es wichtig zu erwähnen, dass durch die Unterstützung von Themes, das Design sehr variabel und individuell sein kann. Das Team wird einige Standard-Designs als Themes hinterlegen, die gut auf das Tool abgestimmt sind, darunter gehört auch ein automatischer Nacht-Modus, damit der Nutzende direkt von der zum Tool passenden Farbwahl profitiert.
Die vier Widgets wurden in ein Nachrichten-, Quicklinks-, Todo- und Wetterwidget

[47] vgl. Herfurtner, 2003, S. 13 ff.
[48] vgl. Fleming und Koman, 1998, S. 3.

unterteilt, bei welchen das Design stets an die jeweilige Funktion angepasst wurde. So ist es zum Beispiel bei dem Wetterwidget der Fall, dass dieses mit Hilfe von Wettersymbolen lesbarer gemacht wird.

In der oberen rechten Ecke befinden sich Datum und Uhrzeit, welche es den Nutzenden ermöglicht diese immer im Blick behalten zu können. Über den Widgets befindet sich zudem noch eine Suchleiste, welche eine direkte Google-Suche ermöglicht. All diese Elemente schließen sich dem Widgetdesign an, um ein einheitliches Auftreten von Webapplikation und Dashboard zu ermöglichen.
Im nächsten Teil dieses Unterkapitels wird ein genaueres Augenmerk auf das Interactiondesign gelegt. Das Interactiondesign beschäftigt sich mit der Schnittstelle zwischen Produkt und Mensch und versucht zu erahnen, wie Nutzende mit bestimmten Elementen und Dialogfeldern interagieren. Es setzt sich also in diesem Fall mit der Kommunikation zwischen dem Nutzenden und der Webapplikation auseinander und beschreibt den Dialog mit dem Dashboard in unterschiedlichsten Situationen.[49]
Ein mögliches Beispiel hierfür wäre die Art der Interaktion zwischen dem Nutzenden und dem Nachrichtenwidget, sobald die Maus sich über einer der verschiedenen Nachrichten befindet. In diesem Fall reagiert die Applikation entsprechend indem sich die ausgewählte Meldung leicht nach rechts bewegt und somit dem Nutzenden aufzeigt, dass sich diese aktuell in der Auswahl befindet und mit Hilfe eines Rechtsklicks öffnen lassen würde. Wie bereits anfänglich erwähnt, gibt es in dem gewählten Beispiel eine Art Dialog zwischen der Applikation und dem Nutzenden.

4 Umsetzung

Für die Umsetzung eines solchen Projekt bedarf es mehrerer Komponenten, darunter gehört ein Frontend, ein Backend und die Datenebene, Diese drei Hauptkomponenten bilden die Basis der Applikation und sind der Hauptbestandteil einer sogenannten 'Three-Tier Architecture'[50], die in dem Projekt auch eingesetzt wurde. Das Frontend behandelt sowohl das User Interface, als auch die durch den Nutzenden gewünschten Funktionen. Das Frontend ist hierbei immer der Auslöser einer Kommunikation zum Backend. Das Backend hat die Hauptaufgaben der Anfragenverwaltung, Datenbankverwaltung und Dateiverwaltung, es muss die Anfragen eines Users korrekt behandeln und alle Schritte in den Weg leiten, die benötigt werden,

[49] vgl. Kolko, 2010, S. 12 ff.
[50] vgl. Blas et al., 2020, S. 14.

um die Anfrage zu behandeln. Neben dem Front- und Backend gibt es noch viele weitere Komponenten, die für einen laufenden Betrieb einer Web-Applikation benötigt werden, so zum Beispiel Proxies, Firewalls, Datenbanken und Fileserver.

4.1 Umsetzung: Frontend

Für die Implementierung des Frontends wurde das Javascript Framework Nuxt.js, eine Unterform von Vue.js, angewendet, da dies zum ersten die Static-Side Generation[51] unterstützt, welches für das Erscheinen bei Suchmaschinen essentiell ist, da Suchmaschinen Javascript Dateien schlecht lesen und gemäß der SEO-Parameter auswerten können. Weitere Vorteile, die für die Auswahl von Nuxt.js relevant waren, sind das einfache State-Management via Vuex[52], die Standard-Architektur Vorgaben und die umfangreiche Dokumentation.

4.1.1 Erster Entwurf und Hierarchiekonzept

Um ein Nuxt Projekt zu erstellen, wird als eine von mehreren Möglichkeiten, der NPM Paket Manager[53] verwendet und mit ihm der Initialisierung-Befehl für eine neue Nuxt App genutzt.

```
npm init nuxt-app todo-app
```

Nach dem ersten Aufsetzen des Nuxt Projekts, war das Ziel einen ersten Entwurf zu erstellen, der lediglich das grobe Layout vorgeben sollte.

```
<template>
  <div>
    <div class="page-frame">
      <div class="left-bar-left"></div>

      <div class="top-bar"></div>

      <div class="dashboard">
        ...
      </div>
```

[51] vgl. Varbanov und Filipov, 2020, S. 27f.
[52] vgl. You, 2022b.
[53] vgl. npm Inc., 2022.

```
    </div>
  </div>
</template>
```

Zu dem HTML wurde nun CSS-Quelltext erstellt und eingebunden, so dass aus dem Template Quelltext und dem darin enthaltenen CSS ein für den Nutzenden erstes sichtbares Bild erzeugt wird.

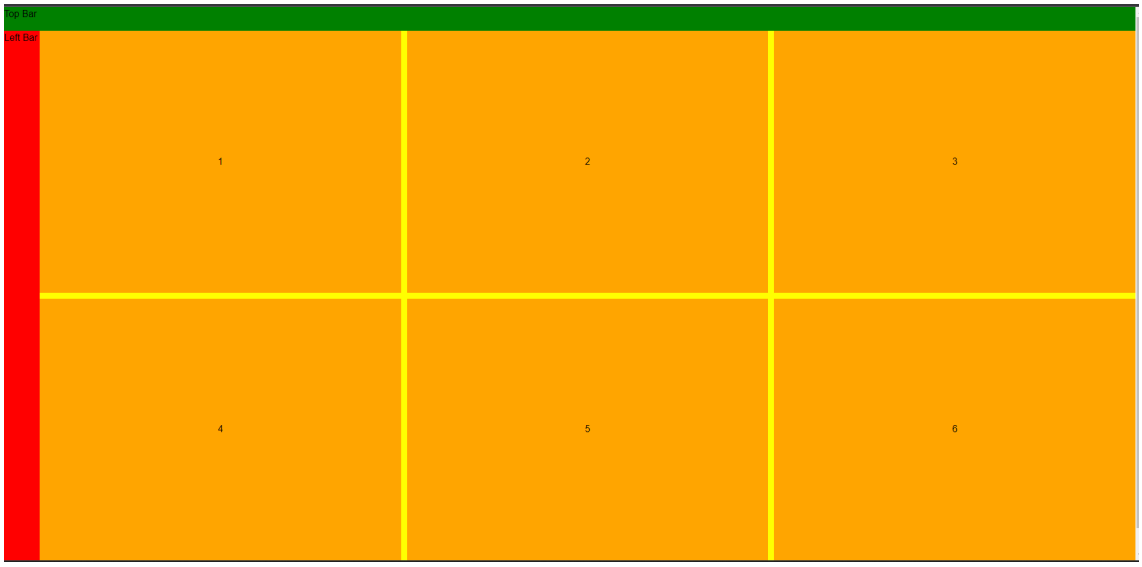

Abbildung 8: Erster Layoutentwurf für das Dashboard

Nachdem das grobe Design stand, ging es an die Architektur des Projekt. Hier wurde die Entscheidung getroffen, das sowohl das Design, das sogenannte Theme, als auch die Kacheln, auch Widgets genannt, konfigurierbar sein sollen, weswegen hierfür eine Javascript Konfiguration eingesetzt wurde, in der sowohl mehrere Themes einstellbar sind, als auch die Widgets austauschbar sind.

```
export default {
  data() {
    return {
      widgets: [
        {
          id: 0,
          is: 'WidgetsDefault',
          data: {
            title: 'Nachrichten',
            comp: 'WidgetsNews',
```

```
        },
        useTheme: true,
      },
      ...
    ],
    themes: [
      {
        page: {
          backgroundColor: 'transparent',
          '--pageAccent': 'var(--accent)',
        },
        pageBackground: {
          background:
            'url(' +
            require('@/assets/dashboard/bg-5-whiteblue.jpg') +
            ')  no-repeat center center fixed',
          filter: 'blur(64px)',
        },
        ...
      },
    ],
    ...
  }
},
```

Erstellung des Hierarchiekonzeptes

Bereits in der Konzeptionsphase stellte sich heraus, dass das Projekt eine Hierarchie bilden wird, es gibt mehrere Seiten, jede Seite hat einen eigenen Inhalt und gerade die Widget-Seite bildet zusätzlich Hierarchien in sich selber. Da es sich bei Nuxt.js um ein Framework handelt, welches sich nach Komponenten orientiert, war es für diesen Teil des Projektes die richtige Wahl. Nuxt.js erlaubt es Komponenten ineinander zu verschachteln, diese gleichzeitig aber auch mehrmals zu nutzen und dennoch liegen alle Daten, die zentral liegen müssen, an einem einzigen Ort, nämlich dem Store.[54] Jedoch war für das Team klar, dass für die Implementierung, diese Hierarchien abgebildet werden müssen, um daraus abzuleiten, welche Komponenten wie in Korrelation stehen. Hierfür wurde folgendes Konzept entwickelt:

[54] vgl. Freeman, 2018, S.509 f.

