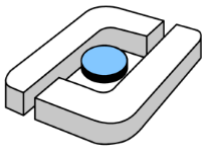


**Hochschule Osnabrück**

University of Applied Sciences

**Fakultät**

**Ingenieurwissenschaften und Informatik**


Schriftliche Ausarbeitung zum Thema:

# Auftragsverwaltungssystem

im Rahmen des Moduls
Webanwendungen
des Studiengangs Informatik-Medieninformatik


| | |
|---|---|
| Autoren: | Jan Weimer, Nelson Morais |
| Matr.-Nr.: | 868487, 879551 |
| E-Mail: | jan.weimer@hs-osnabrueck.de, nelson.morais@hs-osnabrueck.de |
| Dozent: | Björn Plutka |


Abgabedatum: 09.03.2022

# Inhaltsverzeichnis

# Abbildungsverzeichnis

## Source-Code Verzeichnis

# 1 Einleitung (Jan Weimer)

## 1.1 Vorstellung des Themas

Das Thema der Projektarbeit handelt von der Realisierung eines Auftragsverwaltungssystems in Form einer nativen App auf einem mobilen Endgerät. Hier können Teams eines Unternehmens eine Übersicht über vorhandene und neue Aufträge bekommen. Hier besteht die Möglichkeit Aufträge von einem schwarzen Brett zu untersuchen und anzunehmen. Dabei werden die angenommenen Aufträge in der Teamseite gezeigt und verschwinden vom schwarzen Brett. Außerdem soll die App ein sogenanntes Threadboard besitzen, womit andere Unternehmensmitglieder sich miteinander verständigen können. Dazu können die Profildetails jederzeit verändert werden.

## 1.2 Zielgruppe

Zielgruppe dieser App sind Unternehmen, die von Kunden Aufträge entgegennehmen. Dabei ist es nicht wichtig um was für eine Art von Unternehmen es sich handelt. Optimaler Weise wird die App von Firmen genutzt, die eine Arbeitsweise in Teams pflegen wollen.

## 1.3 Aufbau des Berichts

Zuerst werden alle Grundlagen erklärt, die zum Verständnis der Realisierungsweise des Projekts nötig sind. Danach wird auf das Konzept eingegangen. Was ist der Plan zum Aufbau der Oberfläche? Anschließend werden dann alle Schritte erklärt, die gemacht wurden, um das Ergebnis zum Projektende zu erreichen. Abschließend wird ein Fazit aus dem Projekt gezogen.

# 2 Darstellung der Grundlagen (Jan Weimer)

## 2.1 Technische Grundlagen

### Vue.js

*Abbildung 1: Vue.js Logo*

Vue.js ist ein clientseitiges JavaScript-Framework, welches sehr leichtgewichtig ist und komponentenweise arbeitet. Dadurch können bessere Ordnerstrukturen geschaffen werden und ausgelagert werden, welches der Software-Architektur zugutekommt. Der Code wird dadurch lesbarer und besser wartbar. Die Leichtgewichtigkeit resultiert in der winzigen Downloadgröße von ca. 18 KB.

Die Flexibilität von Vue.js stellt jedoch auch Risiken dar. Dadurch kann es passieren, dass in Unternehmensteams viele Entwickler einen anderen Codingstil haben und dadurch andere Probleme bekommen, diese Realisierungsschritte nachzuvollziehen. Demnach sind intern festgesetzte Codingrichtlinien eine gute Möglichkeit das Risikoausmaß zu minimieren. Beispielsweise kann festgehalten werden das Listen und Tabellen immer in Komponenten ausgelagert werden müssen.

