Smart Knowledge Capture:

Filter Editor

Bachelorarbeit

Frühlingssemester 2023

Studiengang Informatik
OST – Ostschweizer Fachhochschule

Autoren: Daniel Frick & Zvonimir Serkinic
Betreuer: Prof. Dr. Frieder Loch

Abstract

Boolesche Ausdrücke, welche auf mathematischen Grundlagen, nämlich der booleschen Algebra, basieren, sind in der Informatik allgegenwärtig. Sie können unter anderem genutzt werden, um Filterausdrücke zu definieren, wobei diese beliebig lang werden können. Die dadurch entstehende Komplexität gilt es zu reduzieren. Im Rahmen dieser Arbeit sollen deshalb Konzepte entwickelt werden, welche die Nutzenden bei der Erstellung von Filterausdrücken unterstützen.

Um das Ziel der Arbeit zu erreichen, wurden zuerst voneinander unabhängige Konzepte, welche als Teilfunktionen des zu entwerfenden Filter-Editors angesehen werden können, erarbeitet. Diese Konzepte wurden anschliessend in Form eines Figma Prototyps umgesetzt, welcher über mehrere Iterationen verbessert wurde. Dabei wurde zu Beginn jeder Iteration ein Vorschlag erarbeitet, welcher im Anschluss mit diversen Personen getestet wurde. Das erhaltene Feedback konnte jeweils für die nächste Iteration verwendet werden. Basierend auf dem entstandenen Figma Prototyp konnte dann eine Web Component implementiert werden. Dabei kamen TypeScript und Stencil.js zum Einsatz. Um die Benutzungsfreundlichkeit zu validieren, wurden Usability Tests durchgeführt. Die technische Umsetzung wurde zudem mit End-to-End Tests abgesichert.

Der am Ende gebaute Filter-Editor, in Form einer Web Component, bietet alle grundlegend notwendigen Funktionen, um einen beliebigen Filterausdruck zu erstellen. Für die Darstellung des Filterausdrucks gibt es zwei Ansichten, eine Baumansicht und eine Zeilenansicht. Die Baumansicht ermöglicht es Filterausdrücke auf einfache Weise per Drag & Drop zusammenzustellen und zu verändern.

Management Summary

Ausgangslage

Im Rahmen des Forschungsprojekts «Smart Knowledge Capture» werden Ansätze entwickelt, um die Erstellung und Pflege technischer Dokumentationen zu erleichtern. Eine grosse Herausforderung ist dabei die Beschreibung von Bedingungen für bestimmte Informationen beziehungsweise Textabschnitte. Solche Bedingungen können als Filterausdruck betrachtet werden, welcher zu einem Textabschnitt erfasst wird. Dieser Filterausdruck ermöglicht bei der Erstellung der technischen Dokumentation darüber zu entscheiden, ob der Textabschnitt in die technische Dokumentation übernommen werden soll oder nicht. Ein Filterausdruck, welcher zu einem Textabschnitt erfasst wird, könnte zum Beispiel so aussehen:

$$GearType = Manual \; \boldsymbol{AND} \; (MotorType = v6 \; \boldsymbol{OR} \; MotorType = v8)$$

Ein solcher Filterausdruck basiert auf booleschen Teilausdrücken und Verknüpfungen. Das Verständnis des gesamten booleschen Ausdrucks ist dabei für viele Personen nicht sehr intuitiv, da die booleschen Verknüpfungen (AND und OR) vom umgangssprachlichen Gebrauch abweichen. Ein weiteres Problem sind grosse Filterausdrücke mit vielen Teilausdrücken und Verknüpfungen, welche schwierig nachvollziehbar sind.

Aus diesem Grund sollen im Rahmen dieser Arbeit Ansätze entwickelt werden, um die Erstellung von Filterausdrücken zu unterstützen. Die erarbeiteten Ansätze sollen dabei in einem Prototyp implementiert werden, so dass am Ende ein Filter-Editor entsteht, welcher es erlaubt Filterausdrücke auf möglichst benutzungsfreundliche Weise zu erstellen.

Vorgehen

Zuerst wurde eine Literaturrecherche durchgeführt. Parallel dazu wurden Konzepte entwickelt, wie man Nutzende bei der Erstellung von Filterausdrücken bestmöglich unterstützen kann. Basierend auf diesen Konzepten wurde ein erster Prototyp gebaut.

Da im Rahmen der Arbeit auf menschenzentriertes Design gesetzt wurde, wurde der Prototyp über vier Iterationen iterativ evaluiert und verbessert. Dabei wurde einerseits wiederholt Feedback von Fachleuten der STAR AG eingeholt und andererseits Usertests mit Mitgliedern der technischen Redaktion der STAR AG als auch mit weiteren Testpersonen durchgeführt.

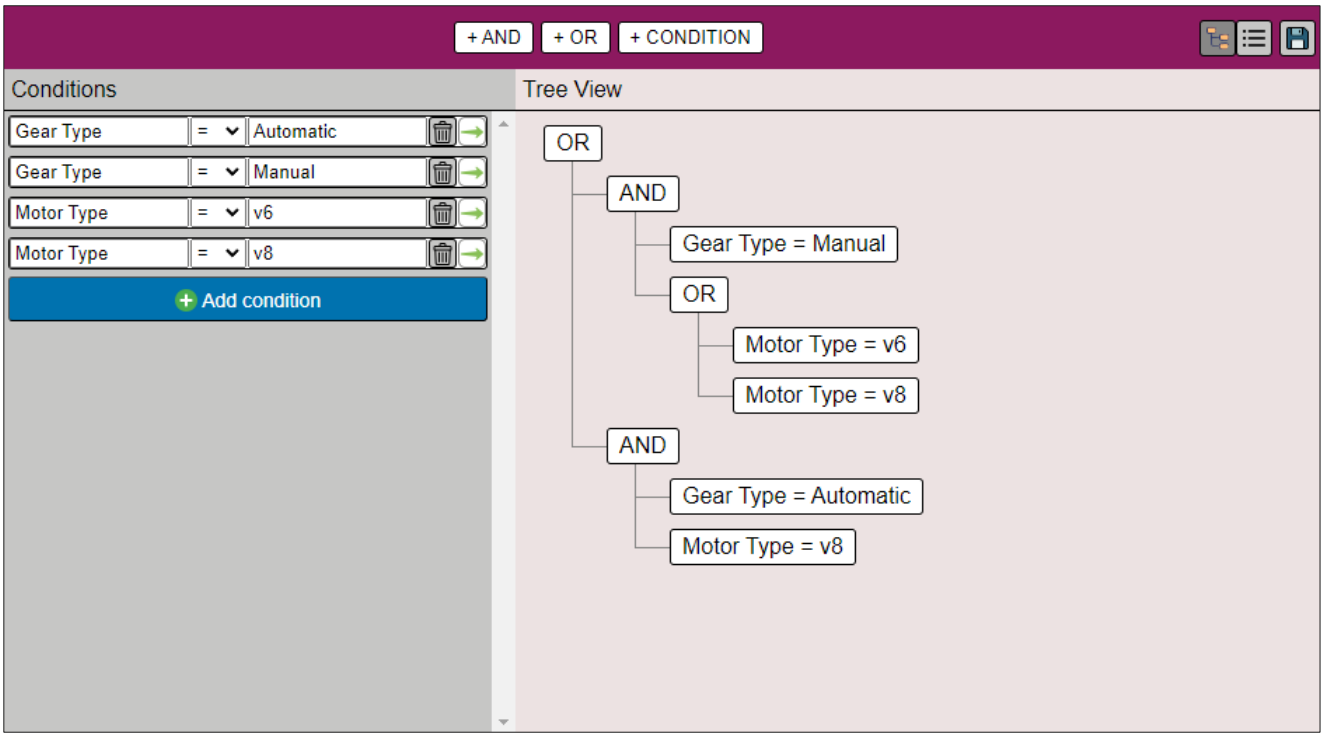
Die finale Version des Prototyps wurde anschliessend in Form einer Web Component implementiert. Gegen Ende der Arbeit wurden dann noch Usability Tests durchgeführt, um die Benutzungsfreundlichkeit des gebauten Filter-Editors zu evaluieren.

Ergebnis

Der entwickelte Filter-Editor in Form einer Web Component bietet alle grundlegend notwendigen Funktionen, um einen Filterausdruck zu erstellen. Dabei bietet der Filter-Editor neben der Baumansicht (Tree View) in der Abbildung unten, zusätzlich eine Zeilenansicht (Line View), welche den Filterausdruck in textueller Darstellung anzeigt.

Die Möglichkeit einen Filterausdruck in der Baumansicht per Drag & Drop zusammenzustellen und bearbeiten zu können, wurde dabei von allen Testpersonen als äusserst komfortabel wahrgenommen. Das Gleiche gilt auch dafür, dass die Bedingungen aus dem linken Teil wiederverwendet werden können und eine Änderung in einer Bedingung direkt für alle verknüpften Elemente im Baum übernommen wird. Die Benutzungsfreundlichkeit des Filter-Editors wurde generell als sehr gut bewertet.

Der Filter-Editor lässt sich bezüglich Unterstützung des Nutzenden aber noch weiter verbessern. So wurden zwei weitere Konzepte entwickelt (Funktion zur Vereinfachung von Filterausdrücken, Funktion zur Validierung des Filterausdrucks anhand eines Testausdrucks), welche sehr positives Feedback erhalten haben, jedoch aus Zeitgründen nicht umgesetzt werden konnten.



Danksagung

An dieser Stelle möchten wir uns herzlich bei Prof. Dr. Frieder Loch, der unsere Bachelorarbeit betreut und begutachtet hat, bedanken. Seine konstruktive Kritik sowie seine hilfreichen Anregungen waren für uns eine grosse Unterstützung.
Des Weiteren möchten wir uns bei der STAR AG, ganz besonders bei Markus Romberg und Florian von Lepel sowie allen Mitgliedern der technischen Redaktion, welche mit uns die Usertests durchgeführt haben, für die angenehme Zusammenarbeit bedanken. Unser Dank gilt auch allen weiteren Testpersonen als auch den Korrekturlesenden.

Inhaltsverzeichnis

1 Einleitung

Benutzungsfreundliche Software wird im Rahmen des technologischen Fortschritts immer wichtiger. Dies betrifft auch komplexe Anwendungen wie zum Beispiel solche, welche fundiertes Verständnis in mathematischen Themengebieten, wie der booleschen Algebra, erfordern. In diesem Zusammenhang befasst sich diese Bachelorarbeit mit der Aufgabe einen benutzungsfreundlichen Filter-Editor im Rahmen des Forschungsprojekts «Smart Knowledge Capture», welches mit der STAR AG zusammen durchgeführt wird, zu erstellen.
Dieses Kapitel dient als Einleitung in die Bachelorarbeit. Als Erstes wird in der Problembeschreibung erklärt zu welchem Zweck ein solcher Filter-Editor benötigt wird. Anschliessend wird auf das Ziel der Arbeit eingegangen und das Vorgehen während der Arbeit beschrieben.

1.1 Problembeschreibung

Im Forschungsprojekt «Smart Knowledge Capture» werden Ansätze entwickelt, um die Erstellung und Pflege technischer Dokumentationen zu erleichtern. Das Ziel des Forschungsprojekts ist es eine leichter verständliche Benutzeroberfläche für GRIPS zu entwickeln.
Eine grosse Herausforderung ist dabei die Beschreibung von Bedingungen für einzelne Informationseinheiten (z.B. Artikel oder Textabschnitte). Diese Bedingungen können als Filterausdruck betrachtet werden, welcher es erlaubt zu entscheiden, ob eine Informationseinheit für eine bestimmte technische Dokumentation relevant ist oder nicht.
Dies lässt sich gut anhand eines einfachen Beispiels aus der Auto-Industrie nachvollziehen. Wenn zum Beispiel in einem Auto eine Automatik verbaut ist, kann es sein, dass die Wartung anders zu erfolgen hat, als wenn ein Auto eine manuelle Gangschaltung hat. Um diesem Unterschied gerecht zu werden, wird nun auf einem Textabschnitt, welcher nur für ein Auto mit verbauter Automatik relevant ist, ein Filterausdruck mit der Bedingung *«GearType = Automatic»* erfasst. Wenn nun das Handbuch für ein Auto erstellt wird, wird der Textabschnitt nur in das Handbuch übernommen, wenn es sich um ein Auto mit einem Automatikgetriebe handelt.
Dies ist nur ein vereinfachtes Beispiel zum besseren Verständnis, wie Filterausdrücke erfasst und ausgewertet werden. In der Praxis bei der STAR AG kommen auch wesentlich kompliziertere Filterausdrücke zur Anwendung. Um solche Filterausdrücke komfortabel erstellen zu können wird ein Filter-Editor benötigt, welcher im Rahmen dieser Arbeit entwickelt werden soll.

Die Erstellung von Filterausdrücken, welche auf booleschen Ausdrücken basieren, empfinden dabei viele Personen als schwierig. Ein Beispiel eines etwas komplexeren Filterausdrucks sieht folgendermassen aus:

$$GearType = Automatic \; \boldsymbol{AND} \; (MotorType = v6 \; \boldsymbol{OR} \; MotorType = v8)$$

Die Schwierigkeit einen solchen Filterausdruck zu erstellen oder zu interpretieren, liegt in den booleschen Verknüpfungen (fett markiert). Der Grund dafür ist, dass die Bedeutung der booleschen Verknüpfungen vom umgangssprachlichen Gebrauch abweichen. Dies kann anhand von einfachen Beispielen je Verknüpfungstyp nachvollzogen werden.

AND-Verknüpfung

$City = New\ York\ \boldsymbol{AND}\ City = Boston$

Wenn man beispielsweise alle Mitarbeiter, welche in New York und Boston arbeiten, erhalten möchte, dann würde man diese in der Umgangssprache mit einem AND (und) verknüpfen, da man so typischerweise die Optionen erweitert. Umgangssprachlich bedeutet dies, dass man die Mitarbeitenden haben will, welche nur in New York oder nur in Boston oder an beiden Orten beschäftigt sind. In einem booleschen Ausdruck hat das AND (und) aber eine andere Bedeutung und bedeutet, dass beide booleschen Teilausdrücke erfüllt sein müssen, wodurch man nur Mitarbeitende erhält, welche in beiden Orten beschäftigt sind. (Shneiderman (1996, S. 341))

OR-Verknüpfung

$Salad\ dressing = French\ \boldsymbol{OR}\ Salad\ dressing = Italian$

Auch die Interpretation des OR (oder) weicht vom umgangssprachlichen Gebrauch ab. «Ich möchte französische oder italienische Salatsauce» ist in der Umgangssprache ein exklusives OR (oder), bedeutet also, dass man entweder das eine oder das andere will, aber nicht beides zusammen. In einem booleschen Ausdruck hingegen ist das OR (oder) inklusiv, das bedeutet, dass es entweder das eine, das andere oder beide zusammen sein darf. (Shneiderman (1996, S. 341))

1.2 Ziel der Arbeit

Das Ziel der Arbeit ist es einen Prototyp für einen benutzungsfreundlichen Filter-Editor zu entwickeln, welcher die Erstellung von Filterausdrücken unterstützt.
Dabei sollen auch die Bedürfnisse der STAR AG berücksichtigt werden. Ein erster Einblick in GRIPS hat verdeutlicht, wie Filterausdrücke bei der STAR AG aussehen. Es hat sich gezeigt, dass aufgrund der vielen Ausstattungsoptionen für Fahrzeuge, oft sehr lange und komplexe Filterausdrücke verwendet werden. Aus diesem Grund muss der zu entwickelnde Filter-Editor darauf ausgerichtet werden, dass beliebig komplexe Filterausdrücke gebaut werden können. Im Weiteren geht es insbesondere darum herauszufinden, was für zusätzliche Hilfsfunktionalitäten der Filter-Editor bieten könnte, um den Nutzenden die Arbeit zu erleichtern.
Von der technischen Seite her gilt es zu entscheiden, mit welcher Technologie der Filter-Editor gebaut werden soll, damit er nachträglich möglichst einfach in den bestehenden Forschungsprototyp des «Smart Knowledge Capture»-Projekts, welcher mit React entwickelt wird, integriert werden kann.

1.3 Vorgehen

In einem ersten Schritt wird eine Literaturrecherche durchgeführt. Im Zentrum stehen dabei die Boolesche Algebra als auch Menschenzentriertes Design. Parallel dazu werden durch eigene Überlegungen und Recherchen in Fachliteratur Ideen entwickelt, was einen guten Filter-Editor ausmacht. Diese Ideen fliessen dann in die Entwicklung konkreter Konzepte ein. Die Konzepte, welche Teilfunktionen des zu entwerfenden Filter-Editors beschreiben, sollen dabei auch die Anforderungen der Software GRIPS, welche von der STAR AG entwickelt und verkauft wird, miteinbeziehen.
In einem nächsten Schritt werden die erarbeiteten Konzepte in einem Figma Prototyp verbaut. Der Figma Prototyp wird dabei durch Usertests evaluiert und iterativ verbessert. Die finale Version des Prototyps wird anschliessend als Web Component implementiert. Dabei werden gegen Ende der Implementierung zusätzlich Usability Tests durchgeführt, um allfällige Probleme aufzudecken und zu eliminieren. Abbildung 1 veranschaulicht das Vorgehen grafisch.

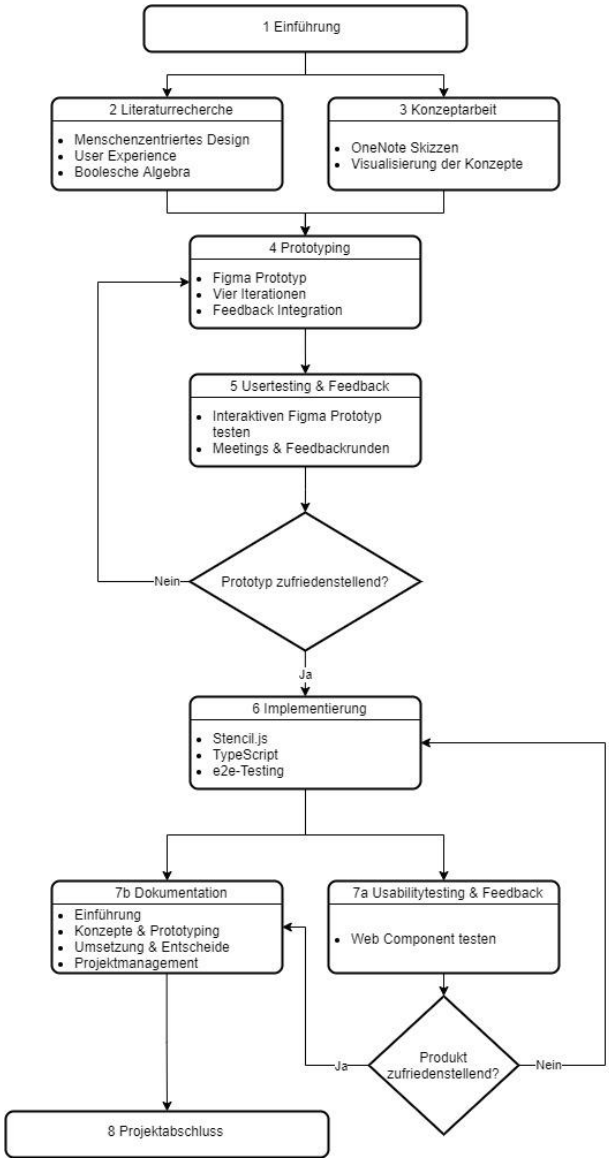


Abbildung 1: Graphische Darstellung des Vorgehens
Quelle: Eigene Darstellung

2 Theoretische Grundlagen

Die erarbeiteten theoretischen Grundlagen basieren auf einer Literaturrecherche. Als Erstes wird eine kurze Einführung in die «Boolesche Algebra» durchgeführt. Danach folgt eine ausführliche Einführung in das Thema «Menschenzentriertes Design» und im Anschluss werden wichtige Aspekte zum Thema «Usability Testing» vorgestellt. Zum Abschluss folgt dann ein Kapitel «Diskussion» in welchem darauf eingegangen wird, wie die theoretischen Grundlagen in die Arbeit einfliessen.

2.1 Boolesche Algebra

Die boolesche Algebra ist ein wichtiger Bestandteil dieser Arbeit, da sie beim Erstellen von Filtern eine zentrale Rolle spielt. Aus diesem Grund wird das Thema an dieser Stelle aus Sicht der Mathematik behandelt, wobei nur auf für die Arbeit relevante Aspekte eingegangen wird.

Boolesche Ausdrücke haben immer einen Wahrheitswert von «wahr» oder «falsch» und können miteinander verknüpft werden. Um mehrere boolesche Ausdrücke miteinander zu verknüpfen, werden sogenannte Junktoren verwendet. (Tietje (2019, S. 1-2))
Da der mathematische Begriff «Junktor» schwer zu verstehen ist, wird zum einfacheren Verständnis, für den Rest der Arbeit der analog verwendbare Begriff «Verknüpfung» verwendet. Im Folgenden wird nur auf für die Arbeit relevante Verknüpfungen eingegangen. Diese wurden aus GRIPS abgeleitet, wo folgende Möglichkeiten zur Verfügung stehen: FLT-UND, FLT-ODER und LOGIK=[POS/NEG]. Folgende Verknüpfungen aus der booleschen Algebra werden damit abgebildet:

- **Konjunktion (∧):** Bei der Konjunktion handelt es sich um eine *Und*-Verknüpfung. Die Konjunktion zweier Aussagen ist genau dann wahr, wenn beide Aussagen wahr sind (Tietje (2019, S. 4)). Das FLT-UND aus GRIPS entspricht der Konjunktion.
- **Disjunktion (∨):** Bei der Disjunktion handelt es sich um eine *Oder*-Verknüpfung. Die Disjunktion zweier Aussagen ist genau dann wahr, wenn mindestens eine der beiden Aussagen wahr ist (Tietje (2019, S. 5)). Das FLT-ODER aus GRIPS entspricht der Disjunktion.
- **Negation (¬):** Bei der Negation handelt es sich um eine *Nicht*-Verknüpfung. Die Negation einer Aussage ist genau dann wahr, wenn die Aussage falsch ist (Tietje (2019, S. 3)). Das LOGIK=[NEG] aus GRIPS entspricht der Negation.