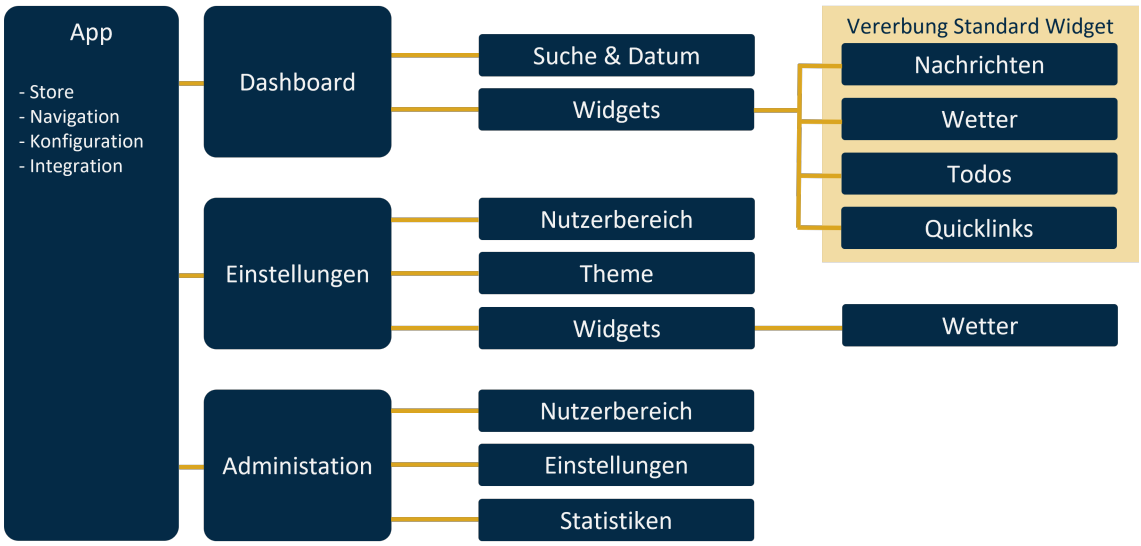

Abbildung 9: Hierarchiekonzept

Aus diesem Hierarchiekonzept wird ersichtlich, dass ganz oben auf der App Ebene, eine einheitliche Navigation existieren muss, damit die Seiten Dashboard, Einstellungen und Administration sich in das Applikations-Ökosystem integrieren. Neben der optischen Einheit, gibt es auch funktionale Anforderungen an die Applikation, nämlich, dass auf dieser Ebene die Stores als zentraler Datenspeicher liegen müssen und auch die Konfiguration von zum Beispiel der URL des Backends etc. hier geschehen muss. Dieses Rahmenwerk nennt sich in Nuxt.js Layout[55], dieses Layout ist ein Container, in dem dann die Seiten, in Nuxt.js Pages genannt, liegen. Seiten unterscheiden sich dadurch, dass sie im Routing unterschiedliche URLs nutzen, zum Beispiel hat die Einstellungsseite die URL *https://freiff.de/dashboard/settings* und die Administrationsseite die URL *https://freiff.de/dashboard/admin*. Besonders hervorzuheben ist hier der Widgets Bereich auf dem Dashboard, welcher im folgenden Teil genauer behandelt wird.

4.1.2 Widgets

Ein zentraler Teil des Projekts sind die Widgets[56], die auf dem Dashboard des Nutzenden angezeigt werden. Auch wenn jedes Widget in sich eine eigene Funktion hat, zum Beispiel Wetter anzeigen oder Nachrichten anzeigen, muss gewährleistet werden, dass sie sich alle gut in das User Interface integrieren. Auch wenn dies in jedem

[55] vgl. Kok, 2020, S.60.
[56] vgl. Melnyk et al., 2014, S.342.

Widget selber gelöst werden kann, macht es wegen der Übersichtlichkeit und Redundanz Sinn, hier eine Vererbung anzuwenden, so dass jedes Widget nur ein Teil eines darüberliegendes Standard-Widgets ist. Dieses Standard-Widget hat genaue Einstellungen zur Größe und zum Aussehen, es bietet somit ein Rahmenwerk in dem sich alle Widgets befinden müssen. Je nach Konfiguration durch die Applikation bzw. den Nutzenden, kann nun der Inhalt in dem Rahmenwerk beliebig geändert werden, dennoch bleibt das Design einheitlich.

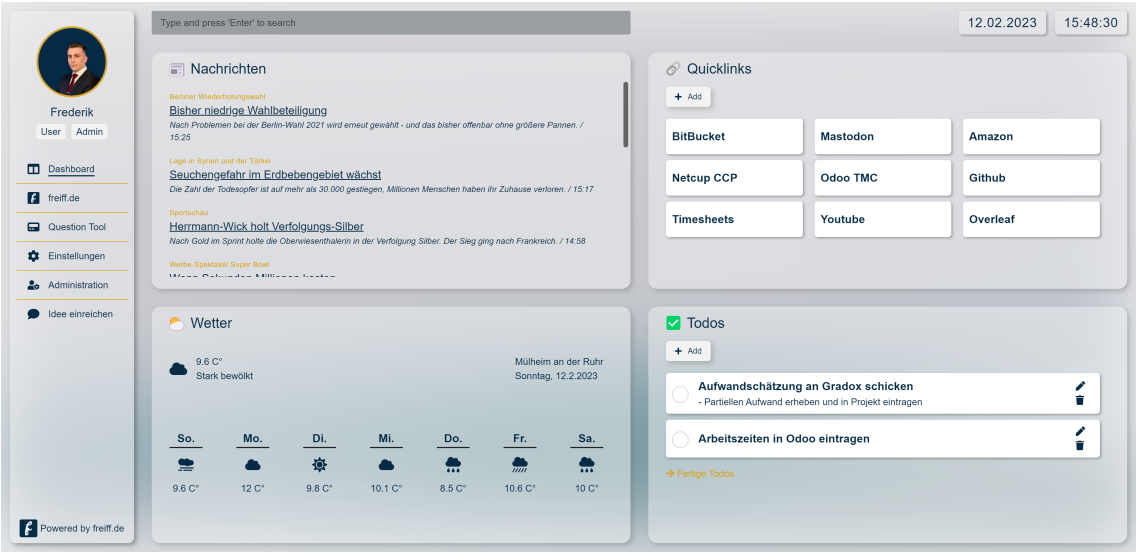

Abbildung 10: Unterschiedlicher Inhalt bei gleichem Design

Hierbei kriegt das Standard Widget den Pfad des darunter anzuzeigenden Widgets sinngemäß vererbt, dies wird durch das 'prop' Attribut gewährleistet. Dieses Attribut wird in dem HTML Platzhalter namens '<component ... />' eingefügt, so dass das vererbte Widget in dem Standard Widget angezeigt wird.

```
<template>
  <div class="widget-frame box-shadow">
    <h2 v-if="widget.title" class="title">{{ widget.title }}</h2>
    <component v-if="widget.comp" class="widget-content"
        :is="widget.comp" />
  </div>
</template>

<script>
export default {
  props: {
    widget: {
```

```
        ...
    },
  },
}
</script>
```

Nachrichten Widget

Das Standard Widget ist nur das Hüllenelement für die richtigen Widgets. An dem Nachrichten Widget kann sehr gut der Aufbau der eigentlichen Widgets festgemacht werden. Dieses nutzt die öffentlich erreichbare API der Tagesschau, weshalb es bei der Erstellung eine Anfrage an die Schnittstelle der Tagesschau stellt, um die aktuellen Nachrichten zu empfangen und lokal im Widget unter dem Bereich 'data()' abzuspeichern.

```
export default {
  data() {
    return {
      feed: undefined,
      ...
    }
  },
  created() {
    this.getNews()
  },
  methods: {
    getNews() {
      // https://tagesschau.api.bund.dev/
      // Get the content of the url https://www.tagesschau.de/xml/rss2/
          with axios and save the contents to the local feed property
      this.$axios
        .get('https://www.tagesschau.de/api2')
        .then((response) => {
          this.feed = response.data
        })
        ...
    },
  },
  ...
}
```

Die Datenschicht der Komponente muss nun noch dem User sichtbar gemacht werden, dies passiert mit dem Template Element des Widgets, in dem HTML genutzt wird und mit Hilfe von Vue.js die Daten eingefügt werden. Hierfür wird eine for-Schleife als Kontrollstruktur genutzt, die über eine sortierte Variante der Nachrichten iteriert. Für jede Nachricht wird nun ein Block an HTML mit Überschrift, Datum, etc. generiert, so dass am Ende direkt durch Vue mehrere Elemente durch wenige Zeilen Code erstellt werden.

```
<template>
  <div class="widget-frame">
    <div class="newsfeed" v-if="feed">
      <div
        v-for="news in getReducedSortedNews"
        :key="news.externalId"
        class="news"
      >
        <a target="_blank" :href="news.shareURL">
          <p class="topline">{{ news.topline }}<br /></p>
          <h2 class="title">{{ news.title }}<br /></h2>
          <p class="preview">
            {{ news.firstSentence }} /
            {{ new Date(news.date).toLocaleTimeString().slice(0, 5) }}
          </p>
        </a>
      </div>
    </div>
    <p v-else>{{ widgetState }}</p>
  </div>
</template>
```