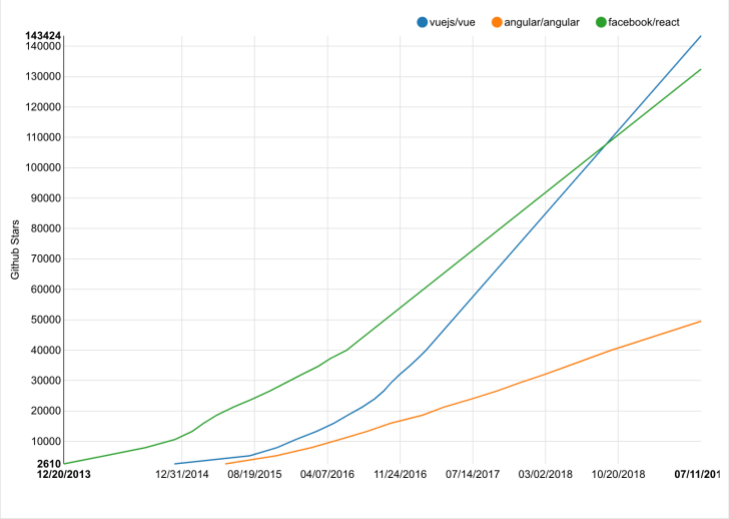


*Abbildung 2: Popularität zwischen Frameworks über Zeit*

### Quasar

*Abbildung 3: Quasar Logo*

Quasar ist ein Framework, welches auf Vue.js basiert. Der Entwickler bekommt dadurch eine Richtlinie, die dabei helfen sollen, eine benutzerfreundliche Oberfläche zu erstellen, die dazu noch responsiv ist. Da die Anwendung am Ende auf mobilen Endgeräten laufen soll, kann hier Capacitor genutzt werden. Der Vorteil liegt darin, dass der Code nur einmal geschrieben werden muss und diese App in egal welcher Art rausgebracht werden kann.

## 2.2 Konzept

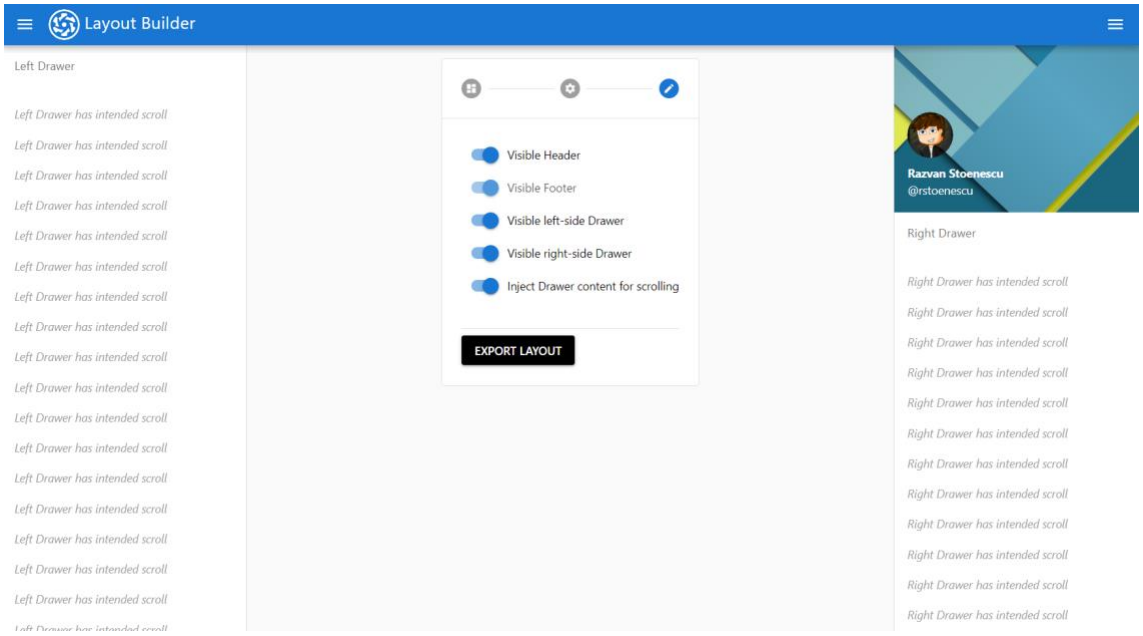


*Abbildung 4: Layout Builder von Quasar*

Mit dem Layout Builder kann das Layout erstellt werden, welches für die Anwendung verwendet werden soll. Hier wurde sich für ein visible Header und ein left- und right-side Drawer entschieden. Der Gedanke darin besteht den left-side Drawer als Leiste zu nutzen, um auf die gewünschten Threads zu kommen. Der right-side Drawer soll die Menüspalte sein, wo man auf das Profil, Team und schwarze Brett gelangen kann.