Bei den Verknüpfungen gilt es die Bindungsstärke zu beachten. Die Negation hat eine grössere Bindungsstärke als die «Und»- und «Oder»-Verknüpfung. Die Konjunktion und die Disjunktion haben hingegen die gleiche Bindungsstärke. Ohne Klammern gilt also: ¬ vor ∧ und ∨. Bei aufeinanderfolgender Verwendung von verschiedenen Verknüpfungen müssen daher, je nach Ziel des booleschen Ausdrucks, die Klammern am richtigen Ort gesetzt werden. (Tietje (2019, S. 12))

2.2 Menschenzentriertes Design

Da sich diese Arbeit an menschenzentriertem Design, besser bekannt unter dem englischen Begriff User-Centered Design (UCD), orientiert, wird hier basierend auf Fachliteratur, auf das Thema eingegangen.

Viele Softwareprojekte, bis zu 50%, erreichen ihre Ziele nicht und scheitern. Oft liegt das Problem in der schlechten Kommunikation zwischen den Entwickelnden und den Endanwendenden. Das Ergebnis davon ist oft ein System, welches die Endanwendenden dazu zwingt, ihr Verhalten anzupassen, um das System nutzen zu können. UCD ist eine Gegenmassnahme, um diese Problematik zu lösen. UCD beschreibt einen Prozess, welcher die Anforderungen, Wünsche und Limitierungen seiner Endanwendenden während jeder Phase des Designprozesses berücksichtigt. Dies soll im Endeffekt verhindern, dass ein «falsches» System gebaut wird. (Shneiderman et al. (2018, S. 137))

In eine ähnliche Richtung geht auch die erste Definition von UCD, welche bereits früh durch Norman und Draper (1986, S. 61) geliefert wurde:
«But user-centered design emphasizes that the purpose of the system is to serve the user, not to use a specific technology, not to be an elegant piece of programming. The needs of the users should dominate the design of the interface, and the needs of the interface should dominate the design of the rest of the system.»

Diese eher philosophischen Umschreibungen von UCD sind gemäss Still und Crane (2017, S. 43) nicht ausreichend. Es werden zuverlässige Prinzipien benötigt, welche man wiederholt in Projekten anwenden kann. Still und Crane (2017, S. 43-61) schlagen zehn Prinzipien vor, durch welche sich UCD effektiv anwenden lässt. An dieser Stelle wird auf die vier für diese Arbeit relevantesten Prinzipien eingegangen.

- **Man soll die Nutzenden früh einbeziehen:** Durch das zeitnahe Einbeziehen der Nutzenden wird das Design bereits früh von mehreren Personen bewertet. Dadurch können potenzielle Probleme vorzeitig entdeckt und behoben werden, bevor Änderungen am Design teuer werden.
- **Man soll die Nutzenden oft einbeziehen:** In jeder Phase des Projekts sollen die Endanwendenden des Produkts einbezogen werden. Um die Nutzenden einzubeziehen, gibt es viele Möglichkeiten wie Umfragen, Fokusgruppen, Interviews, Prototyping oder Usability Testing. Es geht darum das Produkt zusammen mit den Nutzenden iterativ zu verbessern.
- **Man soll mehrere Methoden für die User Research verwenden:** Generell darf man darauf vertrauen, was man durch die Nutzenden lernt, allerdings muss man die gewonnenen Informationen immer überprüfen. Hierfür sollte man Informationen nicht nur durch eine User Research Methode generieren, sondern mindestens drei verschiedene Methoden anwenden, welche einem dabei helfen Wissen über die Nutzenden zu generieren. Das dadurch generierte Feedback kann dann fortlaufend in den Designprozess einfliessen.
- **Man soll nie aufhören Feedback einzuholen:** UCD kann als iterativer dreistufiger Prozess betrachtet werden. Dabei wird bei jeder Iteration Feedback eingeholt, um den Nutzen des Produkts zu optimieren. Dieses Feedback fliesst dann in den Designprozess ein, wo man versucht das

Produkt zu verbessern. Dabei können zu Beginn des Projekts auch simple Paper-Prototypen zum Einsatz kommen. Irgendwann gelangt man an einen Punkt, wo man das Produkt ausliefert. Hier endet der Prozess jedoch nicht. Es gilt aus der Auslieferung wiederum zu lernen, Feedback einzuholen und das Produkt anschliessend weiter zu verbessern.

Um die zehn UCD-Prinzipien zu erfüllen, schlagen Still und Crane (2017, S. 62) einen repetierbaren Prozess mit dem Akronym RABBIT vor (siehe Abbildung 2). RABBIT setzt sich aus mehreren Teilschritten zusammen, welche die erfolgreiche Durchführung eines UCD-Projekts unterstützen. Dabei enthält jeder Teilschritt verschiedene Methoden und Techniken, die einem helfen UCD zu betreiben.
Still und Crane (2017, S. 62) halten dabei fest, dass man sowohl die Reihenfolge der Schritte an die eigenen Bedürfnisse anpassen als auch Methoden und Techniken, je nach Form des Projekts, selektiv auswählen soll. Nachfolgend wird auf die für diese Arbeit zentralen Teilschritte des RABBIT Prozesses eingegangen.

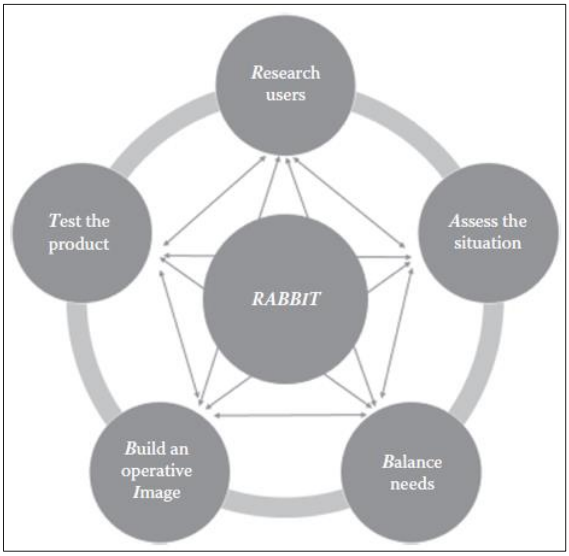


Abbildung 2: RABBIT Prozess
Quelle: Still und Crane (2017, S. 63)

- **Research users:** Es ist entscheidend die User Research Methoden zu wählen, welche am besten für das Projekt geeignet sind. Dabei gilt es mindestens drei Methoden auszuwählen, eine welche erfasst was Nutzende machen, eine die erfasst was Nutzende sagen und eine bei welcher die Designschaffenden sehen, wie die Nutzenden mit dem erstellten Design interagieren. (Still und Crane (2017, S. 68))
  Die Palette von möglichen User Research Methoden ist gross. Still und Crane (2017) beschreiben potenzielle Methoden, welche gemäss Abbildung 3 in fünf verschiedene Typen kategorisiert werden (Still und Crane (2017, S. 69-103)). Ähnliche und weitere User Research Methoden (Contextual Inquiry, Personas) werden von Garrett (2011, S. 46-51), Knight (2019, S. 84-86) und Lowdermilk (2013, S. 43, S. 77-87) vorgeschlagen.

Data analysis	Observation	Self-reporting	Designer analysis	User and use diagrams
Google analytics	Site visits	Focus groups	Task analysis	User flows
Site stats	Shadowing	Surveys	User matrix	User journey maps
Market research	Fly on the wall	Interviews	Use cases	Storyboard
User forums/ reviews	Emotion assessment	Diary studies	Affinity diagramming	Card sorting

Abbildung 3: User Research Methoden
Quelle: Still und Crane (2017, S. 69)

- **Build an operative Image:** Es ist wichtig bereits früh im Designprozess einen Prototyp zu bauen, durch welchen man von den Nutzenden Feedback einholen kann. Wie repräsentativ der Prototyp für das finale Produkt ist, sollte aufgrund der eigenen Bedürfnisse entschieden werden. (Knight (2019, S. 131), Still und Crane (2017, S. 161-162))
  Oft lohnt es sich mit simplen Skizzen oder Paper-Prototyping zu starten. Einfache Skizzen und Paper-Prototypen haben den Vorteil, dass Nutzende weniger zurückhaltend sind Änderungen vorzuschlagen, da diese Designentwürfe in einer rudimentären Form vorliegen. Generell gilt, je weniger es sich wie das Endprodukt anfühlt, desto komfortabler fühlen sich Nutzende am Designprozess mitzuwirken. Dies ist gerade zu Beginn des Projekts wichtig, wo man möglichst viel Feedback bezüglich des geplanten Produkts erhalten möchte. (Still und Crane (2017, S. 166))
  In einem nächsten Schritt können die gewonnen Erkenntnisse dann zu Wireframes weiterentwickelt werden, um den Nutzenden ein noch realistischeres Bild des Endprodukts bieten und dadurch mehr Feedback für Verbesserungen generieren zu können. Bei den Wireframes geht es zu Beginn darum das Skelett des Produkts aufzubereiten, dazu gehören das grundlegende Layout, als auch die Platzierung der wichtigsten Features. Die Wireframes können danach iterativ verfeinert und beispielsweise auch durch Interaktionsmöglichkeiten erweitert werden. Wie weit die Wireframes ausgebaut werden, muss situationsbedingt entschieden werden. (Still und Crane (2017, S. 172-174))
- **Test the product:** Das Produkt sollte fortlaufend, von den verschiedenen Prototypen bis hin zum Endprodukt, von Nutzenden getestet werden. Man spricht in diesem Zusammenhang auch von iterativem Testing, von welchem man über den ganzen Entwicklungsprozess hinweg begleitet wird. (Still und Crane (2017, S. 215))
  Um festzustellen, ob die Nutzenden Aufgaben mit den Prototypen erledigen können und um die Benutzungsfreundlichkeit des Produkts zu verifizieren, werden so genannte Usability Tests durchgeführt (Still und Crane (2017, S. 191)). Usability Testing erhält aufgrund der Bedeutung für die Arbeit an dieser Stelle ein eigenes Kapitel.

2.3 Usability Testing

Im Rahmen der Arbeit werden Usability Tests durchgeführt. Aus diesem Grund lohnt es sich ein gutes Fundament basierend auf der Fachliteratur zum Thema zu erarbeiten, um die Usability Tests anschliessend ordnungsgemäss durchführen zu können.

Sobald ein Prototyp erstellt wurde, muss dieser von realen Testpersonen getestet werden, um Feedback zu erhalten. Hier setzen Usability Tests an, welche dabei helfen die Benutzungsfreundlichkeit des Produkts zu überprüfen. (Still und Crane (2017, S. 191))

Usability Tests sollten dabei eine bestimmte Form haben. Es ist nicht effektiv, die Nutzenden einfach nach einem Feedback zum Produkt zu fragen, wie Lowdermilk (2013, S.95) anmerkt:
«Users can tell us a lot about what's working and what's not. However, the most powerful way to gain insight into what a user needs is by directly observing them. »

Bei Usability Tests geht es darum Testpersonen dabei zu beobachten, wie sie mit dem Prototyp beziehungsweise der Software interagieren, um bestimmte vordefinierte Aufgaben zu lösen. Dabei werden die Kommentare, Aktionen und Fehler, welche die Testpersonen machen, dokumentiert, um wertvolles Feedback zu gewinnen, welches dann später in die Verbesserung des Prototyps einfliessen kann. (Lowdermilk (2013, S. 95-96))

Es gibt einige wichtige Aspekte, welche es für das Usability Testing zu beachten gilt:

- Durch die Vorbereitung eines Testplans wird der Usability Test von den Testpersonen einerseits als seriös wahrgenommen und andererseits erhält dadurch jede Testperson die gleichen Aufgaben, was die Vergleichbarkeit der Resultate verbessert. (Lowdermilk (2013, S. 96))
- Testpersonen müssen darüber informiert werden, dass nicht die Testperson, sondern die Software getestet wird. Zudem gilt es zu erwähnen, dass die Resultate anonym behandelt werden. Dies soll dazu führen, dass die Testpersonen nicht nervös an die Aufgaben rangehen, weil sie Angst haben sich zu blamieren. (Lowdermilk (2013, S. 97), Nielsen (1993, S.182-183))
- Die wohl wichtigste Usability Testing Methode ist gemäss Nielsen (1993, S. 195)) «Thinking aloud». Dabei geht es darum, dass die Testpersonen während des Usability Tests fortlaufend ihre Gedanken verbal mitteilen. Da sich dies für viele Testpersonen unnatürlich anfühlt, werden sich viele dagegen sträuben. Aus diesem Grund ist es wichtig, dass man die Testpersonen fortlaufend dazu motiviert ihre Gedankengänge zu verbalisieren. Durch «Thinking aloud» sieht man nicht nur was die Testpersonen machen, sondern erfährt auch, warum sie es machen. Dadurch wird ein wesentlich hilfreicheres Feedback generiert auf dessen Basis später der Prototyp verbessert werden kann. (Lowdermilk (2013, S. 100), Nielsen (1993, S.195-196))
- Einerseits sollte der Moderator während dem Usability Test die Testperson ihren eigenen Weg finden lassen. Dies führt zu interessanteren Testresultaten und verhindert, dass die Testperson sich dumm vorkommt, wenn der Moderator die Lösung vorzeigt. Andererseits sollte der Moderator, falls die Testperson nicht mehr weiterkommt, mit Hinweisen unterstützend zur Seite stehen, so dass die Testperson den Test fortsetzen kann. (Nielsen (1993, S. 183-184))

2.4 Diskussion

In diesem Kapitel wird darauf eingegangen, wie die erarbeiteten theoretischen Grundlagen in die Arbeit einfliessen.

Boolesche Algebra
Die boolesche Algebra ist der Grundbaustein für den Aufbau von Filterausdrücken. Da das Ziel dieser Arbeit darin besteht einen Filter-Editor für komplexe Filterausdrücke zu bauen, helfen die zur booleschen Algebra erarbeiteten Grundlagen zu verstehen, wie Filterausdrücke mathematisch aufgebaut sind. So lässt sich aus den theoretischen Grundlagen ableiten, welche relevanten Verknüpfungen im Filter-Editor angeboten werden können (AND, OR, NOT). Ebenso von Bedeutung sind die verschiedenen Bindungsstärken der einzelnen Verknüpfungen, welche es in der Umsetzung zu berücksichtigen gilt.

Menschenzentriertes Design
Die zehn UCD-Prinzipien von Still und Crane (2017) werden in der Arbeit so weit als möglich berücksichtigt. Dabei wird der Fokus auf die, aus Sicht des Projektteams, wichtigsten Prinzipien gelegt, welche in Kapitel 2.2 beschrieben sind.
Ein Hauptaugenmerk wird während der Arbeit darauf gelegt, dass Nutzende früh und oft miteinbezogen werden und fortlaufend Feedback eingeholt wird. Dabei werden im Projektverlauf Feedbacks von verschiedenen Personengruppen eingeholt. In einer frühen Phase des Projekts wird man sich mit Fachleuten der STAR AG austauschen, um die ersten Konzepte zu evaluieren. Im weiteren Verlauf des Projekts, sobald ein Prototyp steht, wird dann zusätzlich Feedback von Mitgliedern der technischen Redaktion der STAR AG als auch von weiteren Testpersonen eingeholt. Das dadurch fortlaufend generierte Feedback soll dabei helfen, am Ende einen möglichst benutzungsfreundlichen Filter-Editor zu bauen.
Zum Einholen von Feedback wird auf zwei User Research Methoden gesetzt. Dies entgegen der Empfehlung von Still und Crane (2017) mindestens drei User Research Methoden zu verwenden. Der Grund dafür ist das limitierte Zeitbudget der Arbeit. Zum Einsatz kommen im Rahmen dieser Arbeit Usertests und Usability Testing. Einerseits wird in der Konzeptionsphase ein Prototyp mit Figma erstellt, welcher dann durch Usertests mit den oben beschriebenen Zielgruppen evaluiert und iterativ verbessert wird. Andererseits werden gegen Ende der Implementierungsphase Usability Tests durchgeführt, welche helfen sollen, die Benutzungsfreundlichkeit des entwickelten Filter-Editors zu überprüfen.

Der von Still und Crane (2017) vorgeschlagene RABBIT Prozess wird in der Arbeit ebenfalls berücksichtigt. Das Hauptaugenmerk wird auf die Teilschritte des RABBIT Prozesses gelegt, welche aus Sicht des Projektteams, am wichtigsten für die Arbeit sind. An dieser Stelle wird darauf eingegangen, wie diese Teilschritte in der Arbeit umgesetzt werden.

- **Research users:** Für die User Research wird, wie bereits vorangehend erwähnt, auf Usertests und Usability Testing gesetzt.
- **Build an operative Image:** Es wird mit dem Entwurf von Konzepten auf Papier gestartet um schnell erstes Feedback einholen zu können. Anschliessend wird aus den entwickelten Konzepten ein Prototyp in Figma gebaut, welcher iterativ verbessert und schlussendlich implementiert wird.
- **Test the product:** Es wird fortlaufend über das ganze Projekt hinweg von mehreren Personengruppen immer wieder Feedback eingeholt. Dies geschieht in Form von Usertests und später durch Usability Testing.

Usability Testing

Die zu den Usability Testing gesammelten theoretischen Grundlagen sollen vor allem dabei helfen die Usability Tests im Rahmen der Arbeit sauber durchzuführen. Dabei sind alle Aspekte auch für die Usertests, welche mit dem Figma Prototyp durchgeführt werden, relevant, wodurch die theoretischen Grundlagen zum Thema einen doppelten Nutzen haben. In Bezug auf die Durchführung der Usertests als auch der Usability Tests wird deshalb folgendes gemacht:

- Es wird für beide Testarten jeweils ein Testprotokoll vorbereitet.
- Die Testpersonen werden vor der Testdurchführung immer darüber informiert, dass nicht die Testperson, sondern die Software getestet wird.
- Bei allen Testdurchführungen ist immer ein Moderator dabei, welcher das Feedback der Testperson aufnimmt und bei Problemen mit Hinweisen unterstützend zur Seite steht.
- Der Moderator ermuntert die Testperson wiederholt während der Testdurchführung ihre Gedanken verbal mitzuteilen.

3 Konzepte

In diesem Kapitel wird auf die einzelnen Konzepte eingegangen, welche zum Start der Konzeptphase entwickelt wurden. Alle Konzepte wurden dabei basierend auf bestehenden Erfahrungen mit existierenden Tools und den dadurch entstandenen Ideen entworfen. Aus diesem Grund startet jedes Konzept mit einer Skizze und einer Beschreibung dazu. Danach folgt jeweils eine Kategorisierung, welche die Aufgabe des Konzepts innerhalb des Filter-Editors verdeutlicht. Was die einzelnen Kategorien genau bedeuten, wird in einem eigenen Unterkapitel kurz erläutert. Im Anschluss folgen, falls vorhanden, Praxisbeispiele, welche das Konzept stützen. Zum Abschluss wird auf Hinweise zur Umsetzung eingegangen, welche einerseits auf eigenen Ideen und andererseits auf in der Fachliteratur gefundenen Hinweisen basieren.

Definition Konzept: Ein Konzept beschreibt im Rahmen dieser Arbeit eine Teilfunktion des zu entwerfenden Filter-Editors.

3.1 Kategorien

Um den Konzepten eine Zugehörigkeit zuzuweisen, werden Kategorien verwendet. Die Kategorien können als Leitwörter betrachtet werden, welche jeweils die Lösung zu einem allgemeinen Problem beschreiben. Da einige der Konzepte mehrere Probleme gleichzeitig lösen, ist es möglich, dass einem Konzept mehrere Kategorien zugewiesen sind. Die nachfolgende Tabelle erläutert die verwendeten Kategorien. In der zweiten Spalte wird jeweils beschrieben, welches Problem, durch Konzepte dieser Kategorie, gelöst wird.

Kategorie	Beschreibung
Visualisierung	Für die Nutzenden wird die Komplexität verringert, indem abstrakte Vorgänge und Zusammenhänge grafisch dargestellt werden.
Bedienung	Erlaubt es den Nutzenden mit dem Tool zu interagieren und dieses zu bedienen.
Validierung	Kontrolliert die Ausdrücke für die Nutzenden auf Validität.
Vereinfachung	Für die Nutzenden wird die Komplexität verringert, indem Nutzende bei der Erstellung der Filterausdrücke unterstützt werden.
Verständnis	Erleichtert den Nutzenden das Verständnis der Filterausdrücke.

Tabelle 1: Kategorien für die Konzepte

3.2 Darstellung als Baum oder Zeile

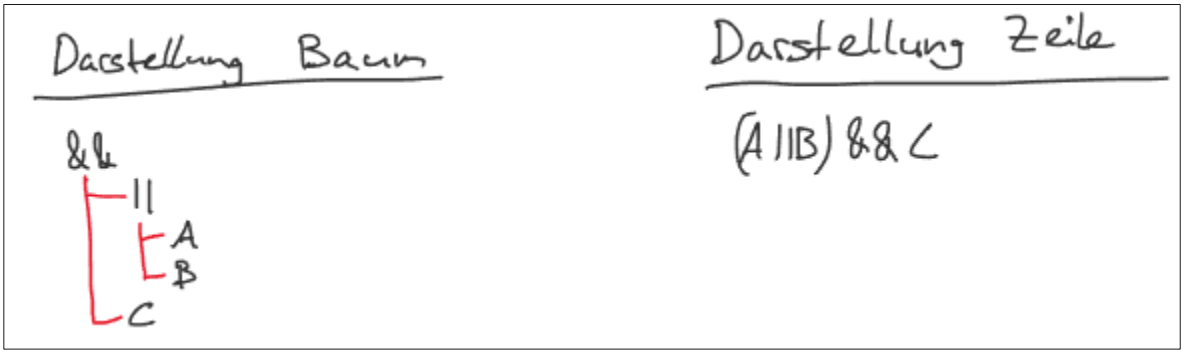


Abbildung 4: Darstellung als Baum oder Zeile Konzept
Quelle: Eigene Darstellung

Beschreibung:
Ein Filterausdruck lässt sich in zwei Arten darstellen. Einerseits gibt es die Baumdarstellung und andererseits die Zeilendarstellung (siehe Abbildung 4). Bei der Baumdarstellung befinden sich die Verknüpfungen (&& - und, || - oder) in den inneren Knoten und die Bedingungen (A, B, C) in den Blattknoten. Bei der Zeilendarstellung hingegen wird mit Klammern gearbeitet. Der Filterausdruck in Abbildung 4 ist der Gleiche, nur die Darstellung ist unterschiedlich.