4.1.3 Daten Management und Store

Während bei statischen Webseiten die Medien und Informationen im Vordergrund sind, befinden sich bei Web-Applikationen besonders Funktionen und Daten im Zentrum. Web-Applikationen verarbeiten Daten mit Hilfe von Funktionen vergleichbar zu normalen Desktop Anwendungen. Daher ist es auch für Web-Applikationen unerlässlich, ein Konzept zur Datenverwaltung und zum Datenzugriff zu haben. Auch wenn durch das falsche Setzen von Daten durch z.B. Fehler im Quelltext nur die

Nutzenden betroffen sind, die die Webseite gerade geöffnet haben und nicht direkt alle User, wie im Backend, ist es dennoch wichtig, dass die User Experience nicht darunter leidet, dass Daten falsch angezeigt werden, oder nicht reaktiv beziehungsweise aktuell sind. Hier zeigt sich ein weiterer Vorteil, der durch den Einsatz von Nuxt.js bzw. Vue.js erzielt werden kann, denn die beiden Framework Ausprägungen bieten einen Vuex Store[57], dies ist praktisch eine Datenumgebung, in der Daten, auch State genannt, ausschließlich über Mutationen verändert werden können. Der Vorteil hier ist, dass jede Mutation dokumentiert und rückgängig gemacht werden kann. Neben dem State und den Mutationen gibt es noch Actions, diese beinhalten die Logik und diese lösen Mutationen unter bestimmten Bedingungen aus. Actions sind die einzige Variante, mit denen Datenveränderungen außerhalb des Stores an sich gemacht werden können. Damit die Daten von außerhalb des Stores erreichbar und lesbar sind, gibt es als viertes Element noch Getter, welche simple Methoden sind, die den State lesen und bei Bedarf in einer klar definierten Formen zurückgeben.
Im Rahmen des Projekts wurde das Konzept des Stores sehr umfangreich genutzt, denn mit diesem lassen sich sehr gut auch logische Trennungen implementieren. So wurde mit den Stores folgende Hierarchie aufgebaut:

```
store
├─ userStore.js
├─ settingsStore.js
├─ todoStore.js
└─ quicklinkStore.js
```

Aus dieser Hierarchie werden die Aufgaben der einzelnen Stores sehr gut ersichtlich, so ist der userStore zuständig für die Nutzerverwaltung, also dem Login, dem Registrieren und Ähnlichem. Zusätzlich dient dieser Store der Rollenverwaltung. Auch wenn das Backend immer eine Verifizierung der Rolle durchführt, da Schnittstellen sonst ausgenutzt werden können, ist es für das Frontend trotzdem wichtig zu wissen welche Rollen der Nutzende hat, denn so kann gewährleistet werden, dass das User Interface zum Beispiel bei normalen Usern in der Navigation den Link für die Administrationsseite ausblendet, da diese auf der Administrationsseite ohnehin keine Funktionen nutzen können.

Daten Management am Beispiel des User-Stores

Da die Applikation, die in dem Projekt umgesetzt wird, persönliche Dashboards mit

[57] vgl. Freeman, 2018, S.509 f.

nutzerbezogenen Daten anzeigen soll, wird ein umfangreiches Nutzer-Management benötigt, das Frontend muss basierend auf Nutzerberechtigungen, -metadaten und weitere Daten unterschiedliche Verarbeitungen durchführen, so dass ein User seine eigenen Todos sieht, seine eigene Konfiguration nutzen kann und sich sowohl authentifizieren, als auch autorisieren kann.

So wird neben wesentlichen Nutzerdaten auch zum Beispiel der Token, ein String-Identifier für die Authentifizierung, im State gespeichert. Hierbei sollte erwähnt werden, dass der Store größtenteils automatisch vom Backend mit Daten gefüllt wird, so zum Beispiel die Token Daten mit denen der User sich beim Backend authentifiziert.

```
export const state = () => ({
  token: {
    value: '',
    expiresAt: '',
  },
  user: {
    id: '',
    email: '',
    username: '',
    roles: [],
  },
  loggedIn: false,
})
```

Damit eine transparente Änderung der Daten erfolgt, werden Mutationen eingesetzt. In diesem Beispiel wird das Nutzer-Objekt im State mit den in der Userdata mitgelieferten Informationen kontrolliert von einer Mutation überschrieben.

```
export const mutations = {
  SET_USER_DATA(state, userData) {
    state.user = {
      id: 'id' in userData ? userData.id : '',
      email: 'email' in userData ? userData.email : '',
      username: 'username' in userData ? userData.username : '',
      roles: 'roles' in userData ? userData.roles : [],
    }
  },
  ...
}
```

Da Mutationen vom Konzept her nur Daten entgegennehmen, um sie kontrolliert in den State zu geben, bedarf es einem Konstrukt, welches die Logik beinhaltet. In Vuex wird dies über 'Actions' realisiert, die Logik beinhalten und bei Bedarf andere Aktionen aufrufen und den State mittels Mutationen verändern. Jene Aktionen werden auch genutzt, um zentrale Orte bereitzustellen, mit dem Backend zu kommunizieren und die Antwort vom Backend entsprechend zu verarbeiten. In dem Beispiel hierunter, nimmt eine Aktion 'credentials', also Anmeldedaten, die der User eingegeben hat, entgegen und startet eine Kommunikation zum Backend an die Login API. Die Aktion wartet im Anschluss auf die Antwort, ist diese angekommen, wird die Antwort aufgeteilt. Der in der Antwort enthaltene Token wird mittels Mutation in den State geschrieben, gleiches mit den Nutzerdaten. Nach weiteren Operationen endet die Aktion und der Nutzende ist formal und global angemeldet.

```
export const actions = {
  // Logging in the user
  async login({ commit }, credentials) {
    try {
      const { data } = await this.$axios
        .post(`${backendURL}/user/login`, {
          loginData: credentials,
        })
      ...

      commit('SET_TOKEN', data.auth.authToken)

      // Setting the user data in the state via the SET_USER_DATA
          mutation
      commit('SET_USER_DATA', {
        ...
      })

      // Setting the token in the local storage
      localStorage.setItem('authToken',
          JSON.stringify(data.auth.authToken))

      // Setting the loggedIn state to true
      commit('SET_LOGGED_IN', true)
```

```
      return true
    }
    ...
  },
  ...
}
```

4.1.4 Token-Authentifizierung

Da in diesem Projekt der Nutzende im Vordergrund steht, ist es wichtig, dass die Nutzung des Dashboard so einfach ist, wie nur möglich. Klassischerweise authentifiziert ein Nutzer oder eine Nutzerin sich mit einem Spezifika, also Nutzername oder Email, und einem Passwort. Dies ist auch bei diesem Projekt der Fall, jedoch ist es aus Nutzerperspektive eine schlechte Erfahrung, wenn vor jedem Öffnen des Dashboards, erstmal ein Login erfolgen muss. Um dies zu Lösen wurde sich dazu entschieden, das Konzept der Token-Authentifizerung[58] zu nutzen, ein Konzept in dem der User sich nur ein mal registrieren muss und von nun an einen einmaligen, persönlichen und zufällig generierten Token erhält. Dieser Token wird nun im Frontend in dem userStore hinterlegt, damit der Token von anderen Methoden, wie die Todo- oder Quicklink Methoden, genutzt werden kann, um sich am Backend zu authentifizieren, ohne dabei das Passwort zu nutzen. Dieser Token wird nicht nur im aktuellen State gespeichert, sondern auch in dem lokalen Speicher des Browser, so dass der Token auch noch benutzt werden kann, wenn der Browser geschlossen oder der PC neugestartet wurde. Hierdurch bilden sich einige Vorteile:

1. Automatischer Login: Das Frontend liest direkt beim Öffnen den lokalen Speicher des Browser, erkennt den Token und führt einen automatischen Login durch.

2. Einmalig und einzigartig: Der Token wird im Gegensatz zu einem Passwort nur einmal genutzt und ist einzigartig, weswegen er die Sicherheit maßgeblich erhöht, da das Passwort nur sehr selten genutzt werden muss.

3. Zeitliche Begrenzung: Im Gegensatz zu einem Passwort, kann ein Token ablaufen, in diesem Projekt ist ein Token bis zu sieben Tage gültig. Ist er abgelaufen, muss der Nutzende sich erneut mit Spezifika und Passwort authentifizieren.

[58] vgl. Sandhu und Samarati, 1996, S.241 f.

4. Einweg-Verschlüsselt: Token können gleich wie Passwörter mittels Hashing verschlüsselt werden, so dass ein Angreifender keine Authentifizerungsdaten aus der Datenbank lesen kann.