Innerhalb der App befinden sich die Pages, wo hin- und wegnavigiert werden kann. Möglichst soll vieles als Component realisiert werden, um eine bessere Trennung zu schaffen.

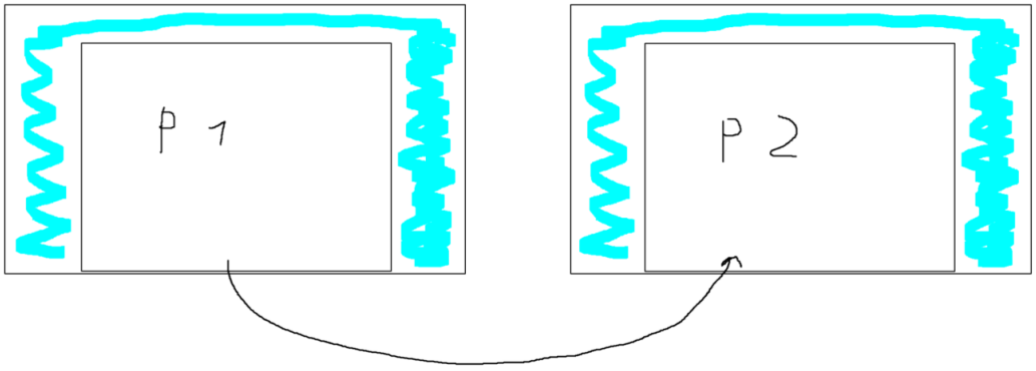


*Abbildung 5: Routingbeispiel: hier ist das Feld in blauer Farbe das Layout, welches sich nicht ändert. Lediglich wird die Seite geändert.*

# 3 Anwendung

## 3.1 Layout (Nelson Morais)

Das Layout-Konzept der Anwendung basiert auf einer "View"-Eigenschaft, die Quasar implementiert. Diese Eigenschaft ermöglicht die Anpassung des Layout-Containers und der Schubladen, einschließlich Kopf- und Fußzeile, durch einfaches Ändern eines Strings.

```
<q-layout view="hHh LpR lFr" class="main-frame">
```

*Snippet 1: Quaser Layout mit der Prop Ansicht*

Der Hauptcontainer der Anwendung dient allein dem Zweck, eine Seite anzuzeigen, wenn sich die Route ändert, wobei der Rest der Benutzeroberfläche intakt bleibt, während die Seiten gewechselt werden, um eine einseitige Anwendung zu erhalten. Dies ist nur möglich, wenn der Vue-Router als untergeordnete Komponente verwendet wird.

```
<q-page-container>
  <router-view/>
</q-page-container>
```

*Snippet 2: Hauptcontainer und untergeordnete Komponente*

Standardmäßig wird beim Starten der Anwendung nur der Inhalt dieses Containers angezeigt, einschließlich des Inhalts, der durch den view prop string konfiguriert wurde. In diesem Fall verfügt die Anwendung über eine Kopfzeile, die sich immer über dem aktuellen Inhalt befindet, der vom Page container angezeigt wird, sowie über die Schubladen, wenn diese geöffnet sind.

```
<q-drawer v-model="rightDrawerOpen" side="right" bordered>
```

*Snippet 3: Quaser drawer*

Die Layout-Schubladen sind standardmäßig geschlossen, so dass der Inhalt der Page container beim Start der Anwendung im Vordergrund steht. Hier werden die Aufträge verwaltet, dies ist das Hauptmerkmal der Anwendung.

## 3.2 Assignment Manager (Nelson Morais)

Als Teamleiter wird der Benutzer beim Start der Anwendung mit der Seite zur Verwaltung von Aufträgen begrüßt. Auf dieser Seite werden die aktuellen Aufgaben, an denen gearbeitet wird,

und ihre jeweiligen Fälligkeitstermine sowie offene Aufgaben, an denen kein Team arbeitet, und abgeschlossene Aufgaben durch das Team des Benutzers angezeigt.