Kategorie:
Visualisierung, Verständnis

Praxisbeispiel:
In der Praxis können beide Darstellungsarten vorgefunden werden. Die Baumdarstellung findet sich zum Beispiel im Filter-Editor von DevExpress (siehe Abbildung 5). Die Zeilendarstellung hingegen wird unter anderem von YouTrack eingesetzt, um eine Filterung des Boards zu ermöglichen (siehe Abbildung 6).

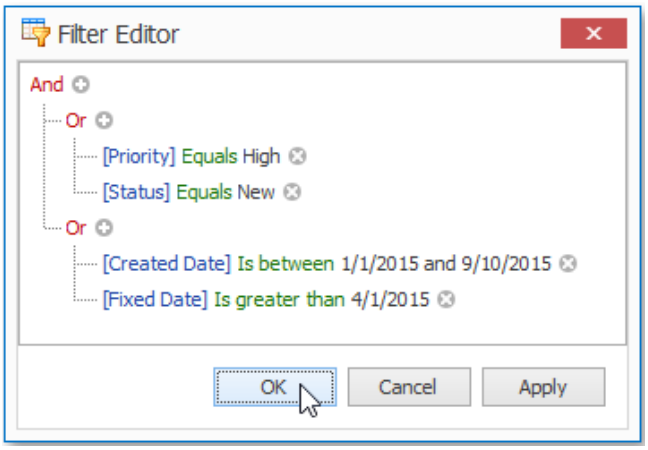


Abbildung 5: Filter-Editor von DevExpress
Quelle: https://docs.devexpress.com/WindowsForms/114645/controls-and-libraries/data-grid/getting-started/walkthroughs/filter-and-search/tutorial-filter-editor

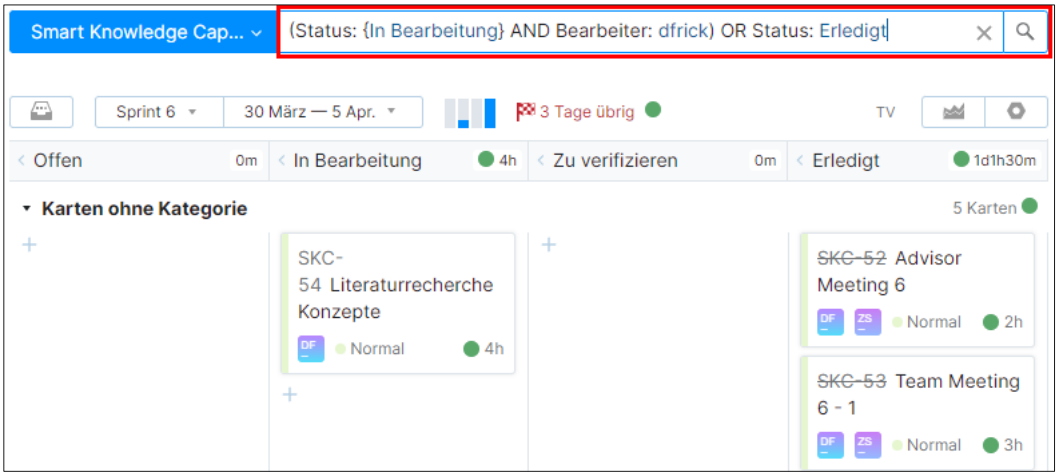


Abbildung 6: Filter-Zeile in YouTrack
Quelle: Eigene Darstellung

Anmerkungen zur Umsetzung:

- Das «Law of Familiarity» besagt, dass man das UI möglichst mit Komponenten aufbauen sollte, mit welchen die Nutzenden bereits vertraut sind (Knight (2019, S. 108)). Die Darstellung eines Filterausdrucks in Form eines Baums oder einer Zeile macht daher Sinn, da die Wahrscheinlichkeit relativ gross ist, dass Nutzende bereits einmal mit einer ähnlichen Darstellung in Berührung gekommen sind. Je vertrauter eine Komponente auf einen Nutzenden wirkt, desto komfortabler fühlt sich die Bedienung an (Knight (2019, S. 108)).
- Die Entscheidung für eine der beiden Darstellungen ist schwierig. Die präferierte Variante hängt vermutlich von der Präferenz des Nutzenden ab. Dies da jeder Nutzende sein eigenes mentales Model davon hat, wie ein Produkt, in diesem Fall der Filter-Editor, idealerweise aussehen sollte (Weinschenk (2020, S. 80)). So wäre es denkbar den Nutzenden beide Darstellungsmöglichkeiten zur Verfügung zu stellen. Usertests werden dann zeigen, welche Darstellung von den Nutzenden besser angenommen wird.

3.3 Trennung von Bedingungen und Verknüpfungen

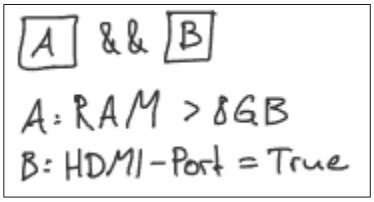


Abbildung 7: Trennung von Bedingungen und Verknüpfungen Konzept
Quelle: Eigene Darstellung

Beschreibung:
Die Idee dieses Konzepts ist es die Bedingungen separat von den Verknüpfungen zu verwalten (siehe Abbildung 7). Dies hat einerseits den Vorteil, dass Bedingungen, die mehrmals in einem Filterausdruck vorkommen, wiederverwendet werden können. Andererseits ermöglicht es auch zuerst alle Bedingungen zu erfassen, bevor man sich mit den Verknüpfungen auseinandersetzen muss.

Kategorie:
Bedienung, Vereinfachung, Verständnis

Praxisbeispiel:
Die Trennung der Bedingungen von den Verknüpfungen findet man auch in der Praxis vor. Ein Beispiel dafür ist in der Report Filterung von Salesforce zu finden (siehe Abbildung 8). Wie man sieht, werden bei Salesforce die Bedingungen im roten Bereich separat erfasst und dann anhand der Bedingungsnummern im grünen Bereich verknüpft.

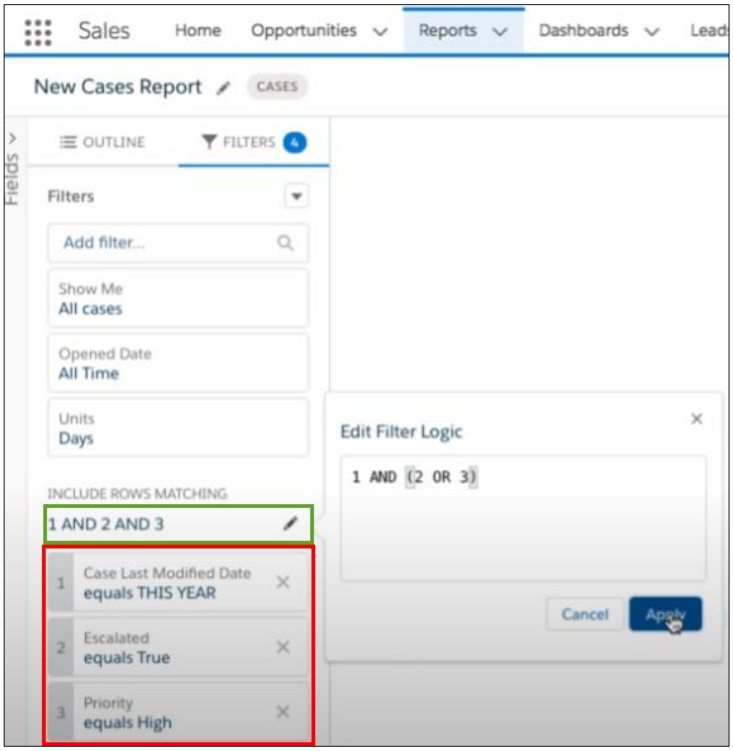


Abbildung 8: Report Filterung von Salesforce
Quelle: https://www.youtube.com/watch?v=n6ht2schjJ4&t=154s

Anmerkungen zur Umsetzung:
- Die Trennung der Bedingungen und Verknüpfungen lässt sich sowohl mit der Darstellung als Baum als auch der Darstellung als Zeile kombinieren.
- Es wäre denkbar eine Importfunktion für Bedingungen in den Filter-Editor anzubieten, wodurch diverse Bedingungen im Voraus zur Verfügung gestellt werden können, so dass diese nicht jedes Mal neu erfasst werden müssen.

3.4 Tokenisierung

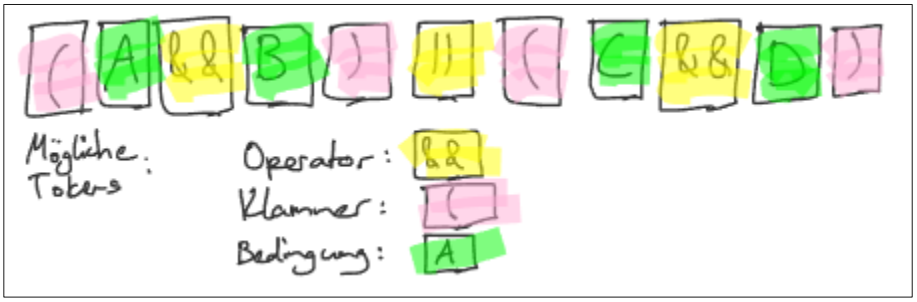


Abbildung 9: Tokenisierung Konzept
Quelle: Eigene Darstellung

Beschreibung:
Ein Filterausdruck setzt sich aus mehreren Bestandteilen zusammen. Die einzelnen Bestandteile sind die Bedingungen, die Verknüpfungen (Operatoren) und die Klammern zur Strukturierung des Ausdrucks. Die Idee ist es diese verschiedenen Bestandteile des Ausdrucks als einzelne Tokens zu behandeln. Abbildung 9 zeigt, wie dies anhand der Darstellung als Zeile aussehen könnte. Bei der Baumdarstellung würde ebenfalls mit Tokens gearbeitet werden, die Klammern wären aber implizit durch die Baumstruktur gegeben und nicht direkt sichtbar.

Kategorie:
Visualisierung

Anmerkungen zur Umsetzung:
- Es sollte mit Constraints gearbeitet werden, welche vom Typ des Tokens abhängig sind. Solche Constraints helfen dem Nutzenden dabei weniger Fehler zu machen (Knight (2019, S. 117)). Über ein Kontextmenü könnte beispielsweise auf den Verknüpfung-Tokens der Austausch von AND durch OR oder umgekehrt ermöglicht werden. Auf Klammer-Tokens wäre denkbar eine Löschfunktion anzubieten, welche den Teilausdruck in der Klammer aus dem Filterausdruck entfernt.
- Die Tokenisierung ermöglicht die Nutzung von Drag & Drop. Dabei handelt es sich um ein separates Konzept, welches im nächsten Kapitel behandelt wird.

3.5 Drag & Drop

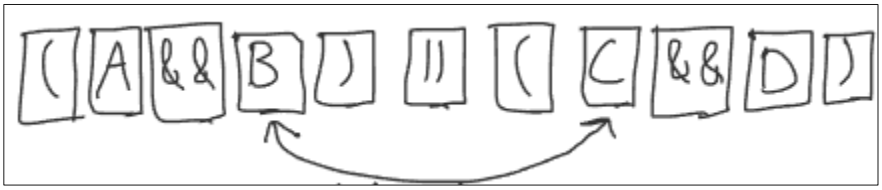


Abbildung 10: Drag & Drop Konzept
Quelle: Eigene Darstellung

Beschreibung:
Drag & Drop ermöglicht es den Nutzenden einen Filterausdruck einfach zusammenzustellen und zu verändern. Abbildung 10 zeigt, wie dies anhand der Darstellung als Zeile aussehen könnte. So wäre es möglich zwei Bedingungen per Drag & Drop auszutauschen. Auch in der Baumdarstellung wären Drag & Drop Operationen möglich, so könnten dort sogar ganze Teilbäume eines Filterausdrucks umgehängt werden.

Kategorie:
Bedienung

Anmerkungen zur Umsetzung:
- Im Falle der Realisierung des Konzepts «Trennung der Bedingungen von den Verknüpfungen», wäre ein Drag & Drop aus der Liste der Bedingungen heraus denkbar.
- Es wäre eine Idee, dass Verknüpfungen (AND und OR) direkt aus einer Toolbar per Drag & Drop in den Filterausdruck eingefügt werden können.
- Es ist wichtig, dass die Affordance gewährleistet wird. Damit ist gemeint, dass visuelle Elemente wie die Tokens den Nutzenden Hinweise liefern, wie sie mit dem Element interagieren können (Knight (2019, S. 116), Weinschenk (2020, S. 18)). Es sollte für die Nutzenden verständlich sein, dass man die Tokens per Drag und Drop verschieben kann.
- Es gilt Constraints einzusetzen, um dem Nutzenden zu helfen Fehler zu vermeiden (Knight (2019, S. 117)). So können einerseits nicht alle Tokens per Drag und Drop verschoben werden und andererseits gilt es auch einzuschränken, welche Tokens wohin gedroppt werden können.

3.6 Testing und Highlighting

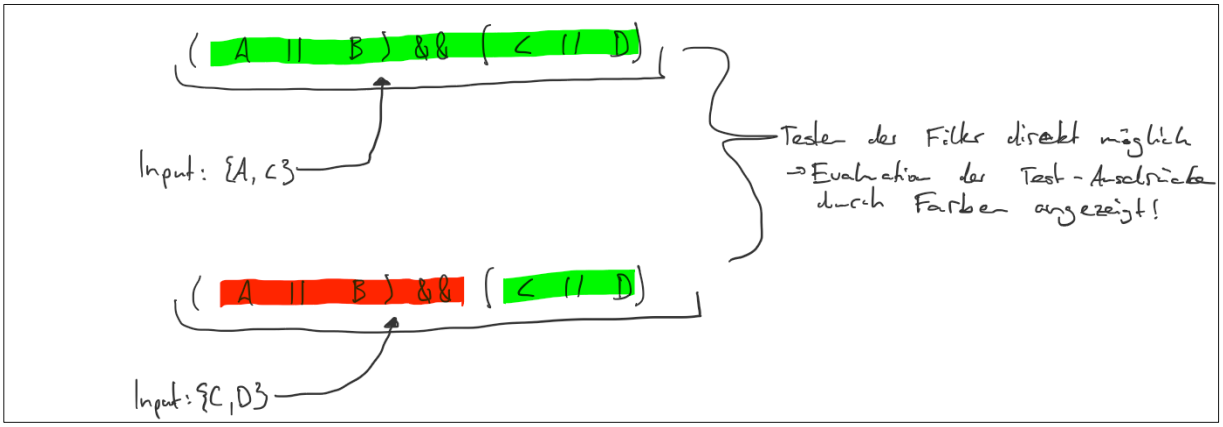


Abbildung 11: Testing und Highlighting Konzept
Quelle: Eigene Darstellung

Beschreibung:
Bei diesem Konzept geht es darum, einen erstellten Filterausdruck mit einem Test-Input testen zu können. Als Resultat des Testings werden dabei Teile des Ausdrucks farblich hinterlegt, so dass ersichtlich ist, welche Teilbereiche des Filterausdrucks der Testinput erfüllt und welche nicht (siehe Abbildung 11). Durch dieses visuelle Feedback wird der Nutzende optimal dabei unterstützt, um Nachbesserungen am Filterausdruck vorzunehmen.

Kategorie:
Validierung, Vereinfachung, Verständnis, Visualisierung

Anmerkungen zur Umsetzung:
- Um visuell die Aufmerksamkeit zu erlangen, ist es von Vorteil ein Element anders aussehen zu lassen als die anderen. Dies kann beispielsweise durch eine andere Farbe oder Form des Elements erreicht werden. (Weinschenk (2020, S. 13))
- Damit farbenblinde Personen nicht benachteiligt werden, ist es wichtig Farben zu verwenden, welche für alle Personen, unabhängig von einer Sehschwäche, gleich aussehen. Eine weitere Möglichkeit besteht darin ein Element nebst der Farbe auf eine weitere Art hervorzuheben. Beispielsweise kann ein Element zusätzlich durch eine andere Liniendicke hervorgehoben werden. (Weinschenk (2020, S. 25-26))
- Bei der Farbwahl gilt es einerseits zu beachten, dass bestimmte Farben in verschiedenen Kulturen verschiedene Bedeutungen haben. Andererseits muss bei der Farbwahl auch immer der Kontext berücksichtigt werden. So wird beispielsweise die Farbe Rot von für die Buchhaltung zuständigen Personen in einem amerikanischen Unternehmen als negativ wahrgenommen, da dies in ihrem Berufsumfeld mit einem Verlust in Verbindung gebracht wird. (Weinschenk (2020, S. 29))

3.7 Vereinfachung von Ausdrücken

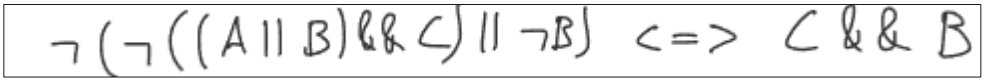


Abbildung 12: Vereinfachung von Ausdrücken Konzept
Quelle: Eigene Darstellung

Beschreibung:
Wenn in einem Filterausdruck viele Bedingungen in verschiedenen Kombinationen berücksichtigt werden müssen, kann es dazu kommen, dass ein sehr umfangreicher Ausdruck entsteht, welcher nur schwer nachvollziehbar ist. Um diesem Problem entgegenzuwirken, wäre eine Funktion wünschenswert, welche den Ausdruck nach Möglichkeit vereinfacht. Abbildung 12 bildet eine solche Vereinfachung eines Ausdruckes ab.

Kategorie:
Vereinfachung

Anmerkungen zur Umsetzung:
- Eine solche Funktion zur Vereinfachung eines Ausdrucks entspricht dem Prinzip, dass man möglichst das System die Arbeit machen lassen soll. Dadurch kann dem Nutzenden ein Teil der Komplexität abgenommen werden. (Knight (2019, S. 75))
- Da sich ein Ausdruck nur in bestimmten Fällen vereinfachen lässt, ist es wichtig, dass dem Nutzenden ein Feedback gegeben wird, ob eine Vereinfachung durchgeführt werden konnte oder nicht. (Knight (2019, S. 116))
- Es muss bei der Umsetzung geschaut werden, was für Möglichkeiten es gibt einen Filterausdruck, was nichts anderes als ein boolescher Ausdruck ist, zu vereinfachen. Sollte ein möglichst kurzer Ausdruck oder eine boolesche Normalform angestrebt werden? Können dem Nutzenden eventuell verschiedene Optionen angeboten werden?

3.8 Diskussion

Es wurde entschieden alle erarbeiteten Konzepte in einem ersten Figma Prototyp zu verbauen (siehe Kapitel 4).

Den Kern des Filter-Editors sollen dabei folgende Konzepte bilden:
- Darstellung als Baum oder Zeile
- Trennung von Bedingungen und Verknüpfungen
- Tokenisierung
- Drag & Drop

Diese vier Konzepte bilden die Basis für den Filter-Editor, da durch ihre Realisierung ein nutzbarer MVP gebaut werden kann. So erlauben diese Konzepte in der Kombination sowohl die Visualisierung als auch die komfortable Bearbeitung eines Filterausdrucks. Der Fokus bei der Erstellung des Figma Prototyps wird deshalb auf diese vier Konzepte gelegt.

Zusätzlich wurde entschieden auch die restlichen Konzepte im Prototyp zu verbauen:
- Testing und Highlighting
- Vereinfachung von Ausdrücken

Diese beiden Konzepte können als Features betrachtet werden, welche die Nutzenden des Filter-Editors bei der Erstellung eines Filterausdrucks zusätzlich unterstützen. Beide Konzepte bieten aus Sicht des Projektteams einen Mehrwert. Die beiden Konzepte sollen deshalb ebenfalls im Figma Prototyp verbaut werden, der dafür betriebene Aufwand soll sich jedoch in Grenzen halten. Das Ziel ist es, die beiden Konzepte so weit im Figma Prototyp zu integrieren, dass die Konzepte durch Mitglieder der technischen Redaktion der STAR AG als auch weitere Testpersonen in den Usertests bewertet werden können.

4 Prototyping

In diesem Kapitel wird der Weg von den Konzepten hin zum finalen Prototyp beschrieben. Als Erstes wird das verwendete Prototyping-Tool vorgestellt. Danach wird das Vorgehen und die darin enthaltenen Schritte beschrieben, bevor auf die einzelnen Versionen des Prototyps und das jeweils dazugehörige Feedback eingegangen wird.