Beispiel der Token-Authentifizerung

Angenommen der Nutzende ist bereits in dem Besitz eines Tokens, was der Regelfall ist, wenn sich mindestens einmal eingeloggt wurde, dann erkennt das Frontend automatisch, dass sich im lokalen Speicher ein Token befindet, welcher dann für die Erneuerung des Tokens genutzt werden kann:

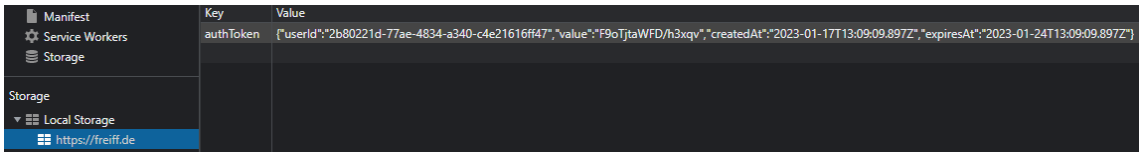

Abbildung 11: Vorhandener Token im lokalen Speicher des Browsers

Dieser Token wird direkt von dem Frontend ausgelesen und in einer Anfrage im userStore direkt an das Backend gesendet, um sich damit zügig zu authentifizieren und den Token und die Nutzerdaten, wie Rollen, somit zu erneuern:

```
async loginWithTokenFromLocalStorage({ commit, dispatch }, token) {
    try {
      const { data } = await
          this.$axios.post(`${backendURL}/user/tokenLogin`, {
        tokenData: {
          userId: token.userId,
          value: token.value,
        },
      })
      ...
    } catch (error) {
        ...
    }
  },
```

In diesem Code-Beispiel ist direkt zu Beginn der Methode die Anfrage an das Backend zu finden, diese richtet sich in der Produktivumgebung an die Backend URL 'https://server.freiff.de/backend/user/tokenLogin', hierbei wird die Anfrage

jedoch nicht leer geschickt, sondern es wird die UserId und der Token-Wert mitgegeben, da diese für das Backend benötigt werden, um den Nutzenden zu verifizieren, ähnlich wie zum Beispiel es auch bei Email und Passwort der Fall ist. Ist der Aufruf der 'Action' erfolgreich, erhält das Frontend eine umfangreiche Antwort:

```
{
    "id": "2b80221d-77ae-4834-a340c4e21616ff47",
    "username": "Frederik",
    "email": "frederikreiff@freiff.de",
    "auth": {
        "authToken": {
            "userId": "2b80221d-77ae-4834-a340-c4e21616ff47",
            "value": "F9oTjtaWFD/h3xqv",
            "createdAt": "2023-01-17T13:09:09.897Z",
            "expiresAt": "2023-01-24T13:09:09.897Z"}
        },
    "roles": [...],
    ...
}
```

Diese Antwort enthält zum einen einen neuen Token und somit auch einem neuen Wert, aber auch einem neuen Ablaufdatum. Des Weiteren enthält die Antwort zusätzliche Informationen zu dem Nutzenden, wie dessen Email und Nutzername, aber auch dessen Rollen und weiteres. Diese Antwort hat das Frontend bis hierhin lediglich erhalten und zwischengespeichert, die Daten sind allerdings noch flüchtig und werden daher in dem State des userStores abgelegt. Hierfür kommen die Mutationen des userStores in Einsatz, hier als Beispiel für die Token Informationen:

```
// Aufruf der Mutation
commit('SET_TOKEN', data.auth.authToken)

=================

// Implementation der Mutation
SET_TOKEN(state, token) {
    state.token = token
    localStorage.setItem('authToken', JSON.stringify(token))
},
```

Nun wurde der Token genutzt, um sich anzumelden und der neue Token wurde überall hinterlegt, damit andere Teile der Applikation ihn nutzen können und der Nutzende bei dem nächsten Aufruf automatisch eingeloggt wird.

4.1.5 Einstellungen und Themes

Bereits in der Projektplanung stellte sich heraus, dass das Projekt einen großen Mehrwert erhalten würde, wenn es Konzepte zur Individualisierung des eigenen Dashboards gibt. Dennoch muss jede Individualisierung abgewägt werden, da sie auch immer die Komplexität erhöht. Nach sorgfältiger Abwägung hat das Team sich dazu entschieden, dass es ein Theme geben soll, welches alle Farben des Dashboard ansprechen kann, sogar bis in die Widgets hinein. Des Weiteren, soll es möglich sein, Widgets auszutauschen. Was allerdings erstmal nicht umgesetzt werden soll, sind unterschiedliche Layouts für die Widget-Anordnung.
Ein Theme ist eine Sammlung an einzelnen Eigenschaften in Form eines JSON-Objekts, hierbei wird ein Theme in logisch getrennten Einheiten geführt und jede Eigenschaft enthält einen Wert, der seinem CSS Pardon entspricht. Damit ein Theme angezeigt wird, muss das JSON-Objekt in das CSS injektiert werden, so dass die einzelnen Angaben Bezug zu ihren Elementen erhalten. Hierbei werden zwei Wege genutzt:

1. Direkte Injektion: Bei der direkten Injektion wird einem HTML Element ein Style Objekt hinzugefügt, welches direkte CSS-Ausrufe enthält, wie *outline: 'none'*.

2. Indirekte Injektion: Bei der indirekten Injektion erhält ein HTML Element ebenfalls ein Style Objekt, dieses jedoch setzt nur Variablen, die von dem statischem CSS genutzt werden können, so dass der statische CSS Ausdruck die dynamischen Inhalte des Style Objekts enthält.

Beide Methoden haben in bestimmten Situation Vorteile, so bietet die direkte Variante die Möglichkeit, dass einzelne Themes zusätzliche CSS-Zeilen beinhalten, obwohl diese in dem Standard-Theme nicht enthalten sind. Bei der indirekten Variante kann man hingegen Animationen und Transformationen erstellen ohne dabei die Dynamik zu verlieren. Dies könnte zwar auch durch die direkte Variante realisiert werden, würde aber enormen Aufwand erzeugen.

Es gilt also festzuhalten, dass beide Methoden die Injektion mittels eines Style-Objektes durchführen. Jedes veränderliche Element erhält somit ein eigenes Objekt in dem *:style* Tag, wie sich im DOM-Beginn des Dashboards erkennen lässt :

```
...
  <div class="page-frame" :style="currentTheme.page">
    <div class="background-image"
        :style="currentTheme.pageBackground"></div>
    <DashboardNavBarDB :style="currentTheme.navBarLeft" />

    <div class="dashboard" :style="currentTheme.dashboard"
        v-if="loggedIn">
    ...
    </div>
  </div>
...
```

Aus dem Quelltext lässt sich bereits ableiten, dass alle Style-Tags mit einem Objekt gefüllt werden, welches sich in einer Variable currentTheme befindet. Das currentTheme ist jedoch nicht nur eine Variable, sondern eine 'computed property':

```
...
  computed: {
    ...
    ...mapFields('settingsStore', ['themes', 'currentTheme']),
  },
...
```

Hierbei wird mittels der mapFields Funktion auf den *settingsStore* zugegriffen, damit hieraus die Variable *currentTheme* aus dem State gelesen wird. Der Vorteil der 'computed property' ist es, dass wenn sich in dem Store der State ändert, dies direkt für eine Neuberechnung der 'computed property' führt. Durch diese Neuberechnung, ändern sich die Style Objekte in dem „Document Object Model“ (DOM) und somit aktualisiert sich bei Änderung des Themes nicht nur logisch, sondern auch visuell, die graphische Oberfläche.

Die Berechnung des aktuellen Themes gestaltet sich hingegen etwas komplexer, denn es soll möglich sein, dass ein Theme gespeichert werden kann, dennoch sollte kein Theme gespeichert sein, soll das Standard-Theme benutzt werden. Damit diese Funktion vorhanden ist, wurde folgender Quelltext entwickelt:

```
...
```

```
  calculateCurrentTheme({ state, commit })
    // Check if the userDefinedTheme from the settings array is in the
       name property of any theme in the themes array and if so set it
       as the currentTheme and if not calculate the currentTheme based
       on the time of day
    let userDefinedTheme = state.settings.find((setting) => {
      return setting[0] === 'userDefinedTheme'
    })
    let theme = state.themes.find((theme) => {
      return theme.name === userDefinedTheme[1]
    })
    if (theme) {
      commit('SET_CURRENT_THEME', theme)
    } else {
      let date = new Date()
      let hours = date.getHours()
      if (hours >= state.dayThemeStart && hours < state.dayThemeEnd) {
        commit('SET_CURRENT_THEME', state.themes[0])
      } else {
        commit('SET_CURRENT_THEME', state.themes[1])
      }
    }
  },
...
```

In der Methode, die das aktuelle Theme berechnet, wird zunächst geschaut, ob es in dem Einstellungs-Objekt, welches der Nutzende beim Login vom Server erhält, eine Einstellung mit dem Schlüssel 'userDefinedTheme' gibt. Ist diese Einstellung vorhanden, wird der darin gespeicherte Wert, also zum Beispiel 'DefaultDay' oder 'PinkishLight', gelesen und geschaut, ob das gespeicherte Theme eins der vorhandenen Themes ist. Ist dem so, wird das Theme aus der Liste aller Themes raussortiert und in dem Store als aktuelles Theme gesetzt. Ist dem nicht so, erhält der Nutzende das Standard-Theme, welches es in zwei Ausprägungen gibt, der Tag- und der Nachtvariante. Welche der beiden Ausprägungen nun in den Store geschrieben wird, ist abhängig von der Uhrzeit, wie in der letzten If-Bedingung zu erkennen ist.

Konsolidierung

Auch wenn diese Ausschnitte einen guten Einblick in die Funktionalität geben, bilden sie zusammen jedoch nur einen kleinen Ausschnitt. Werden alle hier genann-

ten Elemente mit den weiteren Frontend-Implementationen zusammengefügt, bildet sich eine zusammenhängende Nutzererfahrung mit Registrierung, (Auto-)login und natürlich dem Dashboard mit dessen dazugehörigen Bestandteilen, wie mehrerer verschiedenen Widgets. Durch diese Zusammenführung ergibt sich ein solides Frontend, mit vernünftigem Datenmanagement, dem Potenzial zur Skalierung und der Möglichkeit der Integration weiterer Anforderungen.