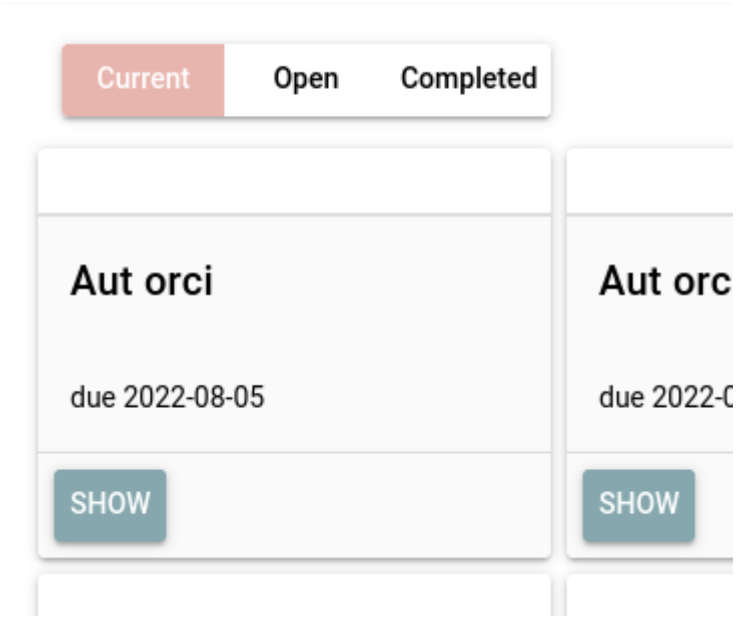


*Abbildung 6: Auftragsseitenfilter und Auftragskarte*

Die Aufträge werden als Karten aufgelistet, die nur die für die Übersicht relevanten Informationen enthalten, um die Benutzeroberfläche nicht zu überfüllen. Detaillierte Informationen zu den Aufgaben können durch Drücken der Schaltfläche "Show" abgerufen werden. Hier öffnet sich ein Modal mit einer detaillierten Beschreibung der Aufgabe.

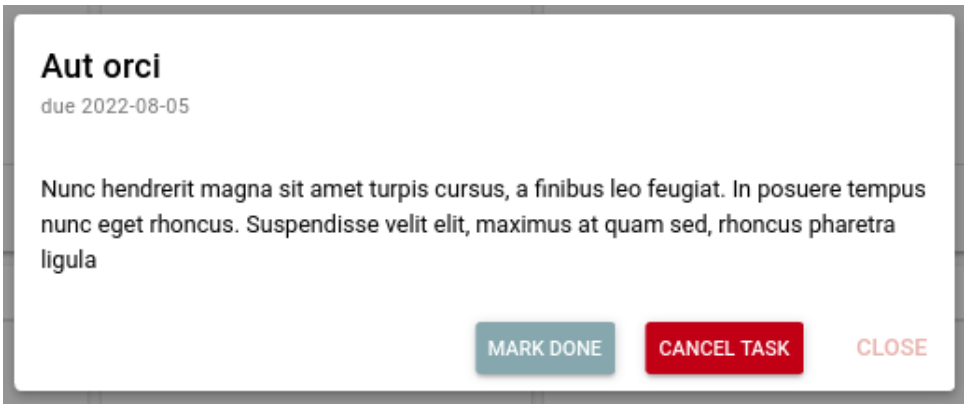


*Abbildung 7: Modal für Auftragsdetails*

Die Interaktionsmöglichkeiten mit einer Aufgabe hängen vom Status der Aufgabe ab. Ein aktueller Auftrag, d.h. ein Auftrag, der vom Team des Benutzers bearbeitet wird, hat 3 Schaltflächen: "Mark Done", "Cancel Task", "Close", ein offener Auftrag hat nur eine Schaltfläche "Add" und eine Schaltfläche "Close". Erledigte Aufträge bieten nur die Schaltfläche “Close” an.

```
methods: {
  async fetchData(){
    const person = await fetch(this.backendUrl + "/persons?username=" + this.pro-
totypeUser).then(res => res.json())
    this.checkAdd = person[0].teamId !== this.obj.teamId;
    this.checkCurrent = person[0].teamId === this.obj.teamId && this.obj.assign-
mentStatus === "ACCEPTED";
    this.checkDone = person[0].teamId === this.obj.teamId && this.obj.assign-
mentStatus === "DONE";
  },
```