4.1 Figma

Um den Prototyp zu bauen, wurde das Design-Tool Figma gewählt. Figma erlaubt es interaktive Prototypen zu designen und testen. Im Tool können mit einfachen Grundbausteinen, wie beispielsweise Pfeilen, Strichen, Rechtecken, Kreisen und Dreiecken komplexe Komponenten gestaltet werden. Die Bausteine können dabei per Drag & Drop auf die existierende Arbeitsfläche gezogen werden. Einzelne Bausteine können anschliessend in der Arbeitsfläche zu einer einzelnen Komponente gruppiert und als solche verwendet werden. Beispielsweise kann eine Button-Komponente erstellt werden, indem ein Rechteck mit einem Text befüllt und anschliessend zu einer Komponente gruppiert wird. Dieser Button kann darauf beliebig oft wiederverwendet werden.
Da viele einfache Komponenten, wie beispielsweise Buttons oder Dropdowns, im Wesentlichen gleich aussehen und gleich funktionieren, möchte man diese nicht nachbauen. Hier kommt die Community von Figma zur Hilfe, welche bereits viele verschiedene Komponenten designt hat und diese anderen Nutzenden kostenlos zur Verfügung stellt.
Neben den Grundbausteinen und Komponenten von der Community braucht es weitere Bausteine, um einen interaktiven Prototyp gestalten zu können. Zum einen sind dies Frames und zum anderen Interactions. Ein Frame ist eine Oberfläche, welche beispielsweise einen Smartphone-Bildschirm oder auch einen Desktop darstellt. Solche Frames können ebenfalls als Bausteine, welche in der Arbeitsfläche verwendbar sind, betrachtet werden. Mehrere Frames können dabei durch Interactions miteinander verknüpft werden. Eine Interaction kann für jeden Baustein definiert und mit verschiedenen Attributen versehen werden. Attribute sind beispielsweise «Wann wird die Interaction ausgelöst?», «Was macht die Interaction?» und «Was ist das Ziel der Interaction?».
Das folgende Beispiel demonstriert, wie die Bausteine kombiniert genutzt werden können, um einen interaktiven Prototyp zu bauen: In einem Frame wird ein Button platziert. Für den Button wird eine Interaction definiert, welche so konfiguriert ist, dass wenn der Button angeklickt wird, das aktuelle Frame ausgeblendet und ein zweites Frame eingeblendet wird.

Die in diesem Kapitel vorgestellten Bausteine von Figma werden vom Projektteam genutzt, um einen interaktiven Prototyp für den Filter-Editor zu bauen.

4.2 Vorgehen

Da für das Vorgehen bei der Entwicklung des Prototyps das menschenzentrierte Design gewählt wurde, ist der Prototyp in mehreren Iterationen gebaut und verbessert worden. In jeder Iteration entstand dabei ein Vorschlag in Form eines Mockups oder Prototyps. Zu jedem Vorschlag wurde darauf jeweils Feedback eingeholt, welches in die nächste Iteration bzw. die Entwicklung des nächsten Vorschlags eingeflossen ist. Dies wurde so lange wiederholt, bis ein zufriedenstellendes Resultat erreicht werden konnte.

4.2.1 Aufbau der Feedbackauswertungen

Alle Feedbacks werden jeweils in einer Tabelle mit drei Spalten aufgeführt. Die erste Spalte enthält die Bewertung und dient dazu, die einzelnen Feedbacks einzuordnen. Positive Kommentare sind mit einem (+) markiert. Verbesserungsvorschläge sind mit einem (=) markiert. Negative Kommentare sind mit einem (–) markiert.
Die zweite Spalte beinhaltet den Kommentar der Testperson, wobei Kommentare, welche mehrfach in ähnlicher Form vorkamen, zusammengefasst wurden. Die dritte Spalte beinhaltet die vom Projektteam vorgenommenen Anpassungen für die nächste Iteration.

4.3 1. Iteration: Mockup

In der ersten Iteration wurde mit dem Ziel möglichst rasch Themen für eine erste Diskussion zu haben, ein erster Vorschlag erstellt (siehe Abbildung 13). Dieser erste Vorschlag war ein in Figma erstelltes Mockup. Das Mockup diente dabei dazu, die Vorstellungen für den Umfang des Filter-Editors zu diskutieren.

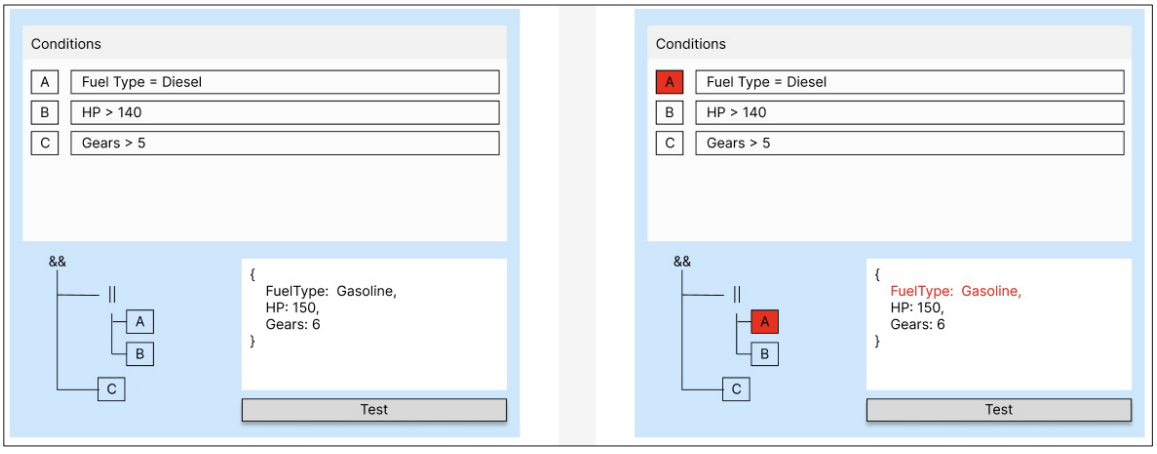


Abbildung 13: Erstes Mockup
Quelle: Eigene Darstellung

4.3.1 Feedback

Die folgenden Aspekte wurden besprochen:

Bewertung	Kommentar	Anpassung
+	Konzepte «Trennung von Bedingungen und Verknüpfungen» und «Testing und Highlighting» auf den ersten Blick erkennbar.	-
-	Darstellung ist zu kompakt. Das gewählte Frame ist zu klein, da Desktops in der Regel grösser sind.	Vergrösserung des Frames.
-	Der Entwurf lässt zu viele Fragen unbeantwortet.	-
-	Wie können Nutzende neue Ausdrücke hinzufügen?	«Add»-Button eingefügt.
-	Wie soll der Baum wachsen? Der Platz ist zu knapp.	Scrollbar für Arbeitsfläche eingefügt (für v1).
-	Was bedeutet «&&» und «||»?	AND anstelle von && und OR anstelle von || verwendet (für v1).
-	Testfeature erfordert von Nutzenden JSON-Kenntnisse.	Testmodal umgebaut (für v1).

Tabelle 2: Feedback zum Mockup

Das Mockup wurde im Anschluss wie folgt überarbeitet (siehe Abbildung 14).

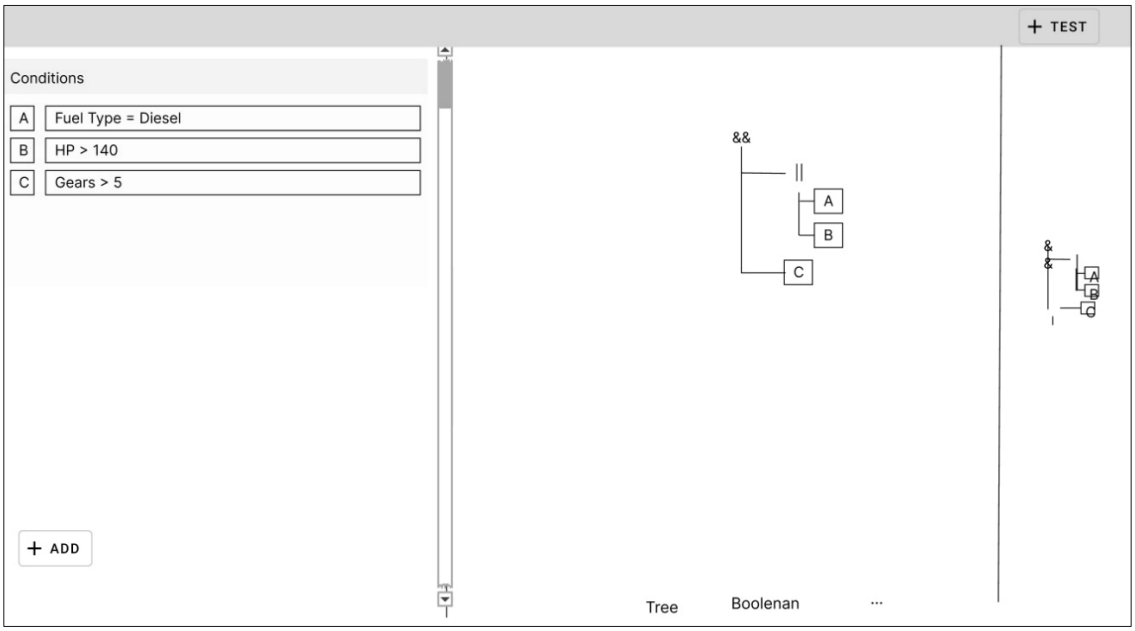


Abbildung 14: Verbessertes Mockup
Quelle: Eigene Darstellung

4.4 2. Iteration: Prototyp Version 1

Auf Basis des verbesserten Mockups wurde ein erster interaktiver Prototyp erstellt. Dieser Prototyp gilt als erster konkreter Lösungsvorschlag. Viele der erarbeiteten Konzepte sind vollständig in diesen ersten Prototyp integriert und in diesem interaktiv testbar. Nachfolgend kommt eine grobe Beschreibung vom Prototyp, bevor im Anschluss auf das Feedback dazu eingegangen wird.

Der Prototyp besteht zu Beginn aus einer leeren Arbeitsfläche (siehe Abbildung 15), die startklar für die Erstellung eines Filterausdrucks ist. Auf der linken Seite befindet sich dabei der Bereich für die Bedingungen und auf der rechten Seite der Platz für die Darstellung des Filterausdrucks. Dazu gibt es eine Menüleiste, welche mehrere Funktionalitäten bietet. So gibt es eine Toolbar, welche die einzelnen Bausteine (AND, OR, CONDITION) enthält, welche in den Filterausdruck eingefügt werden können. Auf der rechten Seite der Toolbar findet man des Weiteren einen Switch, welcher es ermöglicht zwischen Baum- und Zeilenansicht zu wechseln. Gleich rechts daneben gibt es einen Button für weitere Funktionen und nochmals rechts daneben einen Button für das Öffnen des Test-Modals zum Testen des Filterausdrucks. Dazu gibt es noch einen speziellen Bereich ganz rechts, welcher den Filterausdruck in einer Miniaturansicht darstellt. Die Idee der Miniaturansicht ist es, dass sie den Nutzenden bei grösseren Ausdrücken die Navigation erleichtert.
Durch das Klicken auf das Plus-Icon oder das Anklicken eines Bausteins (AND, OR, CONDITION) in der Toolbar kann ein erster Datenblock in die Arbeitsfläche eingefügt werden. Mit den gegebenen Funktionen ist es dem Nutzenden möglich beliebig komplizierte Ausdrücke zusammenzubauen. Abbildung 16 zeigt, wie der Prototyp mit einem erstellten Filterausdruck aussieht.

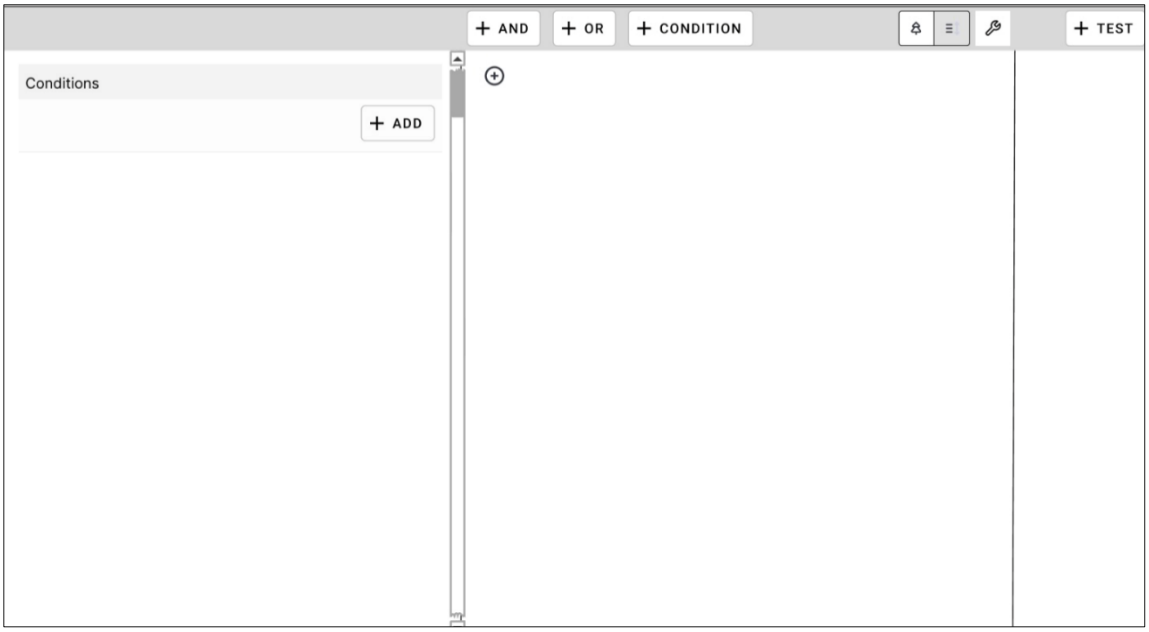


Abbildung 15: Leerer Prototyp (v1)
Quelle: Eigene Darstellung

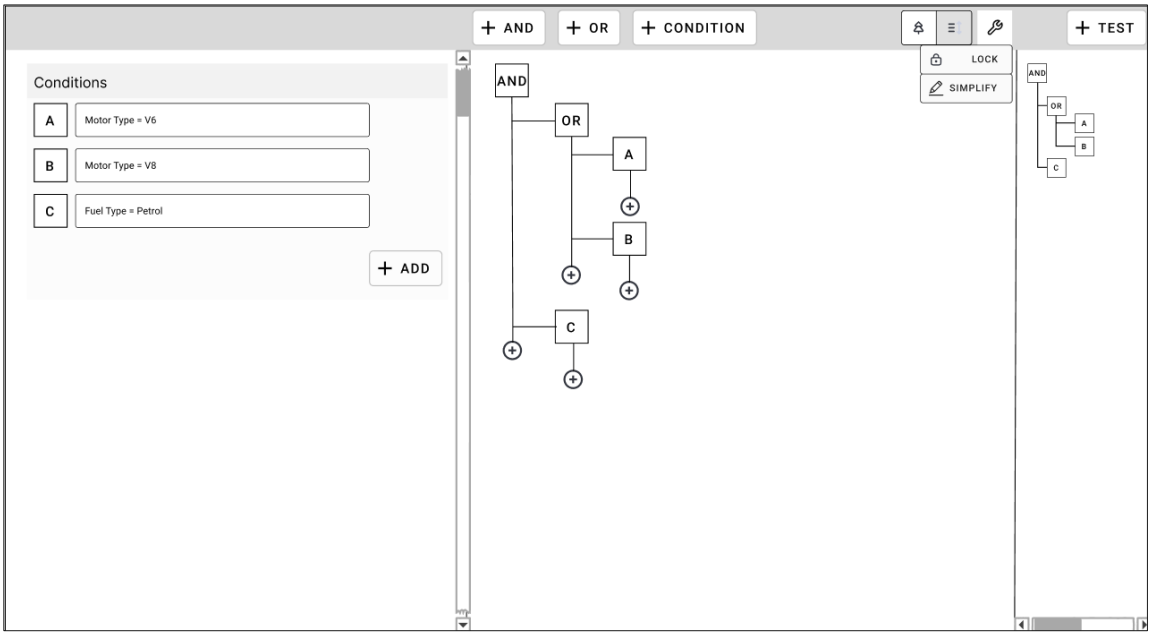


Abbildung 16: Prototyp (v1)
Quelle: Eigene Darstellung

4.4.1 Feedback

Der Prototyp stellte sich als guter erster Lösungsansatz heraus, trotzdem gab es noch viel Verbesserungspotenzial. Nach einer gelungenen Demonstration für die Fachleute der STAR AG kam es zu einer äusserst produktiven Diskussionsrunde. Aus der Diskussion konnten folgende Schlussfolgerungen gezogen werden:

Bewertung	Kommentar	Anpassung
+	Das Test-Feature ist eine sehr gute Idee.	-
+	Der Prototyp ist grösstenteils übersichtlich.	-
+	Vereinfachung der Ausdrücke ist bei der STAR AG ein Bedürfnis. Gutes Feature.	-
=	Umschalten zwischen Ansichten muss von Endnutzenden bewertet werden.	-
=	Zuerst sollen Bedingungen erstellt werden, dann der Baum (Figma Einschränkung).	Umbau der Interaktionen, so dass zuerst Bedingungen erstellt werden.
=	Bedingungen in drei Felder aufteilen wäre besser, um den Nutzenden die Eingabe zu erleichtern.	Bedingungen in drei Felder aufgeteilt.

=	Teilausdrücke sollten kopierbar sein, da sich viele dieser Teilausdrücke oft wiederholen.	Dokumentiert als Weiterentwicklungsmöglichkeit.
-	Baumansicht ist oft schwierig nachzuvollziehen bei vielen Verknüpfungen.	Keine Anpassung vorgenommen. Weitere Usertests müssen durchgeführt werden.
-	Die «Plus» sind störend und machen den sonst übersichtlichen Prototyp etwas unübersichtlicher.	«Plus» zum Hinzufügen von neuen Knoten entfernt.
-	Aliase sind bei vielen Bedingungen schwierig nachzuvollziehen.	Aliase entfernt. Bedingungen im Baum ausgeschrieben.
-	Testausdruck sollte beim Testen eingeblendet werden, um nachvollziehen zu können, was der eingegebene Testausdruck war.	Ansicht mit eingegebenem Testausdruck ergänzt.

Tabelle 3: Feedback zum Prototyp (v1)

4.5 3. Iteration: Prototyp Version 2

In einer nächsten Iteration wurde die erste Version des Prototyps, unter Berücksichtigung des Feedbacks, noch einmal überarbeitet (siehe Abbildung 17).

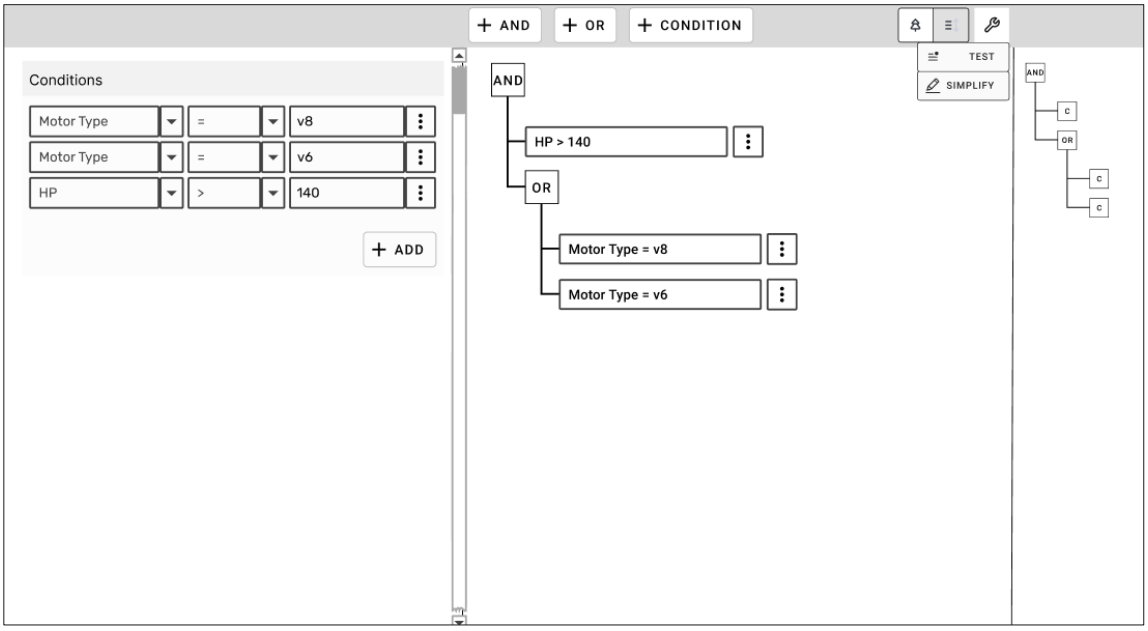


Abbildung 17: Prototyp (v2)
Quelle: Eigene Darstellung

Die auf den ersten Blick erkennbaren Unterschiede sind die Aufteilung der Bedingungen in drei Blöcke. Der erste Block ist eine vorgegebene Eigenschaft eines Fahrzeugs. Der zweite Block ist der angewandte Operator, welcher ebenfalls vorgegeben und wählbar ist. Im dritten Block ist es Nutzenden erlaubt frei einen Text einzugeben. Die zweite auffällige Änderung ist der Verzicht auf Aliase und die Ausschreibung der Bedingungen innerhalb des Baums. Des Weiteren sind auch die störenden «Plus» weg.

4.5.1 Feedback

Für die zweite Version des Prototyps wurden erste Usertests (vgl. Kapitel 7.8.4.2) mit Testpersonen durchgeführt. Zum anderen wurde der Prototyp an die Fachleute der STAR AG gesendet, um ein weiteres Feedback zu erhalten.
Nachfolgend die wichtigsten Punkte aus den verschiedenen Feedbacks zusammengefasst:

Bewertung	Kommentar	Anpassung
+	Gute Bedienbarkeit.	-
+	Trennung von Bedingungen und Baum ist gut.	-
+	Drag & Drop innerhalb des Baums ist gut.	-
+	Testfeature erneut gelobt.	-
+	Zeigt fehlerhaften Pfad an und nicht nur den Ausdruck, der fehlschlägt.	-
+	Baumansicht ist gut und von anderen Orten her vertraut.	-
+	Zeilenansicht ist gut.	-
=	Anleitung für das Testfeature nötig.	-
=	Suchfunktion für Bedingungen in der Liste wäre wünschenswert.	Dokumentiert als Weiterentwicklungsmöglichkeit.
=	Konfiguration für das Testen der Ausdrücke exportierbar / importierbar machen.	Import/Export-Buttons im Testmodal hinzugefügt.
=	Etwas unklar, weshalb man erst Bedingungen erfassen muss (Einschränkung Figma).	In vorheriger Version war es umgekehrt und wurde als negativ empfunden, deshalb wurde keine Anpassung gemacht.