4.2 Umsetzung: Backend

Web-Applikationen brauchen eine Form von zentralem Ort für die Datenverarbeitung und -speicherung. Dieses Projekt benötigt dies unter verschiedenen Aspekten, so müssen alle Nutzerdaten zentral gespeichert und verwaltet werden. Zu dem bedarf es einer Authentifizierung, einer Autorisierung mit Rollen und natürlich auch das Bereitstellen von Schnittstellen für die Applikationsdaten, wie den Todos der einzelnen User.[59] Damit das Backend nach guten Standards errichtet wird, wurde sich hier dafür entschieden, das Javascript Framework Nest.js, eine Node.js Absplitterung, einzusetzen. Dieses bietet eine gute Architektur für die Erstellung von REST-APIs, sowie einfache Wege für die Nutzung von Datenbankmanagementsystemen, wie MySQL oder PostgreSQL.

Datenbankkonzept

Da das Backend sowohl Dateien als auch Daten verarbeitet, müssen die Informationen an einem zentralen Ort abgelegt werden, hierfür nutzen wir das Datenbankmanagementsystem MySQL. Da MySQL ein relationales Datenbankmodell vorsieht, passte es sehr gut zu den Anforderungen des Projekts, so dass eine Skizze der Datenbank samt ihrer Tabellen und Spalten erstellt wurde, die das Projekt von der Datenperspektive beschreibt.[60]

[59] vgl. Sandhu und Samarati, 1996, S.241 f.
[60] vgl. Steiner, 2009, 11 ff.

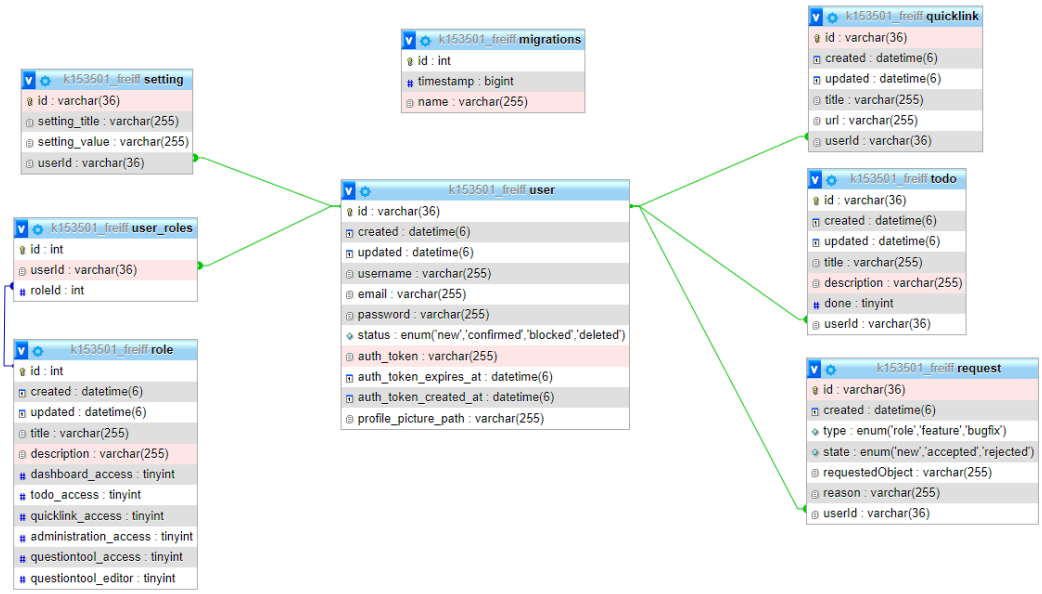

Abbildung 12: Relationaler Datenbankentwurf

Backend Konzept

Die Idee hinter der Nest.js Architektur ist es, die Software-Architektur nach den Entitäten aufzuteilen, so dass das Backend einen Controller für Nutzende, Rollen, Todos, etc. hat. Dies ermöglicht eine logische Separation von Abläufen, was die Skalierbarkeit, Effizienz und auch die Sicherheit erhöht. Die einzelnen Module die daraus entstehen werden in dem Wurzel Dokument 'app.module.ts' gebündelt.

```
@Module({
  imports: [
    ConfigModule.forRoot({
      ...
    }),
    TypeOrmModule.forRoot({
      ...
    }),
    BlogModule,
    UserModule,
    TodoModule,
    RoleModule,
  ],
  ...
})
export class AppModule {}
```

Hierbei stellt als Beispiel das 'UserModule' alle Komponenten für die Backend-Nutzerverwaltung bereit, darunter gehört der 'UserController' mit allen öffentlich erreichbaren Schnittstellen, der 'UserService', der die Logik hinter den öffentlichen, aber auch internen Schnittstellen bereitstellt. Zudem ist hier die Entität 'User' definiert, also das Datenbankobjekt, welches die Applikation in jenes übersetzt.

```
@Entity('user')
export class UserEntity {
  @PrimaryGeneratedColumn('uuid') id: string;
  @CreateDateColumn() created: Date;
  @CreateDateColumn() updated: Date;

  @Column() username: string;
  @Column() email: string;
  @Column() password: string;

  @Column({
    type: 'enum',
    enum: ['new', 'confirmed', 'blocked', 'deleted'],
    default: 'new',
  })
  status: UserState;

  @Column({
    default: null,
  })
  auth_token: string;
  @CreateDateColumn() auth_token_expires_at: Date;
}
```

All diese Komponenten existieren auf ähnlicher Art und Weise auch bei den anderen Entitäten, wie der Todo-Entität oder der Rollen-Entität.

Backend Logik am Beispiel des Todo Moduls

Das Todo Modul ist ein Modul, dessen Logik an sich auf den ersten Blick recht einfach zu seien scheint, allerdings ist es eines der Module, die von allen Aspekten Gebrauch machen: Ist der User eingeloggt, hat der User die entsprechenden Rechte

und ist die Anfrage korrekt gestellt? Aus diesem Grund muss das Todo Modul neben dem eigenen Service auch die Module für den User und die Rollen importieren.

```
@Module({
  imports: [TypeOrmModule.forFeature([TodoEntity]), RoleModule,
      UserModule],
  controllers: [TodoController],
  providers: [TodoService],
  exports: [TodoService],
})
export class TodoModule {}
```

Der Todo Controller, der die öffentlich erreichbaren Schnittstellen hat, besteht aus drei Grundmethoden, eine Methode für das Anlegen eines Todos, eine für das Löschen eines Todos und eine für das Bereitstellen einer Liste der Todos für einen Nutzer oder eine Nutzerin. Jede dieser Methoden bekommt entsprechende Argumente im Body Element der Webanfrage mitgeliefert und leitet diese an die entsprechende Methode mit der passenden Logik im Service weiter.

```
@Controller('backend/todo')
export class TodoController {
  constructor(private readonly todoService: TodoService) {}

  @Post('createTodoForUser')
  createTodoForUser(@Body('todoData') todoData: object): any {
    return this.todoService.createTodoForUser(todoData);
  }

  @Post('getUsersTodos')
  getUsersTodos(@Body('todoData') todoData: object): any {
    return this.todoService.getTodosForUser(todoData);
  }

  @Post('removeTodoForUser')
  removeTodoForUser(@Body('todoData') todoData: object): any {
    return this.todoService.removeTodoById(todoData);
  }
}
```

Todo Service

Der Todo Service beinhaltet alle drei Methoden aus dem Controller, am Beispiel der 'createTodoForUser' Methode, die ein Todo für einen User anlegt, kann sehr gut abgeleitet werden, wie das Backend funktioniert.

Zunächst wird mit einer ausgelagerten Funktion geprüft, ob die API-Anfrage den Anforderungen des Backends gerecht wird, also ob alle Felder die benötigt werden auch mitgeliefert werden, dazu gehört in dem Beispiel der Token, die User-ID und Informationen bezüglich des zu erstellenden Todos.

```
async createTodoForUser(todoData): Promise<any> {
  // Check if all required keys are present in the request
  checkRequestForDataAndTheirKeys(todoData, requiredTodoCreateKeys);
  checkRequestForDataAndTheirKeys(todoData.todo,
     requiredTodoCreateTodoKeys);
  checkRequestForDataAndTheirKeys(todoData.token, requiredTokenKeys);
```

Im weiteren Verlauf wird nun geprüft, ob der mitgelieferte Token dem gespeicherten Token in der Datenbank entspricht, ist dem der Fall, dann ist der Nutzende authentifiziert und das Backend hat die Identität des Nutzenden bestätigt. Da diese Authentifizierung in der Hoheit des Nutzer-Moduls steht, übernimmt der entsprechende Nutzer-Service diese Aufgabe in einer dafür geschriebenen Methode.

```
  // Check if token is valid and refresh it
  const authToken = await this.userService.refreshToken({
    userId: todoData.token.userId,
    tokenValue: todoData.token.tokenValue,
  });
```

Nach dem der Nutzende authentifiziert wurde, muss für die Aktion die er vornehmen will, nämlich das Hinzufügen eines Todos, geprüft werden, ob er die Rechte besitzt. Das Recht, welches der User in dem Produkt braucht heißt 'todo_access'. Diese Aufgabe liegt erneut nicht bei dem Todo Modul, da die Autorisierung in dem Rollen Modul passiert. Daher wird hierfür der entsprechende Rollen Service genutzt. Hat er die Rechte nicht, wird die Anfrage mit einem Fehler, einer HttpException, abgebrochen.

```
  // Check if user has permission to create a todo
```

```
if (
  await !this.roleService.hasUserCertainPermission(
    todoData.userId,
    'todo_access'
  )
) {
  throw new HttpException(
     ...
  );
}
```