*Snippet 4: FetchData() Methode mit Controllern*

Die Logik für die angezeigten Optionen ist in der "AuftragLink"-Komponente implementiert, wodurch sichergestellt wird, dass die übergeordnete Komponente "AuftragTable" bei der Verwendung des Filters die auftrags Interaktionsoptionen nicht berücksichtigen muss, wie in "Abbildung 5" dargestellt.

```
<AuftragLink :obj="props.row" @eventClicked="getDataa"/>
```

*Snippet 5: Aufruf der Komponente "AuftragLink" mit Parametern und Abhören von Ereignissen*

Der an die untergeordnete Komponente übergebene Parameter ist das Auftragsobjekt, das in der Tabelle angezeigt wird. Die Übergabe dieser Informationen an die untergeordnete Komponente ermöglicht es dem "AuftragLink" die benötigten Objektwerte abzurufen, um sie in der Karte sowie im Modal anzuzeigen und die logischen Prüfungen für die verfügbaren Optionen durchzuführen. Mit Hilfe von "@eventClicked" hört die übergeordnete Komponente der untergeordneten Komponente zu und wartet darauf, dass die untergeordnete Komponente ein Ereignis ausgibt. Diese Methode wurde verwendet, um die Möglichkeit zu gewährleisten, die in der Tabelle angezeigten Daten erneut abzurufen, so dass die Benutzeroberfläche für die Tabelle mit den aktualisierten Daten erneut angezeigt wird, wenn der Benutzer einen Auftrag innerhalb des Modals akzeptiert oder abbricht.

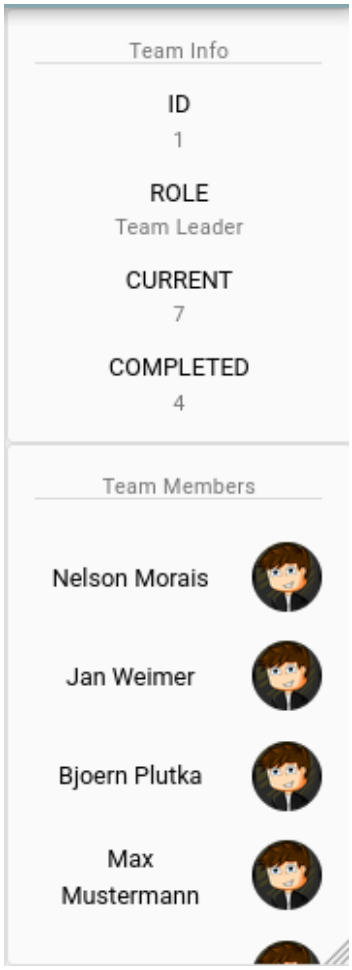


*Abbildung 8: Assignment Sideboard*

Auf der Seite zur Verwaltung von Aufträgen gibt es neben der Auftragstabelle ein Sideboard, das dem Teamleiter relevante Teaminformationen anzeigt. Bei diesen Informationen handelt es sich um die Team-ID, die Rolle, die der Benutzer im Team innehat, die aktuelle Anzahl der vom Team bearbeiteten Aufträge und die Anzahl der Aufträge, die das Team abgeschlossen hat. Unterhalb des Abschnitts mit den Teaminformationen befindet sich eine Liste mit den Mitgliedern des Teams des Benutzers.

## 3.3 Drawers (Jan Weimer)

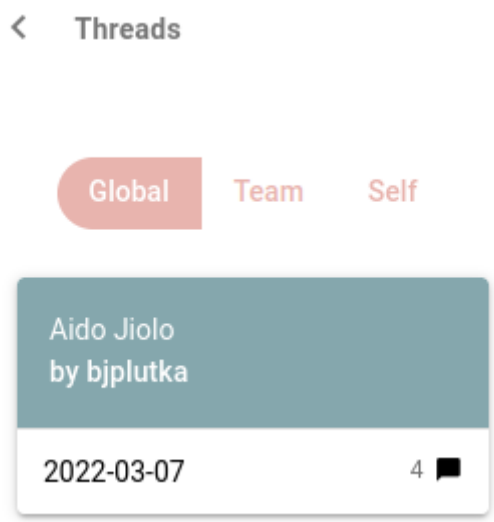


*Abbildung 9: Left-side Drawer*

Die linke Leiste zeigt alles was um Threads handelt. Hier sollen mit ToggleButtons nur globale, Team- und eigene Threads jeweils separat angezeigt werden können. Der ThreadLink zeigt den Titel des Threads und das Datum. Der Button „ADD“ ist die ThreadForm-Komponente. Durch das Klicken auf dieser öffnet sich ein Dialog mit dem Inhalt des Formulars zur Erstellung eines Threads.