=	Erste Bedingung in der Liste links soll von Anfang an vorhanden sein, erspart einen Klick auf «Add»- Button.	Leere Bedingung im leeren Prototyp vorgegeben.
=	Erste Verknüpfung im Baum bereits zur Verfügung stellen, da so oder so immer eine Verknüpfung die Wurzel des Baums sein muss.	Erste Verknüpfung im leeren Prototyp vorgegeben.
=	Kontextmenüs in der Liste der Bedingungen ersetzen durch Drag & Drop.	Einfügen von Bedingungen in den Baum auf Drag & Drop umgestellt.
-	Baumansicht ist nicht verständlich.	-
-	Miniaturansicht ist unnötig.	Keine Anpassung, da nur von einer Testperson als negativ empfunden.
-	Ansicht des Testausdrucks ist zu unscheinbar, sollte etwas auffälliger eingeblendet werden.	Ansicht des Testausdrucks vergrössert.

Tabelle 4: Feedback zum Prototyp (v2)

Das Feedback wurde vom Projektteam als grösstenteils positiv empfunden. Teilweise gingen die Meinungen auseinander. Das beste Beispiel dafür ist die Baumansicht, welche sehr unterschiedlich bewertet wurde. Eine Testperson gab an, dass sie mit der Baumansicht gut zurechtkommt, da sie eine ähnliche Darstellung schon von anderen Orten her kennt. Eine andere Testperson hingegen empfand die Baumansicht als zu kompliziert und würde sie entfernen oder sich zumindest eine Anleitung wünschen. Dies lässt sich auf persönliche Präferenzen zurückführen, wobei diese möglicherweise durch die Vorkenntnisse, der jeweiligen Person, mit booleschen Ausdrücken beeinflusst ist.

4.6 4. Iteration: Prototyp Version 3

Für die vierte Iteration wurde der Prototyp noch ein letztes Mal überarbeitet, basierend auf dem Feedback aus der dritten Feedbackrunde.

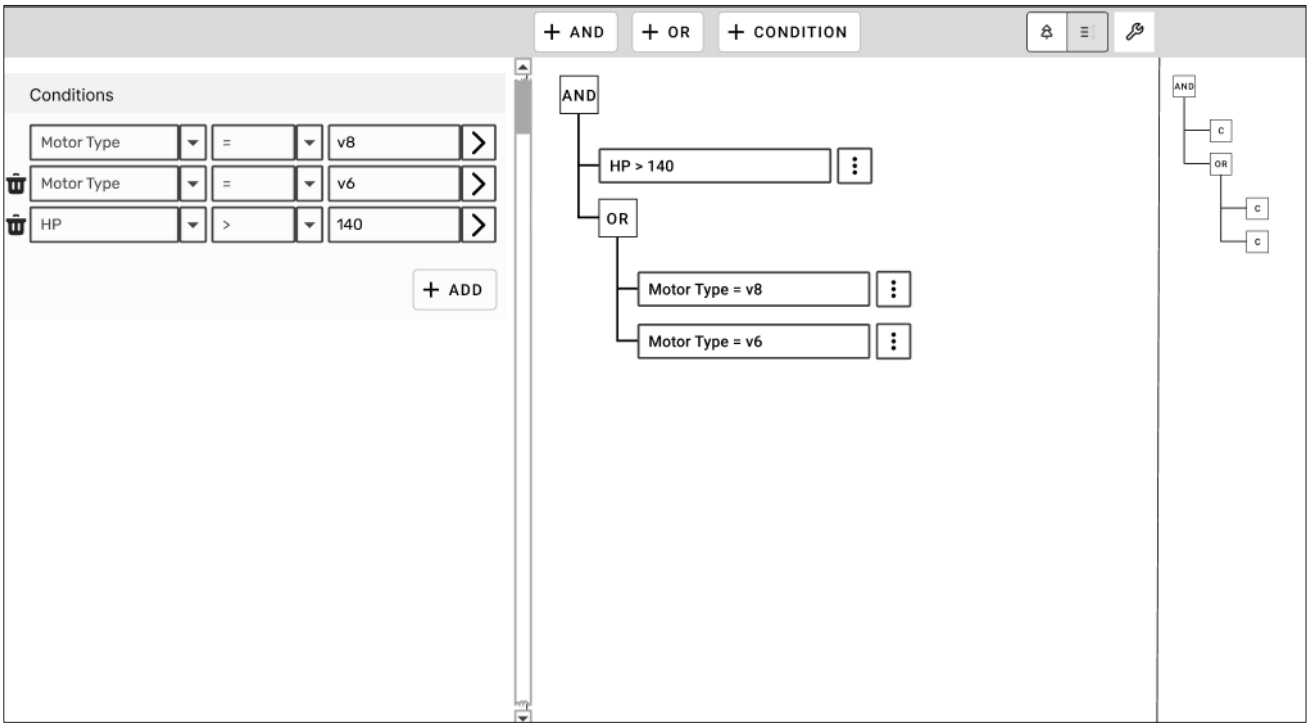


Abbildung 18: Prototyp (v3)
Quelle: Eigene Darstellung

Folgende Dinge, welche im letzten Prototyp basierend auf dem erhaltenen Feedback verbessert wurden, sind in Abbildung 18 direkt erkennbar:

- Als Icon auf der rechten Seite der Bedingungen in der Liste der Bedingungen wird nun ein Pfeil verwendet, da dies die Funktionalität (Drag & Drop) besser beschreibt.
    - Interaktivität des Prototyps: Die Bedingungen werden nicht mehr über ein Kontextmenü in den Baum eingefügt, sondern direkt per Drag & Drop
- Neu gibt es einen Lösch-Button pro Bedingung in der Liste der Bedingungen, ausser für die erste Bedingung, da das Kontextmenü, über welches die Bedingungen vorher gelöscht werden konnten, entfallen ist.

Die Interaktionen mit dem Prototyp waren von Anfang an so angedacht, dass sie mit Drag & Drop umgesetzt werden sollten. Dies wurde jedoch in den vorherigen Iterationen noch nicht umgesetzt. Diese Entscheidung wurde bewusst getroffen, da Drag & Drop Interaktionen in Figma eher verwirrend sind. Auf Grund des Feedbacks von mehreren Personen, dass das Drag & Drop Feature fehlt, wurde die Entscheidung getroffen Drag & Drop trotz möglicher Missverständnisse in der letzten Prototyp Version dennoch umzusetzen.

4.6.1 Feedback

In der letzten Iteration wurden schlussendlich Mitglieder der technischen Redaktion der STAR AG für Usertests (vgl. Kapitel 7.8.4.2) beigezogen. Durch ihr tieferes Verständnis für Filterausdrücke, welches sie im Arbeitsalltag mit GRIPS erlangt haben, waren sie ideale Testpersonen. Das erhaltene Feedback kann mit den folgenden Punkten zusammengefasst werden:

Bewertung	Kommentar	Anpassung
+	Testfeature: Supernützliche Funktion, einfache Bedienung.	-
+	Für den Filter-Editor sind verschiedene Einsatzmöglichkeiten in GRIPS vorstellbar.	-
+	Drag & Drop ist intuitiv.	-
+	Funktion, um Filterausdrücke zu vereinfachen wäre grossartig.	Dokumentiert als Weiterentwicklungsmöglichkeit.
=	Speicherbare Kombinationen auch im Baum ermöglichen statt nur in der Testfunktion.	Dokumentiert als Weiterentwicklungsmöglichkeit.
=	Die gewählten Icons könnten von einigen Nutzenden missverstanden werden.	In Umsetzung verständlichere Icons verwendet.
=	Nach der Verschiebung eines Teilbaums oder einer Bedingung markiert lassen, was als Letztes verschoben wurde. Beim Packen des nächsten Elements die Markierung wieder aufheben.	Dokumentiert als Weiterentwicklungsmöglichkeit.
=	Das Wort «Condition» wird in GRIPS bereits anderweitig verwendet. Alternativer Vorschlag: «Equipment Option».	Dokumentiert für die Integration in GRIPS.
-	Zeilendarstellung ist zu komplex bzw. unübersichtlich ist.	Keine Anpassung, da in vorherigen Feedbacks diese Ansicht gelobt wurde.

Tabelle 5: Feedback zum Prototyp (v3)

Da die Entwicklung zu diesem Zeitpunkt bereits im Gange war, wurden die Ideen und Änderungsvorschläge aus der letzten Feedbackrunde als Erweiterungsmöglichkeiten für den Filter-Editor vermerkt und nicht mehr in den Figma Prototyp integriert.

Alle Versionen des Prototyps sowie das erste Mockup können im Anhang eingesehen werden.

5 Technische Umsetzung

In diesem Kapitel wird auf verschiedene Aspekte der technischen Umsetzung des Filter-Editors eingegangen. Zuerst wird das Vorgehen beschrieben, wie das Projektteam zum Entscheid gelangte, den Filter-Editor als eine Web Component mit der Library Stencil.js zu realisieren. Anschliessend wird der Aufbau des Filter-Editors beschrieben, bevor auf die vom Filter-Editor angebotene Schnittstelle eingegangen wird. Danach wird die gewählte Ordnerstruktur erläutert, worauf eine Übersicht der implementierten End-To-End Tests folgt. Zum Abschluss werden dann noch die Ergebnisse aus der Auswertung der Usability Tests präsentiert.

5.1 Vorgehen

Bevor mit der Implementierung des Filter-Editors begonnen werden konnte, musste entschieden werden mit welcher Technologie der Prototyp gebaut werden soll. Da der, im Rahmen des «Smart Knowledge Capture»-Forschungsprojekts entwickelte, Prototyp mit React implementiert wurde, gab es zwei Möglichkeiten. Die erste Möglichkeit war es ebenfalls React zu verwenden. Die zweite sich anbietende Möglichkeit war auf Web Components zu setzen. Die Entscheidung durfte das Projektteam selbst treffen. Die Empfehlung vom Betreuer als auch vom Hauptverantwortlichen des Forschungsprototyps war jedoch, den Filter-Editor als eine Web Component zu entwickeln.

5.1.1 Entscheidung für Web Component

Da beide Teammitglieder keinerlei Erfahrungen mit Web Components hatten, wurde erst einmal analysiert, welche Vor- und Nachteile Web Components haben. An dieser Stelle wird auf jeweils drei zentrale Vor- und Nachteile von Web Components eingegangen:

Vorteile:

- Web Components basieren auf der Kombination von drei Technologien (Custom Elements, Shadow DOM und HTML-Templates), welche alle native Bestandteile des HTML-Standards sind. Dies hat zur Folge, dass Web Components einerseits von allen gängigen modernen Browsern unterstützt werden (siehe Abbildung 19) und andererseits, dass Web Components ohne zusätzliche JavaScript-Library verwendbar sind. (Borodanov (2021))

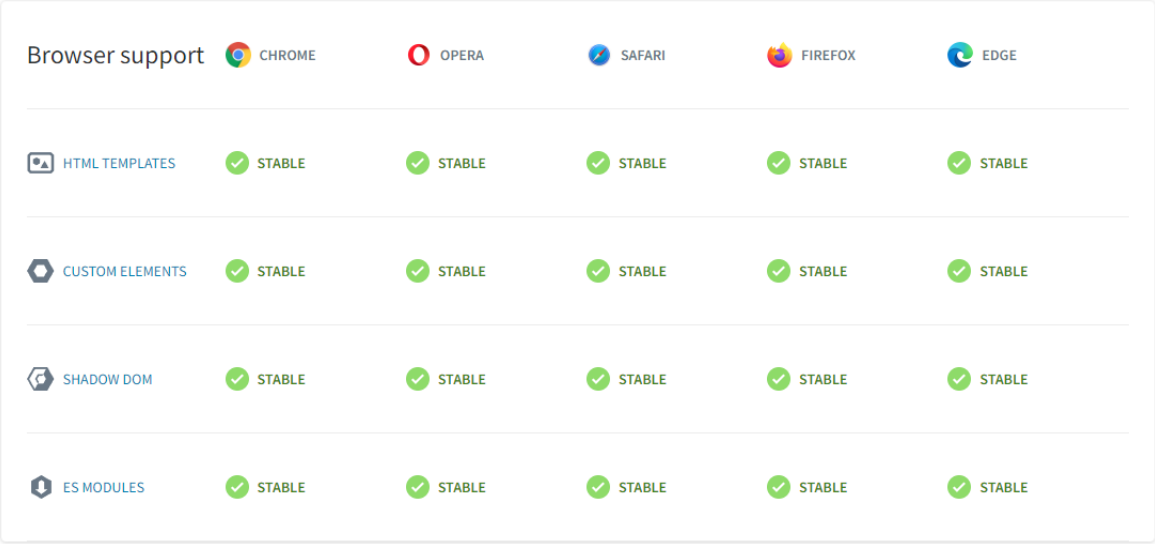


Abbildung 19: Browser Support von Web Components
Quelle: https://www.webcomponents.org/

- Web Components können in allen gängigen Frameworks (React, Vue, Angular) oder auch ohne Framework genutzt werden. Dies macht Web Components flexibel einsetzbar und so lassen sich Web Components in verschiedenen Projekten mit unterschiedlichen Technologie-Stacks wiederverwenden. (Ramirez (2022)) React-Komponenten auf der anderen Seite können nur in einer React-Umgebung wiederverwendet werden (Borodanov (2021)).
- Web Components verwenden den Shadow DOM, welcher das DOM der Web Component isoliert. Dies hat den Vorteil, dass das zur Web Component definierte Styling (CSS) nur die Web Component selbst betrifft und auch kein Styling von ausserhalb in das Web Component eindringen kann. Des Weiteren sorgt die Kapslung eines Web Components auch dafür, dass DOM-Knoten innerhalb des Web Components von ausserhalb mit document.querySelector() nie selektiert werden. (Foo (2022)) Somit agiert eine Web Component als eigenständiges Element nach aussen.

Nachteile:

- Web Components sind weniger verbreitet als die gängigen Web-Entwicklungs-Frameworks und deshalb gibt es weniger Informationen zur Entwicklung von Web Components als zu beispielsweise React im Web. Die Community ist auch wesentlich kleiner. Bei allfälligen Problemen wird es wahrscheinlich zeitaufwändiger Lösungsansätze für ein Problem zu finden.
- Weil ein Web Component ein Custom Element ist und es erst mit JavaScript registriert werden muss, kann es einige Millisekunden dauern, bis das Web Component schliesslich gerendert wird. Während diesem Zeitraum bleibt das Web Component ungestaltet oder versteckt. Aus diesem Grund sind Web Components nicht die optimale Wahl für Webseiten, die nur einmal besucht werden. Für Webseiten, die mehrmals besucht werden, ist dies aufgrund von Caching weniger ein Problem (Borodanov (2021)).
- Web Components sind aktuell noch SEO unfreundlich und Webseiten, welche sie verwenden, können an ineffizienter Indexierung leiden (Borodanov (2021), Ramirez (2022)).

Das Projektteam hat sich vorwiegend mit den potenziellen Nachteilen auseinandergesetzt und bewertet diese aus folgenden Gründen als vernachlässigbar:

- Die geringere Informationsdichte zu Web Components wird zu mehr Aufwand führen, was das Projektteam aber nicht abschreckt.
- Das langsame erstmalige Rendern ist nicht von grosser Bedeutung, weil der Filter-Editor zur Bearbeitung von Filterausdrücken wiederholt aufgerufen wird.
- Die Probleme bezüglich SEO sind nicht entscheidend, da es sich beim Filter-Editor um ein Feature handelt, welches keine grosse SEO-Relevanz hat.

Aus diesen Gründen spricht aus Sicht des Projektteams nichts gegen die Empfehlung des Betreuers, auf eine Web Component zu setzen.

Nach einem ersten Versuch eine simple Web Component über die Standard API für Web Components zu implementieren, hat sich schnell gezeigt, dass eine Realisierung des Filter-Editors auf diese Art sehr zeitaufwendig werden dürfte. Die Standard API stellte sich als komplex und nicht sehr entwicklungsfreundlich heraus. Aus diesem Grund wurde eine Library gesucht, welche die Entwicklung von Web Components erleichtert.

5.1.2 Entscheidung für Web Component Library

Eine erste Analyse hat ergeben, dass es viele Libraries gibt, welche die Erstellung von Web Components erleichtern. Unter anderem können folgende Libraries zur Entwicklung von Web Components genutzt werden: Hybrids, Lit, Polymer, Skate.js, Slim.js und Stencil.js (Webcomponents.org (2023)).
Das Projektteam hat sich daraufhin entschieden die beiden Libraries Lit und Stencil.js zu evaluieren. Dies vor allem aus dem Grund, dass diese beiden Libraries aktiv gepflegt werden und eine grössere Anzahl an Mitwirkenden aufweisen, was bei den anderen erwähnten Libraries nicht der Fall ist. Zusätzlich ist für beide Libraries eine ausführliche Dokumentation verfügbar, was den Einstieg erleichtert.

Um eine abschliessende Entscheidung zwischen Lit und Stencil.js zu treffen, wurde mit beiden eine minimale Komponente entwickelt. Dabei hat sich wie erwartet gezeigt, dass beide Libraries geeignet sind um den Filter-Editor als Web Component zu implementieren. Am Ende fiel der Entscheid aus den folgenden Gründen zugunsten von Stencil.js aus:

- Bei Stencil.js funktioniert sehr vieles «out of the box». Vom Builden über das Testen bis hin zum Publizieren des Web Components über NPM wird einem viel Arbeit abgenommen. Bei Lit hingegen muss man sich um wesentlich mehr selbst kümmern. Aufgrund des eingeschränkten Zeitbudgets im Rahmen der Arbeit spricht dies klar für den Einsatz von Stencil.js.
- Während dem Prototyping wurde festgestellt, dass sich die «Developer Experience» mit Stencil.js wesentlich angenehmer anfühlt als mit Lit. Der Compiler von Stencil.js hilft bei der Entwicklung sehr und erinnert von der «Developer Experience» stark an eine Mischung aus React und Angular, mit welchen beide Teammitglieder bereits Erfahrungen sammeln konnten.
- Stencil.js vereint einige der besten Features aus React als auch Angular. Beispielsweise wird eine Komponente, wie in Angular, mit einer Annotation definiert. Innerhalb der Komponente kann mit JSX, was aus React kommt, gearbeitet werden. Da beide Teammitglieder mit beiden Frameworks bereits Erfahrung mitbringen, ist eine schnelle Einarbeitung möglich. Bei Lit hingegen wäre die Einarbeitung aufwändiger, da Lit stark an React angelehnt ist und sich der Erfahrungslevel der Teammitglieder in dieser Technologie stark unterscheidet.
- Bei Stencil.js kann das Styling für ein Web Component in einem separaten .css-File abgelegt werden. Dadurch ist die saubere Trennung von Logik und Styling möglich, was unter anderem zu einer besseren Übersicht führt. Bei Lit hingegen ist vorgesehen, dass das Styling direkt im .ts-File des Web Components als statische Variable erfasst wird, wodurch die Übersichtlichkeit sinkt.

5.2 Aufbau des Filter-Editors

Der realisierte Prototyp wurde als eine Komponente, welche *skc-filter-editor* heisst, erstellt. Der *skc-filter-editor* setzt sich aus mehreren Subkomponenten zusammen (siehe Abbildung 20). Nachfolgend wird die Funktion der einzelnen Subkomponenten beschrieben. Es gibt dabei noch eine zweite Ansicht die Zeilenansicht, welche in Abbildung 20 nicht sichtbar ist.

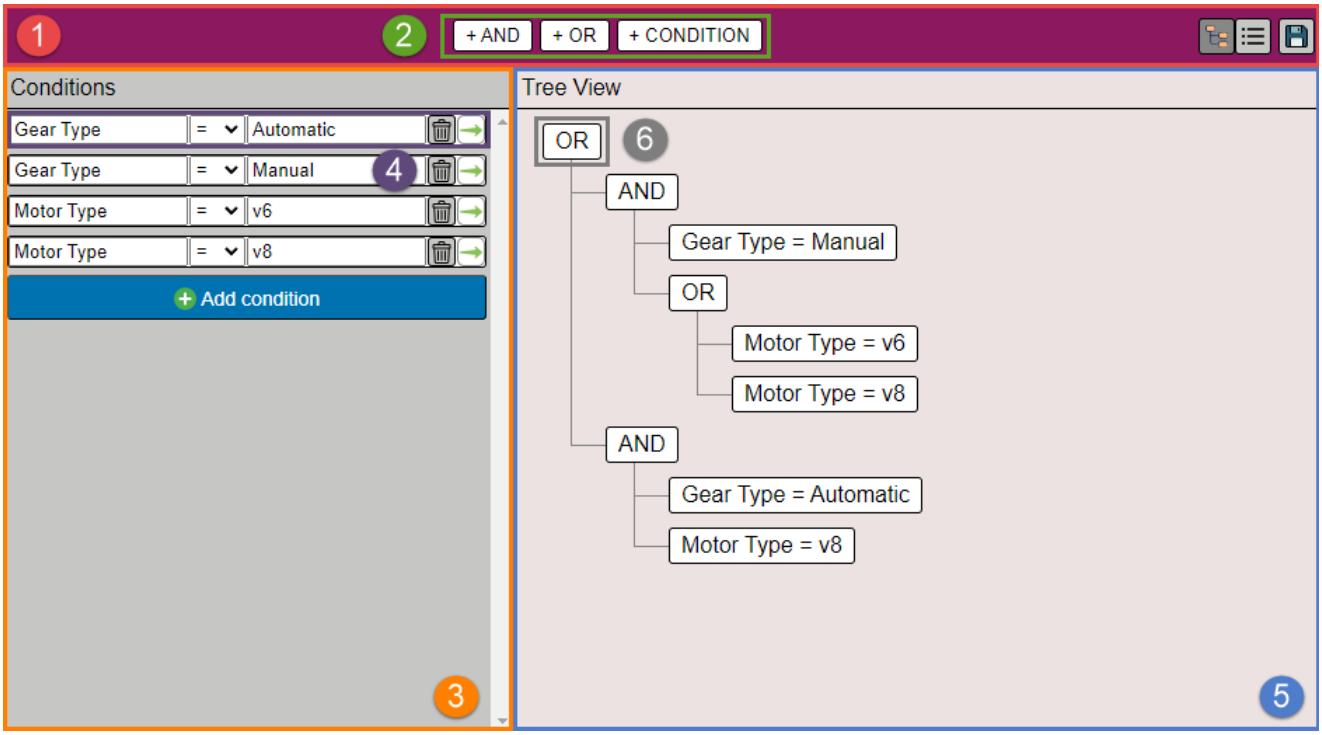


Abbildung 20: Aufbau des Filter-Editors
Quelle: Eigene Darstellung

Nr.	Komponente	Beschreibung
1	skc-menu	Die Menükomponente besteht aus der Toolbar (mitte), einem Switch und einem Button zum Speichern (rechts). Der Switch ermöglicht das Umschalten zwischen Baum- und Zeilenansicht. Der Button zum Speichern ermöglicht den Transport des Filterausdrucks nach Aussen über ein Event (siehe Kapitel 5.3.3).
2	skc-toolbar	Die Toolbar enthält drei Elemente, welche per Drag & Drop in die Baumansicht gezogen werden können, um den Filterausdruck aufzubauen.
3	skc-condition-list	Die Liste der Bedingungen erlaubt es Bedingungen separat zu erfassen und zu bearbeiten.