Nach der Authentifizierung und der Autorisierung ist sichergestellt, dass bekannt ist, wer der Nutzende ist und ob er die Rechte hat, ein Todo anzulegen. Somit kann nun die eigentliche Erstellung eines Todo Objektes durchgeführt werden, die geschieht auf Basis der Todo Entität. Anschließend wird die Datenbank aufgefordert, das neu erstellte Objekt zu speichern.

```
// Create todo
const todo = this.todoRepository.create({
  userId: todoData.userId,
  title: todoData.todo.title,
  description: todoData.todo.description,
});
await this.todoRepository.save(todo);
```

Die primäre Aufgabe der Methode ist an der Stelle getan, jedoch wurde bei dieser Aktion der Token des Users erneuert, weshalb das Frontend eine Antwort des Backend benötigt, die den aktualisierten Token beinhaltet. Des Weiteren erhält der User noch die Bestätigung des Erfolgs der Methode und erhält das tatsächlich eingefügt Todo.

```
const returnObject = {
  todo: todo,
  auth: {
    authToken: authToken,
  },
};
```

```
    return returnObject;
```

Mit dem 'return' ist die Anfrage abgearbeitet und das Backend kann nun weitere Anfragen bearbeiten. Durch das Prüfen der Identität mittels eines Token und der Rollen eines Users, wird gewährleistet, dass alle Vektoren, die auf Daten führen, kontrolliert und geprüft geschehen. Dieser Ansatz wurde im ganzen Backend umgesetzt und betrifft alle öffentlich erreichbaren Schnittstellen.

4.3 Automatisches Deployment mit Github Actions

Das Produkt, welches durch das Team in Testumgebungen entwickelt wurde, soll schon während der Entwicklung auf Live-Systemen gehostet werden. Da das Team sich für einen agilen Ansatz entschieden hat, macht es Sinn, dass fertiggestellte Features kontinuierlich in Form von Produkt-Inkrementen in diese Live-Umgebung übersetzt werden. So kann frühestmöglich das minimal funktionsfähige Produkt direkt zu der Zielgruppe geliefert werden und diese können die Entwicklung neuer Features somit direkt erleben.
Damit das Produkt allerdings tatsächlich Live ist, braucht es erstmal die entsprechenden Systeme. So hat das Team sich dazu entschieden einen Apache Webserver für das Hosting des Frontends zu nutzen. Für das Backend wurde ein virtueller Server aufgesetzt, der mit Docker-Containern das Backend bereitstellt. Die Datenbank wurde auf einem dedizierten Datenbankserver eingerichtet. Alle Server wurden mit Firewalls verbunden und mittels dem Einbau von Reverse Proxies erhielt sowohl das Backend, als auch das Frontend die Möglichkeit eine SSL Verschlüsselung einzusetzen, damit die Webseite und das Backend mit dem Applikationsprotokoll https erreichbar sind. Dies sorgt nicht nur für eine verschlüsselte Kommunikation, sondern verstärkt die Sicherheit der Kommunikation massivst, weshalb es heute Standard ist, dass Browser nur noch https Seiten anzeigen.

Frontend Deployment

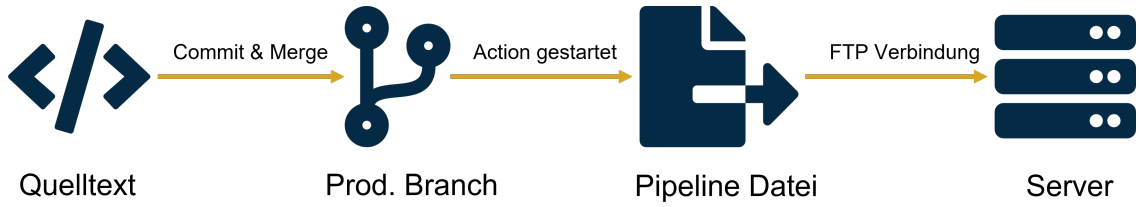

Abbildung 13: Ablauf des Frontend Deployments

Damit ein Nuxt Projekt vom Quelltext bis hin zur laufenden Umgebung kommt, bedarf es einiger Schritte, die dazwischen liegen. Zunächst einmal muss der neu geschriebene Quelltext in das geteilte Github Repository hochgeladen werden, ist dies abgeschlossen integriert der neue Quelltext sich mit dem bereits vorhandenen Quelltext. Dies alles geschieht auf einem 'Dev' Branch, also ein Abschnitt nur für die Entwicklung. Ist ein Feature abgeschlossen und kann in die Live-Umgebung überführt werden, so werden jene Commits in einer Merge-Anfrage gebündelt. Sind nun alle relevanten Änderungen in der Merge Anfrage vorhanden, wird diese freigegeben und der sich auf dem 'Prod' Branch befindene Quelltext wird um die Änderungen ergänzt. Sobald nun der 'Prod' Branch die neuen Features mit in dem Quelltext enthält, wird eine sogenannte Action gestartet.

```
name: Pushing to freiff.de Frontend
on:
  push:
    branches:
      - prod

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Use Node.js 16.x
        uses: actions/setup-node@v3
        with:
          node-version: 16.x
          cache: 'npm'
      - run: echo "${{ secrets.HOSTING140583_SECRET }}" >
          connection_string.txt
      - run: chmod 400 connection_string.txt
      - run: npm install
      - run: npm run generate
      - run:
          ssh -o StrictHostKeyChecking=no -i ./connection_string.txt
              hosting140583@freiff.de "
          rm -r ~/httpdocs &&
          mkdir ~/httpdocs"
      - run: scp -i ./connection_string.txt -r ./dist/*
```

```
hosting140583@freiff.de:/httpdocs/
```

In dem Definitions-Quelltext lassen sich sehr gut die Schritte der Automatisierung ableiten. Zunächst einmal wird in dem 'on' Block definiert, unter welchen Bedingungen eine Action ausgeführt werden soll, in dem Fall ist zu erkennen, dass diese Action gestartet wird, wenn ein 'push' auf den 'prod' Branch geschieht.
Ist dies der Fall, wird der Job 'deploy' gestartet, hierbei wird eine virtuelle Umgebung in dem Fall eines Ubuntu Systems aufgezogen. Anschließend werden auf diesem Ubuntu System die unter 'steps' definierten Schritte abgearbeitet. Zunächst wird sichergestellt, dass Node.js und der Paketmanager NPM installiert sind, damit alle für das eigentliche Skript notwendigen Bedingungen vorhanden sind.
Damit sich die laufende Action am Server authentifizieren kann, kann sie ein Secret nutzen, welches als Umgebungsvariable mitgegeben wurde. Dieses Secret wird mittels des 'echo' Befehls in eine neue Text Datei mit dem Namen connection_string.txt geschrieben. Da das Secret eine vertrauensvolle Information darstellt, wird die Datei mittels des 'chmod' Befehls nur für den Besitzenden mit Leserechten versehen, so dass die Gruppe oder alle anderen keine Lese oder gar Schreibrechte haben. Da das Nuxt.js Projekt sowohl die eigenen Abhängigkeiten hat, welche durch weitere im Projekt definierte Module ergänzt werden, müssen diese auch in der virtuellen Ubuntu Umgebung installiert werden. Hierfür werden mit der Befehlskombination *npm install* mit dem Paketmanager NPM alle Abhängigkeiten installiert. Darauffolgend wird mit NPM die statische Webseite generiert, der Befehl wird nach seiner Ausführung nun in der Ordnerstruktur einen dist-Ordner mit den produktionsbereiten Dateien erstellen.
Nun liegen also die Daten, die auf dem Webserver benötigt werden in dem dist-Ordner. Damit diese von der virtuellen Ubuntu Umgebung auf den Server kommen und funktionieren, muss zunächst sichergestellt werden, dass sich dort keine alten Dateien befinden und anschließend müssen alle neuen Dateien zu dem Webserver kopiert werden. Für das Aufräumen der alten Dateien, wird eine SSH-Sitzung auf dem Webserver gestartet, hierbei wird sich mit der connection_string.txt Datei authentifiziert. In der SSH-Sitzung wird nun der httpsdocs Ordner gelöscht und neu angelegt, so dass nun ein leerer Ordner für die neuen Dateien existiert. Der letzte Teil der Pipeline beginnt nun, hier wird der Secure-Copy Befehl 'scp' genutzt, welcher ähnlich wie die SSH-Sitzung arbeitet, jedoch sich die generierten Dateien in dem dist-Ordner in den auf dem Webserver befindlichen httpsdocs Ordner kopiert. Ist diese Kopie abgeschlossen, befindet sich die neue Variante des Frontends auf dem Webserver und ist Live.

Durch dieses automatische Deployment des Frontends kommen einige Vorteile für die Qualität und Umsetzung des Projekts. Durch die Automatisierung wird viel manuelle Arbeit gespart, da alle diese Schritte sonst manuell durchgeführt werden müssten, des Weiteren kann durch diese Variante eine Klassifizierung der Änderungen erfolgen, um nur vollständige und gewünschte Features in die Live-Umgebung zu übernehmen. Ein weitere Vorteil der Automatisierung ist, dass diese die Fehleranfälligkeit reduziert und einen transparenten Ablaufplan enthält.