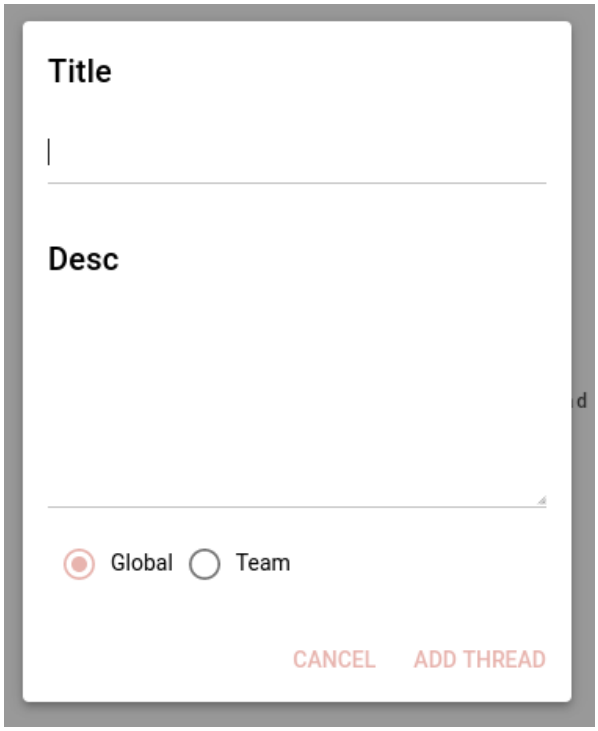


*Abbildung 10: Formular Thread erstellen*

Hier kann man den Titel, die Beschreibung und aussuchen, ob es global oder teamintern sichtbar sein soll.

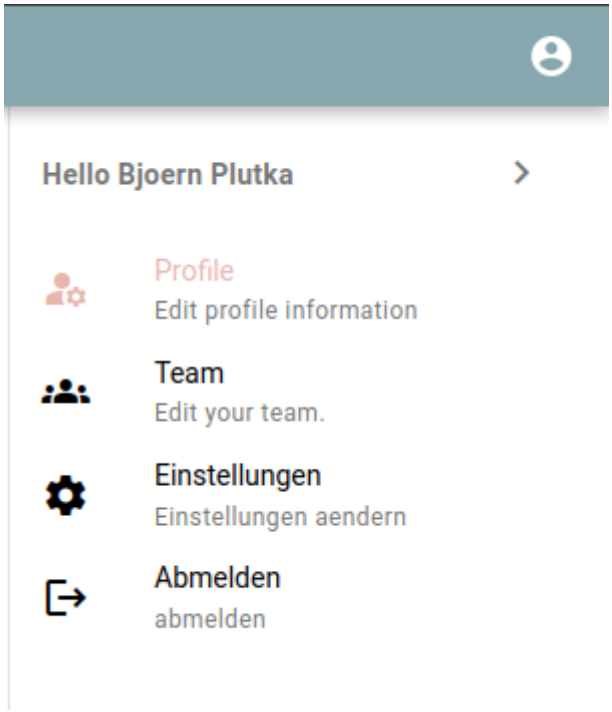


*Abbildung 11: Right-side Drawer*

Hier soll die Navigation zwischen den Pages stattfinden. Oben wird der Name des Nutzers dargestellt. Darunter sind die EssentialLinks, die Titel, Untertitel, Icon und Link beinhalten. Die Icons werden von material.io benutzt und werden auch von quasar unterstützt.