4	skc-condition-item	Repräsentiert eine einzelne Bedingung, welche per Drag & Drop in die Baumansicht gezogen werden kann, um den Filterausdruck aufzubauen. Die in den Baum gezogene Bedingung ist anschliessend mit dem Baum verknüpft, so dass eine Änderung in der Bedingung direkt im Baum übernommen wird.
5	skc-tree-view	Die Baumansicht stellt den Filterausdruck als Baum dar.
6	skc-tree-node	Stellt eine Verknüpfung oder eine Bedingung des Filterausdrucks im Baum in Form eines Knotens dar. Jeder Knoten lässt sich dabei innerhalb des Baums per Drag & Drop verschieben. Dazu gibt es auf jedem Knoten ein Kontextmenü, über welches sich eine Verknüpfung invertieren lässt (AND zu OR und umgekehrt). Zusätzlich können über das Kontextmenü Knoten wieder aus dem Baum entfernt werden.
*	skc-line-view	Die Zeilenansicht stellt den Filterausdruck in Textform dar.
*	skc-line-item	Repräsentiert eine Verknüpfung oder eine Bedingung des Filterausdrucks in Textform. Ergänzt dabei die notwendigen Klammern am richtigen Ort. Bei Mouse-Over einer Klammer wird die Klammer und ihr Gegenstück hervorgehoben.

Tabelle 6: Subkomponenten des Filter-Editors

Die mit einem * gekennzeichneten Elemente sind in Abbildung 20 nicht sichtbar.

5.3 Schnittstelle des Filter-Editors

Die Web Component soll später in unterschiedliche Anwendungen integriert werden. Hierfür ist eine Schnittstelle erforderlich, über welche Filterausdrücke in den Filter-Editor importiert und danach wieder exportiert werden können.
Der Filter-Editor bietet zwei Properties an (height (Höhe des Elements) und data (Filterausdruck)), welche gesetzt werden können. Dazu kann ein Event auf dem Filter-Editor abonniert werden, welches bei der Speicherung innerhalb des Filter-Editors, den Filterausdruck aus dem Filter-Editor zurückliefert. Des Weiteren können auf dem Filter-Editor viele CSS-Variablen überschrieben werden, um den Filter-Editor nach eigenen Bedürfnissen zu gestalten.

5.3.1 Height-Property

Über das «height»-Property lässt sich die Höhe des Filter-Editors festlegen. Es wird eine Zahl erwartet, welche in Pixel angewendet wird. Die Standardhöhe des Filter-Editors ist 520 Pixel.

Das «height»-Property lässt sich direkt im HTML auf dem Filter-Editor setzen (siehe Abbildung 21):

```
<skc-filter-editor id="filter-editor" height="500"></skc-filter-editor>
```

Abbildung 21: Setzen des Height-Property
Quelle: Eigene Darstellung

5.3.2 Data-Property

Über das «data»-Property lässt sich ein Filterausdruck in den Filter-Editor importieren. Erwartet wird ein Array, welches mindestens ein *FilterEditorDataElement* enthält. Die Schnittstellen-Klasse befindet sich im Git-Repository unter *src/interface/FilterEditorDataElement.ts* und kann theoretisch zur Hilfe verwendet werden, um die Daten für den Filter-Editor aufzubereiten. Dies ist allerdings nicht notwendig, da man das Array auch direkt ohne Typisierung erstellen kann, dabei müssen allerdings die folgenden Regeln befolgt werden, damit der Import des Filterausdrucks funktioniert.

Ein einzelnes *FilterEditorDataElement* beinhaltet folgende Felder:

Nr.	Feld	Typ	Beschreibung	Erlaubte Werte
1	type	Number	Typ des Elements	1 für AND, 2 für OR, 3 für eine Bedingung
2	key	Number	Schlüssel der für die Verknüpfung der Elemente benötigt wird	Einmaliger Schlüssel innerhalb des Arrays.
3	parentKey	Number	Verweis auf den Schlüssel des Parent-Elements	Gültiger Schlüssel innerhalb des Arrays (es darf nur auf ein Element vom Typ 1 oder 2 verwiesen werden).
4	property	String	Bezeichnung der Eigenschaft einer Bedingung	Beliebiger String
5	operator	Number	Operator einer Bedingung	1 für =, 2 für !=, 3 für >, 4 für >=, 5 für <, 6 für <=
6	value	String	Wert für die Eigenschaft einer Bedingung	Beliebiger String

Tabelle 7: Felder der Schnittstellen-Klasse des Filter-Editors

Weitere Punkte, welche es zu beachten gilt:

- Es muss genau ein Root-Element im Array geben, bei welchem kein Parent-Key gesetzt ist. Dieses Element muss zudem vom Typ 1 oder 2 sein.
- Es darf kein Element im Array geben, ausser dem Root-Element, welches nicht über eine Reihe von Parent-Keys am Ende zum Root-Element gelangt.
- Die Felder 4-6 sind nur bei Bedingungen (Typ = 3) zwingend zu erfassen, bei Typ 1 und 2 werden sie ignoriert.

Das «data»-Property lässt sich über JavaScript setzen (siehe Abbildung 22):

```
const filterEditor = document.getElementById( elementId: 'filter-editor');
filterEditor['data'] = data;
```

Abbildung 22: Setzen des Data-Property
Quelle: Eigene Darstellung

Für ein besseres Verständnis zeigt Abbildung 24 ein Beispiel-Array, welches an den Filter-Editor über das «data»-Property übergeben wird. In Abbildung 23 ist das Resultat im Filter-Editor ersichtlich.

```
const data = [
    {
        type: 1,
        key: 1,
    },
    {
        type: 3,
        key: 4,
        parentKey: 3,
        property: 'Motor Type',
        operator: 1,
        value: 'v8',
    },
    {
        type: 3,
        key: 5,
        parentKey: 3,
        property: 'Motor Type',
        operator: 1,
        value: 'v6',
    },
];
```

Abbildung 24: Beispiel Inputdaten
Quelle: Eigene Darstellung

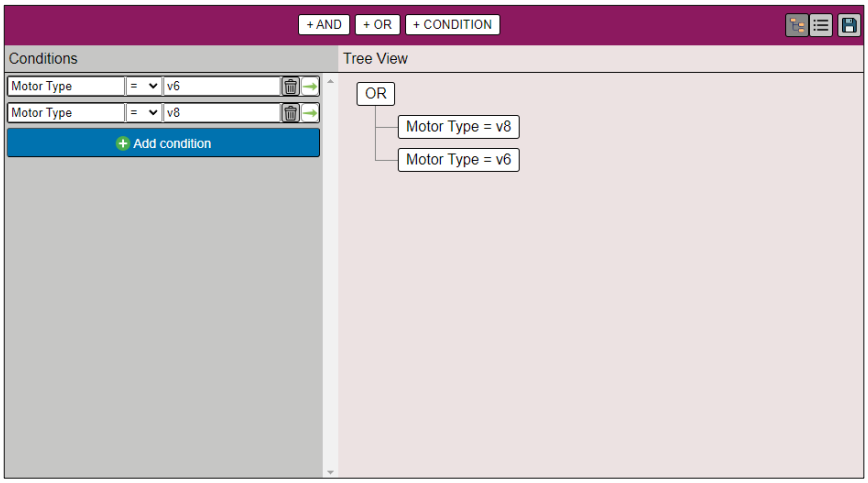


Abbildung 23: Filter-Editor nach Import
Quelle: Eigene Darstellung

5.3.3 DataSaved-Event

Der Filter-Editor bietet ein Event «dataSaved», auf welchem ein Event-Listener registriert werden kann. Dieses Event wird jeweils ausgelöst, wenn im Filter-Editor auf den Speichern-Button geklickt wird. Dabei wird über das Event der Filterausdruck als Array von *FilterEditorDataElement* zurückgeliefert. Das Format des gelieferten Filterausdrucks entspricht dem Format, welches beim Import verlangt wird (vgl. Kapitel 5.3.2).

Das Event «dataSaved» lässt sich über JavaScript abonnieren (siehe Abbildung 25):

```
const filterEditor = document.getElementById( elementId: 'filter-editor');
filterEditor.addEventListener( type: 'dataSaved',
     listener: function (event: CustomEvent<FilterEditorDataElement[]>) {
    console.log(JSON.stringify(event.detail));
});
```

Abbildung 25: Abonnieren des DataSaved-Events
Quelle: Eigene Darstellung

5.3.4 Styling über CSS-Variablen

Der Filter-Editor verwendet «Arial» als Schriftart und die verwendeten Farben orientieren sich am OST Corporate Designs Leitfaden. Die Schriftart als auch die Farben lassen sich von ausserhalb des Filter-Editors mit Hilfe von CSS-Variablen übersteuern. Alle verfügbaren CSS-Variablen können im Git-Repository dem CSS-File *src/components/skc-filter-editor/skc-filter-editor.css* entnommen werden.

Die verfügbaren CSS-Variablen lassen sich über JavaScript übersteuern (siehe Abbildung 26):

```
const filterEditor = document.getElementById( elementId: 'filter-editor');
filterEditor.style.setProperty( property: '--menu-bg-color',  value: '#6b3881');
```

Abbildung 26: Übersteuern der CSS-Variablen
Quelle: Eigene Darstellung

5.4 Verzeichnisstruktur

Das Projekt wurde mit dem, von Stencil.js zur Verfügung gestellten, Starter-Script initialisiert. Die Projektstruktur (siehe Abbildung 27) wurde anschliessend um eigene untergeordnete Ordner ergänzt, um eine bessere Strukturierung reinzubringen. Die hellgrauen Ordner sind Output-Ordner, auf die nicht eingegangen wird. Nachfolgend die Beschreibungen zum Ordner «src» sowie allen seinen untergeordneten Ordnern.

Ordner «src»: Enthält alle relevanten Codeartefakte für den Filter-Editor. Die beiden Files «index.html» und «index.ts» erlauben es, die Schnittstelle des Filter-Editors (vgl. Kapitel 5.3) direkt zu testen.

Ordner «components»: Enthält die Hauptkomponente <skc-filter-editor> sowie alle in Kapitel 5.2 beschriebenen Subkomponenten.

Unterordner «assets»: Enthält die Icons, welche innerhalb der einzelnen Subkomponenten im Filter-Editor verwendet werden.

Ordner «interface»: Enthält die Schnittstellen-Klasse für den Filter-Editor, welche für den Import (vgl. Kapitel 5.3.2) und Export (vgl. Kapitel 5.3.3) eines Filterausdrucks verwendet wird.

Ordner «model»: Enthält die Model-Klassen, welche die Business Logik des Filter-Editors enthalten.

Ordner «utils»: Enthält zwei Klassen, eine für den Import eines Filterausdrucks in den Filter-Editor und eine weitere für den Export eines Filterausdrucks aus dem Filter-Editor.

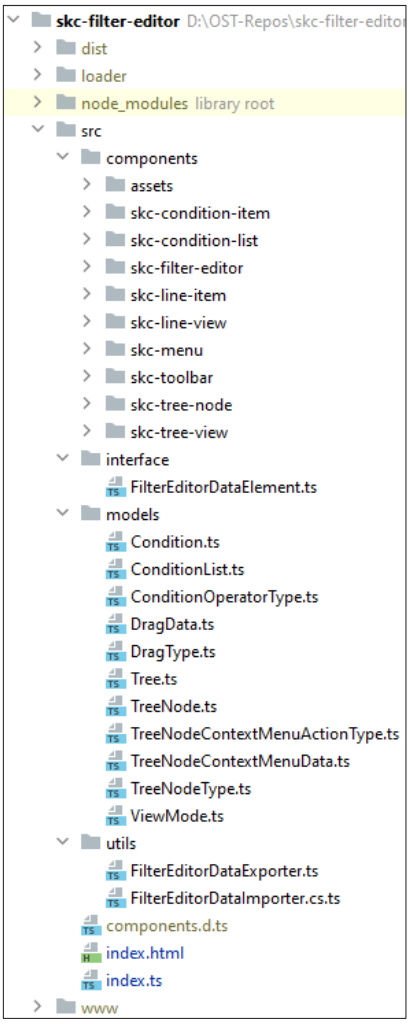


Abbildung 27: Verzeichnisstruktur
Quelle: Eigene Darstellung

5.5 End-to-End Tests

In der nachfolgenden Tabelle sind alle implementierten End-To-End Tests dokumentiert. Die Tests werden in die zwei Kategorien «Anzeige» und «Funktion» eingeteilt. Alle Tests der Kategorie «Anzeige» sind dazu da, um zu testen, ob der Filter-Editor richtig aufgebaut wird und ob alle Elemente am richtigen Ort in der Hierarchie erzeugt werden. Die Tests der Kategorie «Funktion» hingegen, sind fortgeschrittenere Tests, welche Operationen, wie beispielweise eine Manipulation am Baum, testen.

Nr.	Beschreibung	Erwarteter Endzustand	Kategorie
1	Filter-Editor laden.	Der Filter-Editor ist erzeugt und «hydriert».	Anzeige
2	Root div suchen.	Das oberste div mit der Klasse «root» ist erzeugt und auffindbar.	Anzeige
3	Liste mit Bedingungen laden.	Liste mit Bedingungen ist erzeugt und auffindbar.	Anzeige
4	Menü suchen.	Die Menüleiste ist erzeugt und auffindbar.	Anzeige
5	Baumansicht suchen.	Standardmässig ist in der Arbeitsfläche die Baumansicht verfügbar.	Anzeige
6	Zeilenansicht-Button suchen und anklicken. Mit der Maus die erste Klammer hovern.	Innerhalb der Arbeitsfläche ist die Zeilenansicht verfügbar. Beim hovern der ersten Klammer wird bei der dazugehörigen schliessenden Klammer die Klasse «highlight» gesetzt.	Anzeige
7	Liste mit Bedingungen erzeugen.	Die Liste mit Bedingungen auf der linken Seite hat, nach der initialen Erzeugung, genau einen Eintrag.	Anzeige
8	«Add Condition» Button anklicken.	Nach dem Klicken auf den «Add Condition» Button besteht die Liste der Bedingungen aus zwei Einträgen.	Funktion
9	Leere Bedingung in den Baum ziehen.	Nach dem Drag & Drop einer leeren Bedingung in den Baum, ist ein Knoten mit dem Text «=» im Baum auffindbar.	Funktion
10	Leere Bedingung in den Baum ziehen. Erstes Input-Feld der Bedingung mit dem Text «test» befüllen.	Nach dem Drag & Drop einer leeren Bedingung in den Baum, besteht der Baum aus zwei Knoten. Nach dem Befüllen des Textes, in der Bedingung, erscheint beim vorherig erzeugten Knoten der Text «test=».	Funktion

11	Leere Bedingung in den Baum ziehen. Wurzelknoten mit Rechtsklick anklicken. Im Kontextmenü den Menüpunkt «Remove all branches» anklicken.	Baum enthält nur noch den Wurzelknoten.	Funktion
12	UND-Verknüpfung in den Baum ziehen.	Nach dem Drag & Drop einer UND-Verknüpfung in den Baum, enthält der Baum als Kind des Wurzelknotens einen UND- Knoten.	Funktion
13	Leere Bedingung und UND-Verknüpfung in dieser Reihenfolge in den Baum ziehen. Die Bedingung vom Wurzelknoten auf neuen UND-Knoten umhängen.	Nach dem Einfügen der Knoten per Drag & Drop in den Baum hat dieser drei Knoten mit folgenden Texten: «AND», «=», «AND». Nach dem Umhängen mittels Drag & Drop hat der Baum drei Knoten mit den gleichen Texten aber in anderer Reihenfolge: «AND», «AND», «=».	Funktion
14	Wurzelknoten mit Rechtsklick anklicken. Im Kontextmenü den Menüpunkt «Change to OR» anklicken.	Wurzelknoten hat den Text «OR».	Funktion
15	Leere Bedingung und UND-Verknüpfung in den Baum ziehen. Die Bedingung vom Wurzelknoten auf neuen UND-Knoten umhängen. Neuen UND-Knoten mit Rechtsklick anklicken. Im Kontextmenü den Menüpunkt «Remove branch» anklicken.	Baum enthält nur noch den Wurzelknoten.	Funktion

Tabelle 8: End-to-End Tests

5.6 Auswertung der Usability Tests

Durch die Usability Tests (vgl. 7.8.4.3) konnte das Projektteam evaluieren, wie benutzungsfreundlich der entwickelte Filter-Editor in Form einer Web Component ist. An dieser Stelle folgt eine Zusammenfassung der ausgewerteten Usability Tests. Dabei wird zuerst auf die Probleme bei der Durchführung der Usability Tests eingegangen, bevor die eigentlichen Ergebnisse präsentiert werden.

Probleme bei der Durchführung der Usability Tests:
Die Testpersonen konnten im Verlauf des Usability Tests nach jeder ausgeführten Aufgabe ihre Zufriedenheit mit einer Zahl zwischen 1 (sehr unzufrieden) und 5 (sehr zufrieden) bewerten. Es wurde festgestellt, dass das Verständnis der Aufgabenstellung, zum Teil einen Einfluss auf die Bewertung der Zufriedenheit hatte. So hat eine der Testpersonen eine Aufgabe mit einer 1 bewertet und nachträglich mitgeteilt, dass die Benutzungsfreundlichkeit an sich eigentlich gegeben ist, es sei nur die Aufgabenstellung zu wenig klar formuliert. Eine weitere Testperson hätte sich zudem Schlüsselbegriffe, wie «Bedingung» in der Aufgabenstellung in englischer Sprache gewünscht. Dies weil die Beschriftung innerhalb des Web Components ebenfalls in englischer Sprache ist und somit die Möglichkeit für Missverständnisse reduziert werden kann.
Es gilt also festzuhalten, dass das Protokoll für die Usability Tests nicht optimal war und deshalb zum Teil die Bewertungen der Testpersonen dadurch negativ beeinflusst wurden.

Ergebnisse:
Insgesamt war das Feedback, unter Berücksichtigung des oben erwähnten Umstandes, beinahe durchgängig positiv. So wurden alle Funktionen, welche der Filter-Editor in seiner aktuellen Form anbietet, nachdem sie verstanden wurden, als benutzungsfreundlich eingestuft.

Folgende zusätzlichen Erkenntnisse konnten aus den Usability Tests gewonnen werden:
- Das Einfügen von Bedingungen in den Baum per Drag & Drop war nicht für alle Testpersonen sofort klar. Einige Testpersonen hätten sich eine Anleitung oder zumindest Tooltipps gewünscht.
- Alle Testpersonen sind entweder durch Intuition oder durch Ausprobieren darauf gekommen, dass die Knoten innerhalb des Baums ein Kontextmenü besitzen, welches per Rechtsklick geöffnet werden kann. Einige Testpersonen hätten sich trotzdem eine Anleitung gewünscht.
- Eine Testperson schlug eine Weiterentwicklungsmöglichkeit vor. Die Idee wäre es Klammerpaare in der Zeilenansicht beim hovern mit verschiedenen Farben zu hinterlegen. Dadurch wäre für Nutzende noch eindeutiger, was sie gerade betrachten.

Die Erkenntnisse aus den Usability Tests sollten nachträglich berücksichtigt werden, um den Filter-Editor weiter zu optimieren. Im Rahmen des Projekts war dies nicht mehr möglich, da die Usability Tests erst gegen Ende des Projekts durchgeführt werden konnten. Auf eine Integration der Erkenntnisse in das Kapitel 6.3 «Weiterentwicklung» wurde verzichtet.

6 Resultate und Ausblick

Dieses Kapitel geht auf die Resultate und den Ausblick ein. Zuerst wird im Unterkapitel Zielerreichung auf die gesteckten und erreichten Ziele eingegangen. Danach wird das Vorgehen der Arbeit reflektiert, bevor im Anschluss diverse Weiterentwicklungsmöglichkeiten vorgestellt werden. Zum einen sind dies Konzepte, welche auf Grund der begrenzten Zeit für das Projekt, nicht umgesetzt werden konnten. Zum anderen sind es weitere Ideen und Verbesserungsansätze, welche im Verlauf des Projekts aufgetaucht sind. Danach wird auf die potenzielle Integration in GRIPS eingegangen und was es da zu beachten gibt. Zum Abschluss wird dann noch ein alternativer Ansatz, zum im Projekt umgesetzten Prototyp, vorgestellt, welcher vom Projektteam verworfen wurde.