Backend Deployment

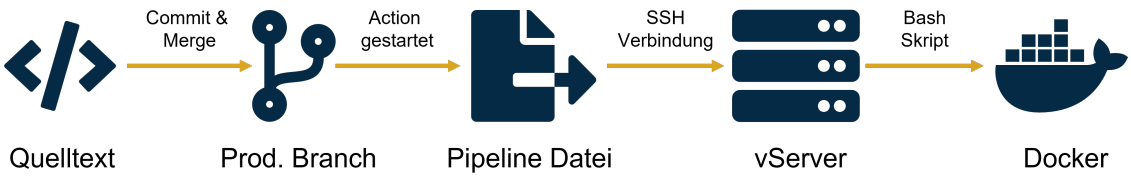

Abbildung 14: Ablauf des Backend Deployments

Das Backend, welches mit Nest.js entwickelt wurde, basiert auf eine Node.js Sprache und benötigt dafür im Bereich Hosting nicht nur einen Webserver, wie Apache oder Nginx. Es braucht eine dedizierte Node.js Umgebung, auf der das Backend bereitgestellt wird. Da das Backend auf einem Server laufen soll, der noch für weitere Zwecke verwendet wird, hat das Team sich dazu entschieden Docker zu verwenden und das Backend in Form eines Docker Images bereitzustellen. Dies bedeutet, dass der Quelltext nicht nur auf den Zielserver muss, sondern das nach jeder Veränderung am Quelltext auch ein neues Docker Image bereitgestellt werden muss, welches dann im Anschluss ohne Datenverlust das veraltete Image ersetzt. Damit auch hier die große Menge an Schritten automatisiert erfolgt, wurde auch hier eine Github Action formuliert:

```
name: Pushing to freiff.de Backend
on:
  push:
    branches:
      - prod

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
```

```
    - run: echo "${{ secrets.V2202211141451206233_SECRET }}" >
        connection_string.txt
    - run: chmod 400 connection_string.txt
    - run:
        ssh -o StrictHostKeyChecking=no -i ./connection_string.txt
            root@v2202211141451206233.megasrv.de "
        cd /var/docker_compose_files/freiff.de_main_backend/ &&
        chmod +x docker-setup-script.bash &&
        ./docker-setup-script.bash"
```

In dieser Action-Datei lässt sich zunächst ähnlich wie bei dem Frontend erkennen, dass diese nur ausgeführt wird, wenn ein Push beziehungsweise in dem Fall Merge von dem 'Dev' Branch zu dem 'Prod' Branch erfolgt. Nur unter dieser Bedingung startet der 'deploy' Job. Da auch diese Action sich am Ende auf ein Zielsystem verbinden muss, wird ebenfalls die Authentifizerung mit einem Secret gelöst, dieses wird der Action als Umgebungsvariable mitgeliefert. Auch hier wird dieses Secret nun der Action in Form einer Textdatei zur Verfügung gestellt und anschließend mit dem Befehl 'chmod' die Berechtigung zum Lesen der sensibelen Datei eingeschränkt. Anschließend wird eine SSH-Sitzung mit dem Zielserver eröffnet, dann wird sich mit Hilfe der Pipeline/ Action in einen bestimmten Ordner begeben, in dem die Docker Informationen abgelegt werden. Hier befindet sich nun ein selbst geschriebenes Skript drin, welches mit 'chmod' ausführbar gemacht wird und anschließend gestartet wird. Das Skript ist der Schlüssel dazu, wie das Docker Image erstellt und deployt wird:

```
#! /bin/bash

echo "Pulling latest changes from git..."
git reset --hard
git pull origin prod

echo "Building docker image..."
docker build -t freiff-node-backend .

echo "Stopping and removing old container..."
docker stop freiff-backend-live
docker rm freiff-backend-live

echo "Starting new container..."
docker-compose up -d
```

```
docker ps # Optional
echo "Done!"
```

Zunächst ist der erste Schritt, dass alle Änderungen, die am Quelltext erfolgt sind mit Git auf den Zielserver gelangen. Hierfür werden gecachte Inhalte mit *git reset –hard* entfernt und der aktuelle Stand des Quelltextes auf dem 'Prod' Branch wird mit *git pull origin prod* heruntergeladen. Nach diesem Schritt ist der Quelltext auf der Maschine lokal vorhanden und es wird ein Docker Image mit der Befehlskombination *docker build* erzeugt. Dieses Docker Image muss nun noch in die Umgebung eingespielt werden, hierfür wird der aktuell laufende Docker-Container gestoppt und entfernt. Anschließend wird mit 'docker-compose' ein neuer Container mit dem neuen Image gestartet. Ab diesem Schritt ist die Action durchgelaufen.

Jedoch geht aus dieser Action eines noch nicht hervor, nämlich wie der Docker-Container gebaut wird und wie verhindert wird, dass die Daten, wie Profilbilder bei entfernen des Containers nicht gelöscht werden. Wie der Container gebaut werden soll geht aus einer Dockerfile heraus, diese ist standardisiert für Node.js Umgebungen und wird daher nicht im Detail betrachtet. Dennoch wurde durch das Team die Docker-Compose Datei verfasst, die beschreibt, mit welcher Konfiguration der Docker-Container gestartet wird:

```
version: '3'
services:
  backend:
    container_name: freiff-backend-live
    image: freiff-node-backend
    restart: unless-stopped
    ports:
      - 8000:8000
    environment:
      - DB_USER=***MASKED_VARIABLE***
      - DB_PASSWORD=***MASKED_VARIABLE***
      - DB_NAME=***MASKED_VARIABLE***
      - DB_URL=***MASKED_VARIABLE***
      - DB_IP=***MASKED_VARIABLE***
      - DB_PORT=3306
      - DB_PROTOCOL=mysql
```

```
    - DEPLOYMENT_TARGET=PROD
  volumes:
    - ./uploads:/usr/src/app/uploads
```

In dieser Konfigurationsdatei lässt sich der Container-Name ablesen und hier ist auch erkennbar, dass das aktuellste Image, welches im Kontext der Action frisch mit der Dockerfile erstellt wurde, verwendet werden soll. Des Weiteren soll der Container immer laufen, selbst nach Neustart des Systems, es sei denn, er wird manuell gestoppt. Anschließend werden die Ports konfiguriert, was vor allem für den Einsatz des Reverse Proxies wichtig ist. Im direkten Anschluss erfolgen sämtliche Konfigurationen für die Docker-Container Umgebung, die benötigt werden, zum Beispiel die Datenbankanmeldedaten. Der letzte Block ist ein sehr wichtiges Kernelement, der Docker-Container erhält hier ein sogenanntes Volume, welches an sein Arbeitsverzeichnis gehangen wird. Wird der Container gelöscht, bleibt der Ordner './uploads' weiterhin bestehen und kann von komplett neuen Docker-Containern genutzt werden, sofern diese im gleichen Arbeitsverzeichnis starten was durch '/usr/src/app/uploads' spezifiziert wird. Somit kann gewährleistet werden, dass alle Daten, die nicht in der Datenbank liegen, wie zum Beispiel Profilbilder, nicht bei Versionsupdates verloren gehen.

Mit all den einzelnen Dateien und Skripten wird eine Automatisierung erzeugt die binnen einer Minute eine neue Version zuverlässig und transparent erstellt. Diese Automatisierung sorgt dafür, dass eine große Summe an Befehlen nicht manuell gestartet werden müssen und erleichtert somit nicht nur die Entwicklung, sondern diese Automatisierung war auch ein essentieller Teil, das Projekt rechtzeitig mit allen minimalen Features live zu schalten. Insgesamt dauert die Entwicklung einer Automatisierung deutlich länger und es bedarf auch umfangreicher Tests, allerdings kann sich bereits durch die Github Dashboards ableiten, dass durch die Automatisierung all die genannten Schritte mit Faktor 52 erspart wurden, da die Pipeline insgesamt 52 mal erfolgreich Backend Updates eingespielt hat, ohne dabei ein einziges mal fehlzuschlagen oder auf unerwartete Probleme zu stoßen.

5 Reflexion und Fazit

Für das Projektteam war es von hoher Wichtigkeit, nach dem Abschluss des Projektes zu reflektieren, wie die Zusammenarbeit und die Erstellung des Produktes

ablief. Dies hebt Stärken hervor und zeigt Schwächen auf, die in folgenden Projekten verbessert werden können.

Auf der Seite der positiven Aspekte ist besonders hervorzuheben, dass das Produkt rechtzeitig und im vollen Funktionsumfang fertig war und dies gerade mit dem Aspekt der teils unklaren Anforderungen zeigt, dass die Projektplanung hier für einen reibungslosen Ablauf gesorgt hat. Ebenfalls hat sich besonders gezeigt, dass ein Prototyp die Anzahl der Design-Umstrukturierungen während der Erstellung reduziert, was besonders auf den Aspekt der Zeitaufwände einspielt. Im Bereich der Zeitaufwände war es ebenfalls sehr positiv, dass sich das Team dazu entschieden hat, alle Deployment-Schritte zu automatisieren, auch wenn dies einige Stunden im Bereich des Wissensaufbaus in Anspruch genommen hat. Alleine durch die Automatisierung konnten mindestens 400 Minuten für redundante Arbeit eingespart werden und somit war das schnelle Liefern von Funktionalitäten an die Nutzer und Nutzerinnen erst möglich. Im Bereich der Technologien waren die eingesetzten Frameworks für Frontend und Backend richtig gewählt, es gab wenig Mehraufwände und beide Frameworks haben den Usecase erfolgreich und gut abbilden können.