## 3.4 Threads (Nelson Morais)

Die Thread-Logik ist das zweite Hauptmerkmal der Anwendung. Diese Funktion soll sicherstellen, dass eine Form der Kommunikation zwischen Teammitgliedern sowie zwischen Teams möglich ist. Wenn der Benutzer die linke Schublade öffnet, wird er mit einer völlig neuen Benutzeroberfläche konfrontiert. Die Seite zur Verwaltung des Auftrags macht einer neuen Seite Platz, die dem Benutzer die Möglichkeit gibt, von anderen Benutzern erstellte Threads zu lesen und zu beantworten, die linke Schublade ist ein scrollbares Element, das dem Benutzer globale, Team- oder eigene Threads sowie die Anzahl der Kommentare und das Veröffentlichungsdatum anzeigt. Die Schublade gibt dem Benutzer auch die Möglichkeit, ein neues Thema zu erstellen, indem er eine schwebende Schaltfläche mit einer kontrastreichen Farbe zu den Akzenten der Anwendungs-UI verwendet.

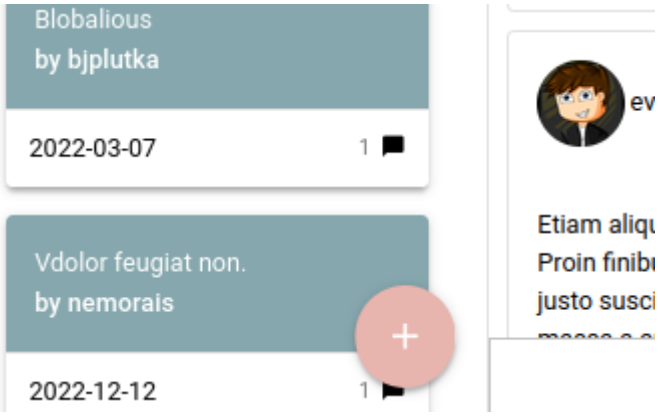


*Abbildung 12: Add Thread Button*

Wenn ein Thread angeklickt wird, wird die Threadseite aktualisiert und zeigt den Titel des Threads, den Autor und den Inhalt an. Innerhalb dieser Seite wird eine Komponente zum Beantworten des Threads angezeigt. Wenn eine Antwort übermittelt wird, entfaltet sich eine Kette von Ereignissen, um eine reaktive Umgebung zu gewährleisten.

```
async submitAnswer() {
  const person = await fetch(this.backendUrl + "/persons?username=" + this.proto-
typeUser).then(res => res.json()).then(json => json[0])
  const res = await fetch(this.backendUrl + "/answers", {
    method: 'POST',
    headers: {
      "content-Type": "application/json",
      "x-access-token": "token-value"
    },
    body: JSON.stringify({
      creator: this.prototypeUser,
      content: this.content,
      threadId: this.thread.id,
      personId: person.id
    })
  })
  this.content = ref('')
  this.answers = await fetch(this.backendUrl + "/threads/" + this.id + "/an-
swers").then(res => res.json())
  this.componentKey +=1;
  this.refresh = true;

},
```

*Snippet 6: "submitAnswer()" Methode, die die reaktive Logik enthält*

Wenn eine Antwort abgeschickt wird, wird das Textfeld gelöscht indem der Wert der Inhaltsvariablen mit "ref(")" geändert wird, Daten werden abgerufen, eine "componentKey"-Variable wird inkrementiert und ein boolesches Flag wird auf "true" gesetzt.

```
<Thread :thread="this.thread" :key="this.componentKey"/>
<div v-for="answer in this.answers" :key="this.componentKey">
```

*Snippet 7: Untergeordneter Komponenten Thread und Antwortmatrix mit Schlüssel*

Durch die Verwendung der "componentKey"-Variable als ":key" zur Verfolgung von Änderungen werden beide Elemente neu gerendert, wenn die "submitAnswer()"-Methode die Variable erhöht und die neuen abgerufenen Daten anzeigt, was bedeutet, dass die Antwort des Benutzers sichtbar ist, sobald der Backend-Server mit den neuen Daten antwortet.

```
async updated(){
  if (this.refresh){
    window.scrollTo(0,document.body.scrollHeight*2);
    this.refresh=!this.refresh
  }
},
```

*Snippet 8: "Updated()" Methode unter Verwendung des Kennzeichens*

Anhand des von der "submitAnswer()"-Methode gesetzten Flags kann die Anwendung feststellen, ob die Ursache des Aktualisierungsaufrufs eine Antwort auf das Thema war; ist dies der Fall, blättert die Anwendung automatisch für den Benutzer nach unten, um die Antwort des Benutzers auf das Thema anzuzeigen.