6.1 Zielerreichung

Das Projekt hatte die folgenden zwei Hauptziele:

1. Entwicklung eines benutzungsfreundlichen Filter-Editors, welcher es ermöglicht beliebig komplexe Ausdrücke zu erstellen.
2. Zusätzliche Hilfsfunktionalitäten erarbeiten, welche die Nutzenden bei der Erstellung von Filterausdrücken zusätzlich unterstützen.

Das Projektteam hat sich dazu entschlossen das erste Ziel als erreicht anzusehen, wenn das Feedback aus den Usertests zum Figma Prototyp sowie aus den Usability Tests zur Web Component grösstenteils positiv ausfällt.
Diese Entscheidung basiert einerseits darauf, dass die Tests mit Nutzenden, welche sehr unterschiedliche Kenntnisse im Themengebiet haben, durchgeführt wurden. Andererseits auf der Erkenntnis, dass bei jeder Iteration gegensätzliches Feedback dabei war. Das beste Beispiel hierfür sind die Meinungsverschiedenheiten bezüglich der Baumansicht, so empfanden einige der Testpersonen die Baumansicht als nützlich und übersichtlich, andere Testpersonen hingegen nicht. Da jedoch sowohl bei den Usertests als auch bei den Usability Tests schlussendlich das positive Feedback überwog, sieht das Projektteam das erste Ziel als erfüllt an.

Das zweite Ziel ist aus Sicht des Projektteams ebenfalls erfüllt. So wurden während des Projekts viele Weiterentwicklungsmöglichkeiten für den gebauten Prototyp erarbeitet, welche man noch umsetzen könnte. Im nachfolgenden Kapitel werden diese Weiterentwicklungsmöglichkeiten, welche die Nutzenden bei der Erstellung von Filterausdrücken noch besser unterstützen, kurz beschrieben.

6.2 Reflexion zum Vorgehen

Die nachfolgende Reflexion bezieht sich auf das gesamte Projekt und dessen Ablauf. Trotz der Erreichung der beiden Hauptziele gibt es auch Verbesserungspotenzial für die Durchführung einer Arbeit dieser Art.

Aus Sicht des Projektteams war es eine gute Entscheidung als Erstes mit einer Literaturrecherche und der Entwicklung von ersten Konzepten für den Filter-Editor zu beginnen, um besser abschätzen zu können, in welche Richtung sich das Projekt entwickeln soll.
In der darauffolgenden Prototyping-Phase wurde zuerst mit einem Mockup, welches nur innerhalb des Projektteams und mit dem Betreuer besprochen wurde, gestartet. Dieses Vorgehen brachte den grossen Vorteil, dass Missverständnisse bezüglich des Ziels der Arbeit bereits früh erkannt und geklärt werden konnten. Ebenfalls war es eine gute Entscheidung bereits die erste Version des Prototyps mit Fachleuten der STAR AG zu besprechen. Dadurch konnten GRIPS spezifische Anforderungen schon früh im Projektverlauf in Erfahrung gebracht, als auch weiteres Feedback der Fachleute der STAR AG im Prototyp berücksichtigt werden.
Im Allgemeinen war das menschenzentrierte Design, aus Sicht des Projektteams, für das Projekt als Vorgehen hervorragend geeignet, weshalb es für ähnliche Projekte weiterempfohlen werden kann. So kam es durch die Anwendung des menschenzentrierten Designs in jeder Iteration zu produktiven Auseinandersetzungen mit dem Prototyp, wodurch dieser mit jeder Iteration weiter verbessert werden konnte.
Verbesserungspotenzial sieht das Projektteam hingegen bei den durchgeführten Usability Tests. Dies wurde bereits im Kapitel 5.6 diskutiert.
Bei der technischen Umsetzung war das Projektteam durch die zahlreichen Erfahrungen, welche sie in der Privatwirtschaft bereits sammeln konnten, sowie die bereits gemeinsam durchgeführten Projektarbeiten während des Studiums, bestens gerüstet, um den Prototypen effizient umzusetzen. Dies führte dazu, dass von Anfang an klar war, wie die Infrastruktur aussehen soll, weshalb dies dann auch keine Probleme machte. Eine der grössten Herausforderungen bei der Umsetzung war das Styling des Web Components. Für ähnliche Projekte ist das Projektteam der Meinung, dass es sich lohnen würde, bei der Technologiewahl bereits vorzeitig zu evaluieren, wie wichtig das Design für das zu entwickelnde Produkt ist, da Web Components grundsätzlich selbst per CSS gestaltet werden müssen, was mit mehr Aufwand verbunden ist.

6.3 Weiterentwicklung

Während des Projekts konnte ein umfangreicher Prototyp in Form einer Web Component entwickelt werden. Dennoch gibt es einige Konzepte und weitere Verbesserungsmöglichkeiten, welche auf Grund der zeitlichen Beschränkung des Projekts nicht realisiert, werden konnten. Nachfolgend wird kurz beschrieben, wie diese in einem Folgeprojekt umgesetzt werden könnten. Am Ende wird dann noch auf die Herausforderungen eingegangen, welche bei einer Integration in GRIPS zu beachten sind.

6.3.1 Testen der Ausdrücke

Um das Testen von Ausdrücken, wie im Kapitel 3.6 «Testing und Highlighting» beschrieben, zu ermöglichen, würde eine Funktion benötigt, welche die einzelnen Knoten des Baums besucht und überprüft, ob der eingegebene Testausdruck beim besuchten Knoten akzeptiert oder abgelehnt wird. Des Weiteren müsste bei allen Teilbäumen aufgrund der Resultate ihrer untergeordneten Knoten geprüft werden, ob sie als wahr oder falsch ausgewertet werden. Eine solche Funktion könnte mit Hilfe des Visitor-Patterns umgesetzt werden.
Wie die UI-Ergänzungen für dieses Feature innerhalb der Web Component umgesetzt werden könnten, kann dem letzten Figma Prototyp im Anhang entnommen werden.

6.3.2 Vereinfachung der Ausdrücke

Das im Kapitel 3.7 beschriebene Konzept «Vereinfachung von Ausdrücken» benötigt einen entsprechenden Algorithmus. Karnaugh Maps (K-Maps) ist eine Standardmethode zur Vereinfachung von booleschen Ausdrücken. K-Maps eignet sich dabei gut für die manuelle Lösung von kleineren Problemen mit bis zu vier Variablen. Bei mehr Variablen wird die Anwendung von K-Maps jedoch bereits zu komplex. Für boolesche Ausdrücke mit mehr als vier Variablen sollte deshalb die Quine-McCluskey Methode verwendet werden. Diese Methode bietet eine automatisierbare Lösung für die Vereinfachung von booleschen Ausdrücken. (Bosworth (2010))
Das Projektteam empfiehlt die Quine-McCluskey Methode für eine Integration in den Filter-Editor zu evaluieren. Bei einer allfälligen Implementierung sollte dabei auch beachtet werden, dass es eine Möglichkeit braucht den Vereinfachungsvorgang wieder rückgängig zu machen, da ein kürzerer Filterausdruck nicht unbedingt verständlicher sein muss. Ein erfahrener Anwender könnte beispielsweise bewusst redundante Verzweigungen im Baum einbauen, damit die Bedeutung des Filterausdrucks auch in Zukunft nachvollziehbar bleibt.
Das Anbieten der Funktion wäre über einen zusätzlichen Button im Menübereich des Filter-Editors denkbar.

6.3.3 Mehrere Ein- und Ausgabeformate

Aktuell bietet der Filter-Editor ein vorgegebenes Ein- und Ausgabeformat, welches in Kapitel 5.3.2 beschrieben wurde. Boolesche Ausdrücke können aber an verschiedensten Orten angetroffen werden. Es wäre deshalb denkbar für den Filter-Editor weitere Ein- und Ausgabeformate anzubieten. Beispielsweise könnte der Filter-Editor genutzt werden, um Where-Klauseln für SQL-Ausdrücke zu definieren. Weitere denkbare Formate sind beispielsweise json, xml oder aber auch yaml.
Um nicht für jedes vorher genannte Format eine Funktion implementieren zu müssen, wäre es praktischer, dass an den Filter-Editor eine so genannte Callback-Funktion übergeben werden kann. In dieser Funktion könnten Nutzende einen Algorithmus definieren, um den erstellten Filterausdruck in eine gewünschte Struktur zu konvertieren. Dies könnte mit Hilfe des Strategy-Patterns implementiert werden.

6.3.4 Zusätzliche Hilfsfunktionen für die Liste der Bedingungen

Bei einem grossen Filterausdruck kann die Liste der Bedingungen sehr viele Bedingungen enthalten und entsprechend unübersichtlich werden. Um die Benutzungsfreundlichkeit auch in einer solchen Situation aufrechtzuerhalten, sollten folgende Hilfsfunktionen angeboten werden:

- In einem zusätzlichen Suchfeld oberhalb der Liste kann über eine Freitextsuche nach einer Bedingung gesucht werden.
- Es gibt einen zusätzlichen Button für die Sortierung der Bedingungen. Die Sortierung erfolgt nach Eigenschaft, Operator und dann nach dem Wert. Das Anbieten einer getrennten Sortierung wäre auch denkbar.
- Innerhalb der Liste ist es möglich die Bedingungen in Gruppen einzuteilen. Die Gruppen können von Nutzenden beliebig erstellt und benannt werden. Diese Gruppen dienen lediglich der Übersicht und haben keinen Einfluss auf die Logik bzw. die resultierenden Ausdrücke.

6.3.5 Zusätzliche Drag & Drop Optionen

Die Möglichkeit mit Drag & Drop zu arbeiten, erleichtert die Nutzung des Filter-Editors. Jedoch sind damit zurzeit noch nicht alle Manipulationen möglich und einige könnten noch verbessert werden. Zum einen ist für die Zeilenansicht aktuell noch kein Drag & Drop implementiert, was noch nachgeholt werden muss. Zum anderen fehlt bei der Baumansicht die Möglichkeit Bedingungen innerhalb der Arbeitsfläche direkt miteinander auszutauschen und es gibt noch weitere Details, die optimiert werden könnten.

Eine mögliche Implementierung in der Baumansicht könnte so aussehen: Wenn eine Bedingung innerhalb der Arbeitsfläche auf eine andere Bedingung per Drag & Drop gezogen wird, dann sollten diese beiden Bedingungen direkt getauscht werden. Wenn eine Bedingung hingegen auf eine Verknüpfung gezogen wird, dann sollte sich, falls ein Austauschen möglich wäre, neu zusätzlich ein Kontextmenü öffnen und zwei Optionen anbieten, entweder wird die Bedingung an die Verknüpfung angehängt (Option «Append»), wie dies in der aktuellen Implementierung automatisch der Fall ist, oder die beiden Elemente könnten ausgetauscht werden (Option «Switch»). Dabei gilt es zu beachten, dass im Falle eines Tausches der gesamte Teilbaum unterhalb der Zielverknüpfung ebenfalls mitverschoben werden müsste. Genauso denkbar ist der Austausch von zwei Teilbäumen beim Drag & Drop von einer Verknüpfung auf eine andere. Zu beachten ist, dass ein solcher Austausch jeweils nur möglich sein darf, wenn die beiden Elemente sich nicht auf demselben Ast im Filterausdruck befinden.

6.3.6 Kopieren & Einfügen von Teilbäumen

Teilausdrücke bzw. Teilbäume können in der Baumansicht aktuell mittels Drag & Drop umgehängt werden. Dies allein genügt für fortgeschrittenere Nutzende nicht. Da es oft vorkommt, dass sich in grossen booleschen Ausdrücken gewisse Teilausdrücke wiederholen, wäre es wünschenswert, dass Teilbäume beliebig kopiert und wieder eingefügt werden können. Eine solche Operation wäre in der Baumansicht über das Kontextmenü auf den Verknüpfungen denkbar (vergleichbar mit Copy & Paste).

6.3.7 Speichern & Wiederverwenden von Ausdrücken

Bei Anwendungen, welche boolesche Ausdrücke intensiv verwenden, werden sich die booleschen Ausdrücke mit der Zeit wiederholen. Um effizienter mit dem Filter-Editor arbeiten zu können, sollte es möglich sein, existierende Ausdrücke abzuspeichern und wiederzuverwenden. Dies kann auf zwei Ebenen implementiert werden. Zum einen könnte dies direkt innerhalb des Filter-Editors ermöglicht werden, indem auf Verknüpfungen über das Kontextmenü der Teilbaum gespeichert und anschliessend über die Toolbar wiederverwendet werden könnte. Zum anderen könnte der Filter-Editor ein Property anbieten, über welches bestehende Ausdrücke in den Filter-Editor importiert werden können.

6.3.8 Miniaturansicht

In der Besprechung des ersten Mockups ist die Idee aufgekommen, eine Miniaturansicht der Baumdarstellung in den Filter-Editor zu integrieren (vgl. Kapitel 4.3.1). Eine solche Miniaturansicht des Baums, könnte die Navigation in einem grossen Filterausdruck erleichtern.
Das Projektteam hat sich nicht genauer damit auseinandergesetzt, wie eine Implementierung der Miniaturansicht aussehen könnte. Es handelt sich aber um eine Idee, welche in einer Weiterentwicklung in Betracht gezogen werden könnte.

6.3.9 Markierung des zuletzt verschobenen Elements

Eine weitere Idee ist es, dass das zuletzt per Drag & Drop verschobene Element jeweils hervorgehoben bleibt, bis ein anderes Element gezogen wird. Dadurch sieht der Nutzende bei einer Navigation in einem grossen Baum auch später noch, welche Änderung er zuletzt vorgenommen hat.

6.3.10 Integration in GRIPS

Um die erstellte Web Component für GRIPS zu verwenden, müssen noch einige zusätzliche GRIPS spezifische Anforderungen beachtet werden.

- In GRIPS wird der Ausdruck «Condition» bereits in einem anderen Kontext verwendet und ist deshalb reserviert. Im Filter-Editor müsste der Ausdruck «Condition» entweder mit einem Tooltipp definiert werden oder es müsste ein anderer Ausdruck wie beispielsweise «Equipment Option» verwendet werden.
- Es existieren bereits Bedingungen in GRIPS. Diese müssten entweder in das vom Filter-Editor aktuell akzeptierte Format (vgl. Kapitel 5.3.2) konvertiert werden, um sie im Filter-Editor bearbeiten zu können oder der Filter-Editor müsste so angepasst werden, dass die von GRIPS vorgegebene Struktur als Ein- und als Ausgabe verwendet werden kann.
- GRIPS unterstützt nur die Operatoren gleich (=) und ungleich (!=). Eine Bedingung wie «Horse Power < 140» müsste übersetzt werden zu «Horse Power = 0», «Horse Power = 1», …, «Horse Power = 139».

- In GRIPS gibt es eine endliche Zahl von Ausrüstungsoptionen. Der Filter-Editor müsste so erweitert werden, dass für die Eigenschaft der Bedingungen nur die von GRIPS vorgegebenen Ausrüstungsoptionen ausgewählt werden können. Operatoren könnten dann nur noch in Abhängigkeit von der Eigenschaft gewählt werden.

Die oben beschriebenen Anforderungen sind gültig, wenn der Filter-Editor eingesetzt wird, um Filterausdrücke für Artikel oder Textabschnitte zu definieren. Die Usertests mit den Mitgliedern der technischen Redaktion haben jedoch ergeben, dass der Filter-Editor in GRIPS nicht nur zum Definieren von Filterausdrücken für Artikel und Textabschnitte verwendet werden kann, sondern auch an anderen Orten. Beispielsweise könnte dieser benutzt werden, um Ausrüstungsoptionen innerhalb von GRIPS zu filtern und mögliche Kombinationen für diese zu definieren. Um den Filter-Editor in dieser Art zu verwenden, müssten die Anforderungen noch erarbeitet werden.

6.3.10.1 Erweiterung für GRIPS

Das Konzept «Testing und Highlighting» (vgl. Kapitel 3.6) könnte so erweitert werden, dass ein Mitglied der technischen Redaktion direkt an der Stelle des Filterausdrucks überprüfen kann, was im Artikel publiziert wird. Es würde also ein Preview des Artikels, welcher den Filterausdruck berücksichtigt, angezeigt werden. Dies ist eine GRIPS spezifische Erweiterung, welche einen enormen Mehrwert für Nutzende bieten würde.

6.4 Alternativer Ansatz

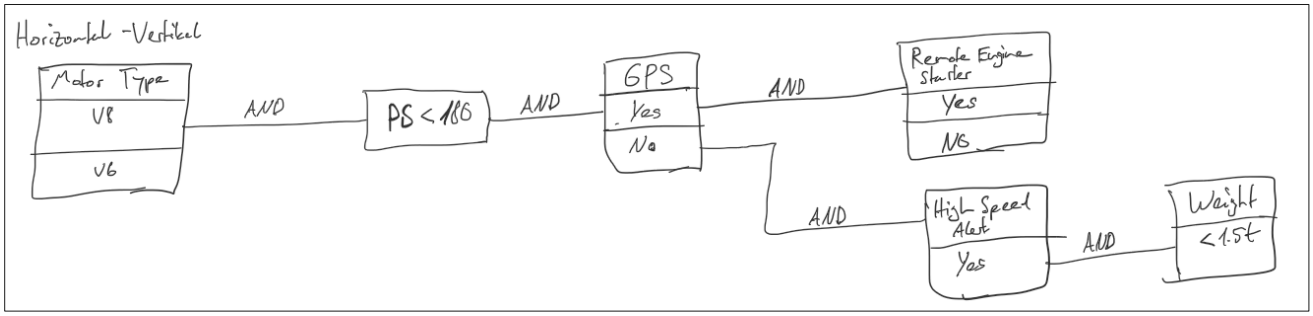


Abbildung 28: Grafischer Ansatz
Quelle: Eigene Darstellung

Eine Alternative zum im Projekt umgesetzten Prototyp wäre ein Ansatz, bei welchem der Fokus vor allem auf den grafischen Aspekt gelegt wird. Wie ein solcher Ansatz aussehen könnte, wird nachfolgend beschrieben.
Beim Grafischen Ansatz (siehe Abbildung 28) sind Ansicht und Bedienung eng miteinander verknüpft. Dabei werden die AND-Verknüpfungen horizontal und die OR-Verknüpfungen vertikal dargestellt. Dies bedeutet, dass Bedingungen als Kästchen, in denen alle erlaubten Varianten von oben nach unten dargestellt sind, erfasst werden (OR-Verknüpfungen). Mehrere Bedingungen können dann mit Linien verbunden werden (AND-Verknüpfungen). Durch diese strikte Trennung der Richtungen, in welche die Verknüpfungen gemacht werden, sind die Ausdrücke leichter zu verstehen.
Die Grundidee, welche von Shneiderman (1996, S. 341) beschrieben wurde, erlaubt nur eine ein- und ausgehende horizontale Linie pro Kästchen. Diese Einschränkung macht komplexe Filterausdrücke jedoch unmöglich. In Abbildung 28 sieht man die vom Projektteam erweiterte Version des Konzepts. Bei dieser Version sind mehrere ein- und ausgehende AND-Verknüpfungen pro Kästchen erlaubt. Dies ermöglicht die Erstellung von beliebig komplexen Ausdrücken. Sobald die Ausdrücke jedoch grösser und komplizierter werden, wird ein Filter-Editor, welcher auf diesem Konzept aufbaut, schnell unübersichtlich. Um viele AND-Verknüpfungen von einer Bedingung aus sauber darstellen zu können, müssten noch Ansätze entwickelt werden.
Das Projektteam entschied sich nach Erarbeitung der Skizze in Abbildung 28 und den erwähnten Problemen des Ansatzes dafür, den Ansatz nicht weiterzuverfolgen. Dennoch ist der Ansatz interessant und könnte als Grundlage für ein Projekt dienen.

7 Projektmanagement

7.1 Rollen und Verantwortlichkeiten

Teammitglied	Rolle	Verantwortlichkeiten
Daniel Frick	Software-Entwickler	• Dokumentation • Projektmanagement • Issue- und Time-Tracking
Zvonimir Serkinic	Software-Entwickler	• Entwicklung • Figma • Kommunikation mit Betreuer

Tabelle 9: Rollen und Verantwortlichkeiten

Anmerkung: Die Verantwortlichkeiten beschreiben nicht, wer die damit verbundenen Aufgaben ausführt. Es geht lediglich darum, welche Person in welchem Aufgabengebiet den Lead übernimmt.

7.2 Prozesse

Zur Planung und Durchführung des Projekts wird für die Grobplanung auf den RUP-Prozess gesetzt und für die Feinplanung wird Scrum verwendet.
Zu Beginn des Projekts wird ein grober Projektplan erstellt, welcher während des Projektfortschritts fortlaufend alle zwei Wochen aktualisiert und falls notwendig angepasst wird. Mehr Informationen zum Projektplan befinden sich im Kapitel 7.5.
Für die Feinplanung wird ein Agiles-Board in YouTrack verwendet, welches den Scrum-Prozess unterstützt. Das Board bietet dabei einerseits eine komfortable Möglichkeit für die Verwaltung von User Stories und Tasks und andererseits einen guten Überblick über noch ausstehende Arbeiten eines Sprints. Das Board kann hier eingesehen werden.
Als Sprintdauer wird eine Woche verwendet, da sich dies bei der Studienarbeit bewährt hat. So ermöglicht dies eine maximale Flexibilität, um auf unerwartete Probleme oder Änderungswünsche zu reagieren. Die Sprint Planung für den nächsten Sprint erfolgt jeweils direkt im Anschluss an das Meeting mit dem Betreuer.

7.3 Meetings

Es wird wöchentlich ein Meeting mit dem Betreuer abgehalten, dieses findet generell donnerstags um 08:30 Uhr statt. Die Meetings werden protokolliert und das Sitzungsprotokoll dem Betreuer im Anschluss zugestellt.

7.4 Verwendete Tools

Tool	Verwendung
Teams, WhatsApp	Teamkommunikation
YouTrack	Issue-Management und Zeiterfassung
MS Office Produkte	Dokumentation
draw.io	Skizzen und Grafiken
OneNote	Konzepte erstellen
Figma	Paper-Prototyp erstellen
JetBrains WebStorm	Entwicklung
GitLab	Code-Verwaltung, CI

Tabelle 10: Verwendete Tools

7.5 Projektplan

Der grobe Projektplan wird mit Excel erstellt. Im Anhang befinden sich sowohl der initial erstellte Projektplan als auch der fortlaufend aktualisierte Projektplan.