Jedoch gibt es auch zwei Aspekte, die bei folgenden Projekten verbessert werden können. Zunächst könnte im Bereich der Projektplanung der Fokus noch etwas mehr auf die Produktgestaltung und die Features fallen. Das Team sah sich noch während der Erstellung damit konfrontiert, dass einige Features eigentlich ein Must-Have sind und das zum Beispiel eine Einstellungsseite doch von Vorteil wäre. Dies hat für Veränderungen an der Anforderungsliste gesorgt, was im agilen Umfeld sehr gut abzufangen ist, jedoch im Vorhinein durch besseres Prototyping und zusätzliche Feature-Diskussion reduziert werden könnte. Der zweite Aspekt, der in Zukunft verbessert werden kann ist eher technischer Natur, denn es wurde bei späteren Tests ein schwerwiegender Sicherheitsfehler entdeckt, der zwar rechtzeitig bekannt wurde und es niemals in die Produktionsumgebung geschafft hat, der jedoch gar nicht erst hätte entstehen sollen. Diese Sicherheitslücke wurde erst durch Zufall entdeckt, als das Team mit automatisierten Tests experimentiert hat. Dies hat dem Team gezeigt, dass es von sehr hoher Wichtigkeit ist, dass immer Testfälle in Zukunft definiert werden sollten, die sofort bemerken, wenn durch neue Features unerwünschte Seiteneffekte auftreten. Insofern sollte in Zukunft nicht mehr auf das automatisierte Testen verzichtet werden und dieses sogar durch den Einsatz von zum Beispiel Test-Driven-Development in den primären Fokus gelangen.

Insgesamt bewertet das Team das Projekt als äußerst gelungen und beherzigt die Verbesserungsmöglichkeiten. In Zukunft wären Features, wie eine individuelle Widget-Anordnung denkbar, sowie die Ergänzung weiterer Widgets, um die User Experience weiter zu verbessern und die Zielgruppe auszuweiten. Das Team freut sich über das fertige Produkt und ist ambitioniert, die aufgelisteten Potentiale auch noch in Zukunft zu erfüllen.

Literatur- und Quellenverzeichnis

Bartonitz, M. (2018). Speed Estimation–viele User Stories in kurzer Zeit schätzen. In *Agile Verwaltung* (S. 113–117). Springer.

Blas, M. J., Leone, H., & Gonnet, S. (2020). Modeling and simulation framework for quality estimation of web applications through architecture evaluation. *SN Applied Sciences*, *2*(3), 1–21.

Botify. (o. D.). Client-Side vs. Server-Side JavaScript Rendering: Which Is Better for SEO? https://www.botify.com/blog/client-side-server-side-rendering-seo

Christoforakos, L., & Diefenbach, S. (2017). Erfolgreiches Prototyping im Ideenstadium der Produktentwicklung. In S. Hess & H. Fischer (Hrsg.), *Mensch und Computer 2017 - Usability Professionals*. Gesellschaft für Informatik e.V. https://doi.org/10.18420/muc2017-up-0193

Cohn, M., & Beck, K. (2019). User stories. *Mountain Goat Software*.

Epping, T. (2011). *Kanban für die Softwareentwicklung*. Springer-Verlag.

Eremit, B., Weber, K. F., Eremit, B., & Weber, K. F. (2016). SMART-Methode–Specific Measurable Accepted Realistic Timely. *Individuelle Persönlichkeitsentwicklung: Growing by Transformation: Quick Finder—Die wichtigsten Tools im Business Coaching*, 93–99.

Fielding, R. T. (1999). Software architectural styles for network-based applications. *Unpublished manuscript*.

Fleming, J., & Koman, R. (1998). *Web navigation: designing the user experience*. O'reilly Sebastopol, CA.

Freeman, A. (2018). Using a Data Store. In *Pro Vue.js 2* (S. 505–540). Apress. https://doi.org/10.1007/978-1-4842-3805-9_20

Harnal, S., & Chauhan, R. (2020). Efficient and Flexible Role-Based Access Control (EFRBAC) Mechanism for Cloud. *EAI Endorsed Transactions on Scalable Information Systems*, *7*(26), e1–e1.

Harwardt, M. (2020). Digitalisierung in Deutschland – Der aktuelle Stand. In M. Harwardt, P. F.-J. Niermann, A. M. Schmutte & A. Steuernagel (Hrsg.), *Führen und Managen in der digitalen Transformation: Trends, Best Practices und Herausforderungen* (S. 17–34). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-28670-5_2

Herfurtner, A. (2003). *Screendesign*. Pearson Deutschland GmbH.

Hohberger, P. (2018). SCRUM – Die „neue“ Art des Projektmanagement. In S. Grote & R. Goyk (Hrsg.), *Führungsinstrumente aus dem Silicon Valley: Konzepte und Kompetenzen* (S. 115–128). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-54885-1_7

Holt, E.-M., Winter, D., & Thomaschewski, J. (2011). Personas als Werkzeug in modernen Softwareprojekten. *Tagungsband UP11*.

IBM. (2022). What is an Application Programming Interface (API) | IBM. https://www.ibm.com/topics/api

Kok, L. T. (2020). *Hands-on Nuxt. js Web Development: Build universal and static-generated Vue. js applications using Nuxt. js*. Packt Publishing Ltd.

Kolko, J. (2010). *Thoughts on interaction design*. Morgan Kaufmann.

Kyriakidis, A., Maniatis, K., & You, E. (2016). *The majesty of Vue. js*. Packt Publishing.

Melnyk, V., Ashok, V., Puzis, Y., Soviak, A., Borodin, Y., & Ramakrishnan, I. (2014). Widget classification with applications to web accessibility. *Web Engineering: 14th International Conference, ICWE 2014, Toulouse, France, July 1-4, 2014. Proceedings 14*, 341–358.

Morris, P. W. G. (2000). *Researching the unanswered questions of project management*.

npm Inc. (2022). NPM. https://www.npmjs.com/

Overflow, S. (2022). Most used web frameworks among developers worldwide, as of 2022. https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/

Persson, M. (2020). JavaScript DOM Manipulation Performance: Comparing Vanilla JavaScript and Leading JavaScript Front-end Frameworks.

Pšenák, P., & Tibenskỳ, M. (2020). The usage of Vue JS framework for web application creation. *Mesterséges intelligencia*, *2*(2), 61–72.

Rabe. (2023). Internetnutzer - Anteil Weltweit nach regionen 2022. https://de.statista.com/statistik/daten/studie/162074/umfrage/penetrationsrate-des-internets-nach-regionen-im-jahr-2010/

Red-Hat. (2022). What is an API? https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces

Sandhu, R., & Samarati, P. (1996). Authentication, access control, and audit. *ACM Computing Surveys (CSUR)*, *28*(1), 241–243.

Schweibenz, W. (2004). Zielgruppenorientiertes interaktionsdesign mit Personas. *Information Wissenschaft und Praxis*, *55*(3), 151–158.

Steiner, R. (2009). *Grundkurs Relationale Datenbanken* (Bd. 7). Springer.

Varbanov, Z., & Filipov, L. (2020). STATIC GENERATION OF WEBSITES–POSITIVES AND NEGATIVES. *Balkan Journal of Applied Mathematics and Informatics*, *3*(2), 25–30.

Vivenzio, A., & Vivenzio, D. (2013). Vorgehensmodelle in der Softwareentwicklung. In *Testmanagement bei SAP-Projekten: Erfolgreich Planen • Steuern • Reporten bei der Einführung von SAP-Banking* (S. 5–10). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-8348-2142-3_2

Vyas, R. (2022). Comparative Analysis on Front-End Frameworks for Web Applications. *International Journal for Research in Applied Science and Engineering Technology*, *10*(7), 298–307.

Wang, H., Cao, J., & Zhang, Y. (2005). A flexible payment scheme and its role-based access control. *IEEE Transactions on knowledge and Data Engineering*, *17*(3), 425–436.

Weber, M. (2021). Definition Projekt. In *Fahrplan für Projektmanagement in sechs Schritten* (S. 3–8). Springer.

You, E. (2022a). Vue Introduction | Vue Docs. https://vuejs.org/guide/introduction.html#what-is-vue

You, E. (2022b). What is Vuex? | Vue Docs. https://vuex.vuejs.org/

Ehrenwörtliche Erklärung

Hiermit versichern wir, dass die vorliegende Arbeit von uns selbstständig und ohne unerlaubte Hilfe angefertigt worden ist, insbesondere dass wir alle Stellen, die wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen sind, durch Zitate als solche gekennzeichnet habe. Wir versichern auch, dass die von uns eingereichte schriftliche Version mit der digitalen Version übereinstimmt. Weiterhin erklären wir, dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde / Prüfungsstelle vorgelegen hat. Wir erklären uns damit einverstanden, dass die Arbeit der Öffentlichkeit zugänglich gemacht wird. Wir erklären uns damit einverstanden, dass die Digitalversion dieser Arbeit zwecks Plagiatsprüfung auf die Server externer Anbieter hochgeladen werden darf. Die Plagiatsprüfung stellt keine Zurverfügungstellung für die Öffentlichkeit dar.

Zusätzlich erklären wir uns mit der Vergabe einer gemeinsamen Note einverstanden.

Mülheim, den 20.02.2023	Frederik Reiff
(Ort, Datum)	(Eigenhändige Unterschrift)
Duisburg, den 20.02.2023	Nassim Doukkali Doukkali Attar
(Ort, Datum)	(Eigenhändige Unterschrift)
Oberhausen, den 20.02.2023	Jan Bendel Jan Bendel
(Ort, Datum)	(Eigenhändige Unterschrift)