## 3.5 Design (Nelson Morais)

Die Funktionen der Anwendung sind auf Produktivität ausgerichtet, daher richtet sich die Zielgruppe der Anwendung an kleine bis mittelgroße Unternehmen oder Organisationen. In diesem Fall wurde unter Berücksichtigung dieser Faktoren ein modernes Design mit Pastellfarben und subtilen, aber intuitiven Navigationselementen entworfen. Diese Designwahl spiegelt eine Software wider, die auf Produktivität ausgerichtet ist, aber dennoch eine einladende Benutzeroberfläche bietet.

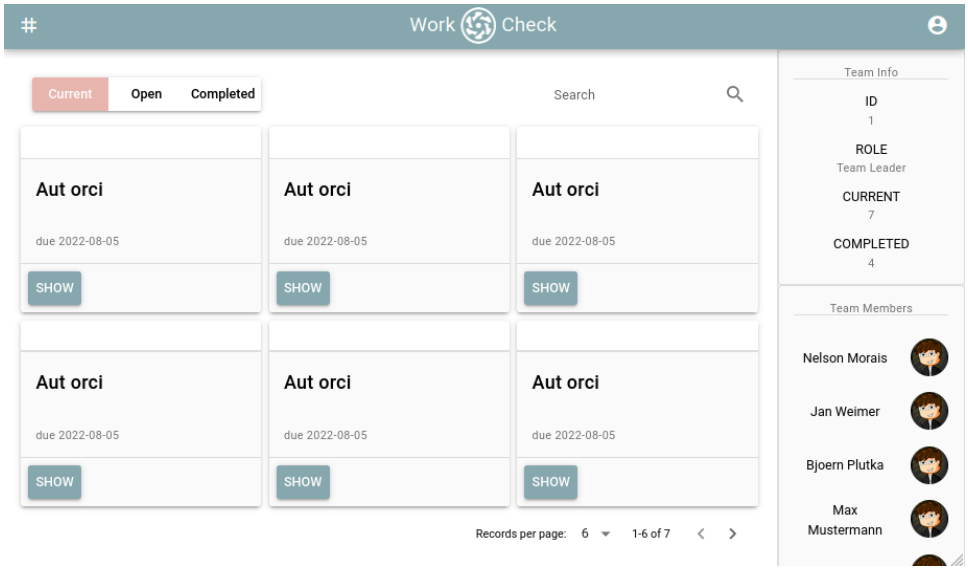


*Abbildung 13: Landing page der Applikation*

Die Anwendung ist für die Verwendung mit Tablets gedacht und zeigt nur die Informationen an, die für die aktuell angezeigte Seite relevant sind. Dadurch wird sichergestellt, dass die Benutzeroberfläche übersichtlich bleibt und dennoch genau die Informationen liefert, die für den Nutzer, der die aktuelle Seite durchsucht, erforderlich sind.

# 4 Zusammenfassung und Fazit (Jan Weimer)

Am Ende wurde eine App realisiert, worauf das Design auf Tablets angepasst ist. Es ist ein funktionierendes Frontend für ein Threadboard entstanden. Dabei wird unterteilt und gefiltert in global, teamintern und eigene Threads. Außerdem ist die Main-Page das Schwarze Brett mit allen offenen Aufträgen. Dazu kann ein Auftrag angenommen und untersucht werden. Beim Besuchen der ProfilPage, werden die Daten aus der API geladen, um die Werte für Name und Team zu füllen. Darüber hinaus können diese im Frontend verändert, wodurch mit Betätigung des Zuständigen Buttons, die neuen Werte zur API zurückgesendet wird. Um welchen Nutzer es sich handelt wird anfangs in den Properties festgelegt.

Die Arbeit mit Vue.js und dem dazugehörigen Framework Quasar war zufriedenstellend. Der rasante Anstieg der Popularität konnte durch die erstmalige Arbeit im Rahmen dieses Projektes nachvollzogen werden. Die einzelnen Komponenten sind schnell und separat realisierbar. Nach der Auffassung der Autoren nach, ist die Einschätzung, dass der Trend des exponentiellen Popularitätsanstieg so weitergehen wird.