7.6 Meilensteine

Meilenstein	Termin	Ziele
M1: Aufgabenstellung	02.03.2023	- Aufgabenstellung klar definiert und verstanden - Betreuer und Gruppe haben das gleiche Verständnis vom Ziel der Arbeit - Das grobe Vorgehen sowie die erwarteten Resultate sind bekannt
M2: Figma Prototyp	19.04.2023	- Figma Prototyp ist bereit für Usertests
M3: Zwischenpräsentation	13.04.2023	- Zwischenresultate präsentiert
M4: Endprodukt	07.06.2023	- Prototyp entwickelt und getestet
M5: Abgabe	16.06.2023	- Dokumentation vollständig - Alle Dokumente im AVT hochgeladen

Tabelle 11: Meilensteine

7.6.1 Vorgenommene Änderungen im Projektverlauf

7.6.1.1 Anpassung Meilenstein 2

Der zweite Meilenstein hatte zu Beginn die Bezeichnung «Konzeptwahl getroffen» (vgl. initialer Projektplan im Anhang). Das Ziel war, sich bis zum 12.04.2023 für eines der erarbeiteten Konzepte zu entscheiden.
Das Projektteam hat sich dann allerdings im Projektverlauf dazu entschieden, alle Konzepte aus Kapitel 3 zu einem einzelnen Prototyp zusammenzubauen, wodurch der Meilenstein «Konzeptwahl getroffen» keinen Sinn mehr machte. Aus diesem Grund wurde der zweite Meilenstein umbenannt zu «Figma Prototyp». Aufgrund des grossen Aufwands einen guten Figma Prototyp zu erstellen, wurde zudem der Termin für den Meilenstein um eine Woche nach hinten geschoben.

7.6.1.2 Entfernung Meilenstein «Prototyp»

Im initialen Projektplan gab es einen Meilenstein «Prototyp». Das Ziel war es bis zum 03.05.2023 einen ersten lauffähigen Prototyp zu haben.
Im Projektverlauf war es dann so, dass der Figma Prototyp über vier Iterationen verbessert wurde. Durch wiederholtes Einholen und Integrieren von Feedback in den Figma Prototyp verschob sich der Start der Entwicklung nach hinten. Dadurch wurde die Entwicklung eines Prototyps bis zum 03.05.2023 unrealistisch. Aus diesem Grund wurde entschieden den Meilenstein «Prototyp» zu streichen.

7.6.1.3 Anpassung Meilenstein 4

Im Meeting vom 04.05.2023 wurde nach Absprache mit dem Betreuer entschieden, dass die Integration in den GRIPS Prototyp keine Priorität hat und aus zeitlichen Gründen nicht mehr angegangen wird. Aus diesem Grund wurde das Ziel für den Meilenstein 4 angepasst auf «Prototyp entwickelt und getestet». Das vorherige Ziel «Integration in den GRIPS-Prototyp» wurde entfernt.

7.7 Risikomanagement

Die Risikoanalyse soll eine Übersicht über mögliche Risiken des Projekts geben. Die Liste ist nicht abschliessend und kann im Verlaufe des Projekts bei Notwendigkeit angepasst, als auch um weitere Risiken ergänzt werden.

7.7.1 Identifizierte Risiken

Nr.	Beschreibung	Vorbeugung	Verhalten beim Eintreten
R1	Die Anforderungen an das gewünschte Endprodukt ändern sich.	- Wöchentliche Meetings mit Betreuer - Bewusstsein für Risiko in Planung sowie Durchführung einbeziehen	- Neu-Priorisierung der Aufgaben - Projektplan anpassen - Abstimmung mit Betreuer
R2	Keines der entwickelten Konzepte überzeugt. Usertests zeigen, dass die Benutzungsfreundlichkeit unzureichend ist.	- Iterative Verfeinerung und Verbesserung der Konzepte - Frühzeitige Usertests	- Neue Definition für Ziel der Arbeit - Meilensteine neu definieren
R3	Das gewählte Konzept für den Filter-Editor lässt sich aufgrund von technischen Einschränkungen nicht umsetzen.	- Bei Entwurf der Konzepte auf Realisierbarkeit achten - Iterative Verfeinerung und Verbesserung der Konzepte	- Alternatives Konzept wählen oder bestehendes Konzept modifizieren
R4	Integration in GRIPS-Prototyp nur schwer möglich oder dauert länger als geplant.	- Frühzeitige Auseinandersetzung mit GRIPS-Prototyp - Unterstützung durch Michael Gfeller	- Zeitbudget zur Anbindung an Prototyp neu evaluieren (Absprache mit Betreuer)
R5	Es werden Technologien und Tools eingesetzt, wo kein oder wenig Vorwissen besteht.	- Frühzeitige Festlegung - Vorzeitige Einarbeitung - Aufgabe an Person mit bester Eignung geben	- Änderung der Technologien evaluieren - Besprechung mit Betreuer
R6	Konflikte innerhalb des Teams.	- Offener und direkter Dialog - Regelmässiger Austausch	- Gemeinsam Blick auf Ziel der Arbeit richten - Besprechung mit Betreuer

R7	Daten (Dokumentation, Code) könnten verloren gehen.	- Regelmässige Backups des OneDrive-Ordners - Gesamter Code in Git-Repository auf GitLab verwalten	- Wiederherstellung aus letztem Backup

Tabelle 12: Identifizierte Risiken

7.7.2 Risikoevaluation

Die identifizierten Risiken werden alle zwei Wochen bezüglich Wahrscheinlichkeit und Schweregrad neu beurteilt. Die in den nachfolgenden Unterkapiteln verwendete grafische Risikomatrix basiert auf Thehos (2019) und wurde den Bedürfnissen entsprechend angepasst.

7.7.2.1 Initiale Risiko Bewertung vom 06.03.2023

- **R1:** Die Wahrscheinlichkeit, dass sich die Anforderungen an das Endprodukt im Verlauf des Projekts ändern werden, ist sehr gross, wie bei den meisten Softwareprojekten.
- **R2:** Es ist durchaus möglich, dass am Ende keines der Konzepte wirklich überzeugt, da das Projekt eher forschungsgetrieben ist. Die Eintrittswahrscheinlichkeit des Risikos wird deshalb im mittleren Bereich eingeschätzt, das Schadenspotenzial ist hoch.
- **R3:** Das Risiko, dass technische Einschränkungen zu Problemen bei der Umsetzung des gewählten Konzepts führen, wird als eher gering eingeschätzt.
- **R4:** Es besteht die Gefahr, dass die Integration in den bestehenden GRIPS-Prototyp nicht wie gewünscht funktioniert. Dies vor allem aufgrund der fehlenden Erfahrungswerte bezüglich GRIPS. Die Eintrittswahrscheinlichkeit wird deshalb als hoch eingeschätzt. Das Schadenpotenzial hingegen ist eher gering, weil eine Integration in den GRIPS-Prototyp gemäss Aufgabenstellung nicht zwingend erforderlich ist.
- **R5:** Das fehlende Vorwissen zu den einzusetzenden Tools betrifft vor allem Figma und eventuell die einzusetzende Technologie für die Umsetzung. Die Eintrittswahrscheinlichkeit wird als eher gering eingeschätzt und auch das Schadenpotenzial, da anstelle von Figma leicht auf Paper-Prototyping ausgewichen werden könnte und bei der Umsetzung im schlimmsten Fall auf React zurückgegriffen werden kann, mit welchem das Projektteam bereits Erfahrung hat.
- **R6:** Konflikte innerhalb des Teams erscheinen sehr unwahrscheinlich, da das Projektteam bereits die Studienarbeit zusammen absolvierte.
- **R7:** Der Verlust von Daten wäre natürlich in jeder Phase des Projekts fatal. Die Eintrittswahrscheinlichkeit ist jedoch sehr gering, da mit Backups und einem Git-Repository gearbeitet wird.

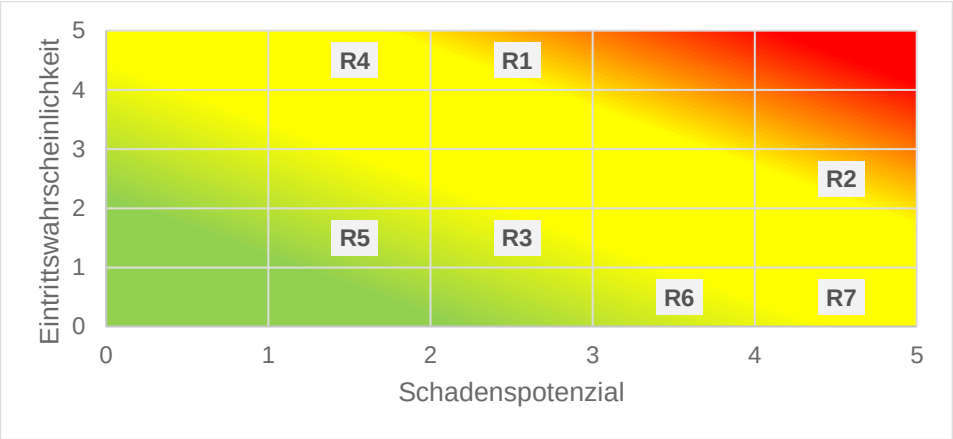


Abbildung 29: Risikomatrix vom 06.03.2023
Quelle: Eigene Darstellung

7.7.2.2 Risikobewertung vom 20.03.2023

Keine Risiken eingetreten

Anmerkungen zu den Verschiebungen:

- **R6:** Die Zusammenarbeit hat sich gut eingespielt. Das Schadenpotenzial wurde deswegen reduziert.
- **R7:** Es wurde ein Backup für den OneDrive-Ordner eingerichtet, dadurch konnte das Schadenpotenzial auf ein Minimum reduziert werden.

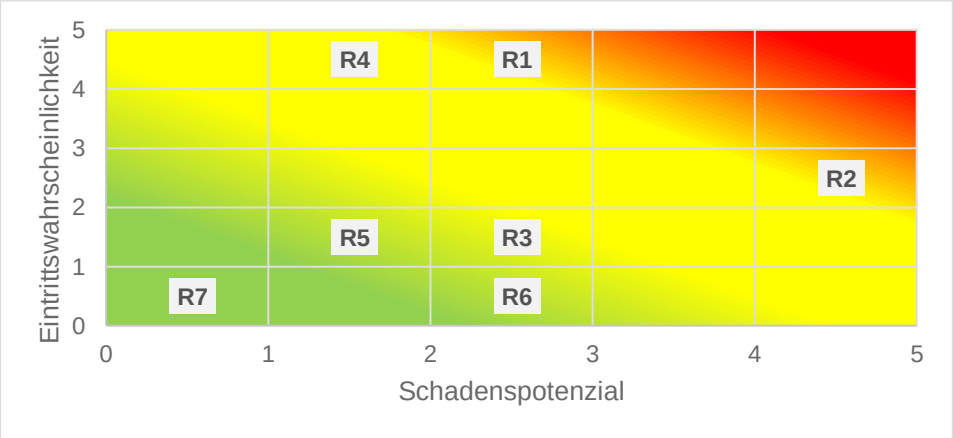


Abbildung 30: Risikomatrix vom 20.03.2023
Quelle: Eigene Darstellung

7.7.2.3 Risikobewertung vom 05.04.2023

Keine Risiken eingetreten
Keine Verschiebungen

7.7.2.4 Risikobewertung vom 17.04.2023

Keine Risiken eingetreten

Anmerkungen zu den Verschiebungen:

- **R1:** Der Figma Prototyp wurde den Fachleuten der STAR AG präsentiert und deren Feedback berücksichtigt. Die Eintrittswahrscheinlichkeit, dass sich die Anforderungen an das Endprodukt noch einmal gross verändern, hat sich dadurch reduziert. Das Gleiche gilt für das Schadenspotenzial, welches nun ebenfalls als weniger gross eingeschätzt wird.

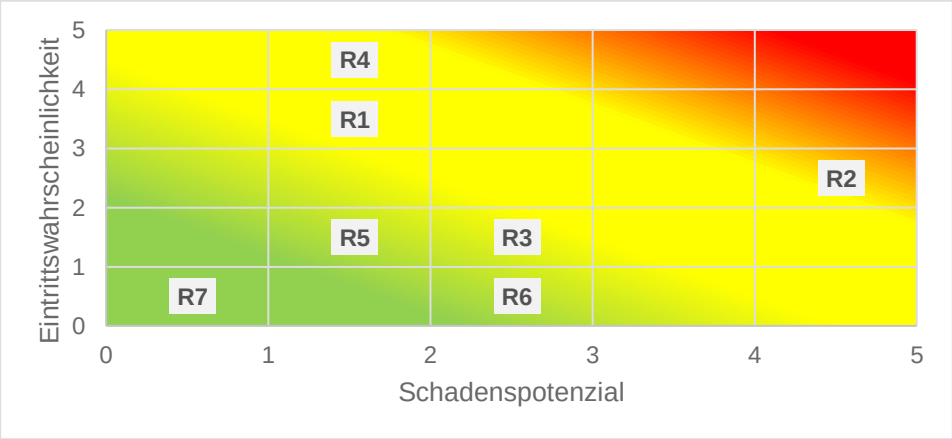


Abbildung 31: Risikomatrix vom 17.04.2023
Quelle: Eigene Darstellung

7.7.2.5 Risikobewertung vom 01.05.2023

Keine Risiken eingetreten
Keine Verschiebungen

7.7.2.6 Risikobewertung vom 15.05.2023

Eingetretene Risiken:

- **R1:** Das Styling des Filter-Editors soll sich am Corporate Design der OST orientieren.
- **R4:** Das Projektteam hat festgestellt, dass die Zeit nicht ausreichen wird für eine Integration in den GRIPS-Prototyp. Nach Absprache mit dem Betreuer wurde im Meeting am 04.05.2023 entschieden, dass die Integration keine Priorität hat und nicht umgesetzt wird. Das Risiko R4 entfällt somit für die restlichen Risikoanalysen.

Anmerkungen zu den Verschiebungen:

- **R2:** Mittlerweile wurden zwei Usertests mit Mitgliedern der technischen Redaktion der STAR AG und fünf Usertests mit weiteren Testpersonen durchgeführt. Die Usertests haben sehr unterschiedliches Feedback ergeben. Einige Personen kamen mit der geplanten Baumansicht sehr gut zurecht, andere wiederum gar nicht. Es wurde festgestellt, dass die Meinungen auseinandergehen. Basierend auf dem Feedback und dem zeitlichen Restbudget wird am jetzigen Konzept festgehalten, womit sich das Schadenpotenzial reduziert.

- **R3:** Die technische Umsetzung wurde mit Stencil.js evaluiert und es wurden erste Features teilweise implementiert. Rein vom technischen her, sollte der Filter-Editor, wie geplant, als Web Component gebaut werden können. Aus diesem Grund reduziert sich die Eintrittswahrscheinlichkeit auf ein Minimum.
- **R5:** Die Einarbeitung in Stencil.js ging problemlos über die Bühne. Aus diesem Grund kann die Eintrittswahrscheinlichkeit auf ein Minimum reduziert werden.

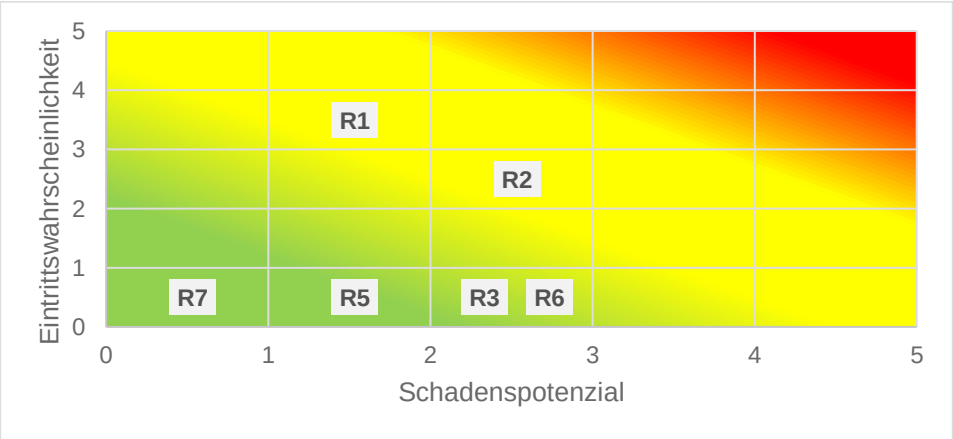


Abbildung 32: Risikomatrix vom 15.05.2023
Quelle: Eigene Darstellung

7.7.2.7 Risikobewertung vom 29.05.2023

Keine Risiken eingetreten

Anmerkungen zu den Verschiebungen:
- **R1:** In der Endphase der Arbeit werden keine Änderungen an das gewünschte Endprodukt mehr akzeptiert. Die Eintrittswahrscheinlichkeit konnte dadurch auf ein Minimum reduziert werden.
- **R2:** Das Feedback aus den Usertests hat sich mit jeder zusätzlichen Iteration verbessert. Der entwickelte Prototyp hat sich bewährt, wodurch sich die Eintrittswahrscheinlichkeit auf ein Minimum reduziert hat.

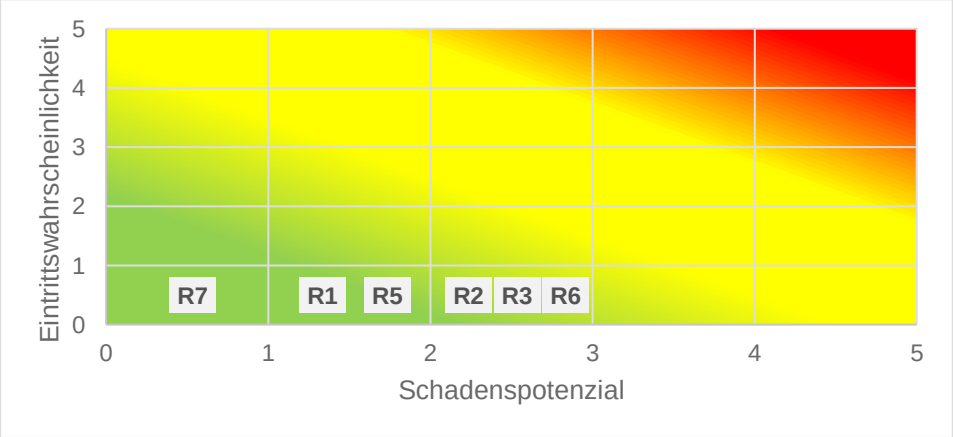


Abbildung 33: Risikomatrix vom 29.05.2023
Quelle: Eigene Darstellung

7.7.2.8 Risikobewertung vom 05.06.2023

Keine Risiken eingetreten
Keine Verschiebungen

7.7.2.9 Abschluss Risikomanagement

Alle Risiken bis auf **R4** konnten im Verlauf des Projekts erfolgreich eliminiert werden. Dabei sind im Projektverlauf nur zwei Risiken eingetreten.
Bei **R1** (Die Anforderungen an das gewünschte Endprodukt ändern sich) war das Eintreten des Risikos eher hilfreich als problematisch. Die Orientierung am Corporate Design der OST half dem Projektteam dabei, den Filter-Editor ansprechend zu gestalten.
Bei **R4** (Integration in GRIPS-Prototyp nur schwer möglich oder dauert länger als geplant) hingegen war dem Projektteam bereits zum Start des Projekts klar, dass die Eintrittswahrscheinlichkeit sehr hoch ist (vgl. Kapitel 7.7.2.1), da die zeitlichen Ressourcen für die Entwicklung im Rahmen der Bachelorarbeit beschränkt sind.
Das Projektteam hat sich im Projektverlauf dafür entschieden, einen umfangreichen Figma Prototyp zu erstellen und diesen ausführlich zu testen. Da dies zeitintensiv ist, wurde nach Absprache mit dem Betreuer entschieden, auf die Integration in GRIPS zu verzichten. Für die darauffolgende Projektphase konnte der Fokus deshalb auf die Implementierung der eigenen Konzepte als Web Component gelegt werden.

7.8 Qualitätsmassnahmen

An dieser Stelle wird auf die Qualitätsmassnahmen eingegangen, welche angewendet werden, um am Ende ein qualitatives Endprodukt abzuliefern. Als Erstes wird auf den gewählten Git-Workflow eingegangen. Danach folgt eine kurze Aufzählung der verwendeten Linters und eine Beschreibung der für das Projekt aufgebauten CI-Pipeline. Zum Schluss wird dann noch auf die verschiedenen Tests eingegangen, welche im Rahmen des Projekts durchgeführt werden.

7.8.1 Git-Workflow

Der gesamte Quellcode wird in GitLab in einem Git-Repository verwaltet. Dabei werden zwei geschützte Branches «main» und «develop» eingesetzt, wodurch kein direktes Pushen auf die Branches möglich ist. Um auf diese Branches Code zu pushen, wird mit Merge-Requests gearbeitet. Dies hat einerseits den Vorteil, dass durch die gegenseitige Kontrolle, welche in Form von Code-Reviews durchgeführt wird, die Code-Qualität gesteigert werden kann und andererseits, dass beide Teammitglieder einen guten Überblick über den Stand des Projekts haben.
Die Entwicklung neuer Funktionalität findet dabei über Feature-Branches statt. Der Workflow sieht folgendermassen aus:

1. Lokal den «develop»-Branch pullen.
2. Ausgehend vom «develop»-Branch einen neuen Feature-Branch erstellen.
    a. Als Name für den neuen Branch ist generell die Nummer des Tasks im YouTrack zu verwenden. Beispiel: *skc-100*.
3. Änderungen lokal vornehmen, um die Ziele des Tasks zu erfüllen, dabei einen oder mehrere Commits vornehmen.
    a. Die Commit-Nachrichten sollen folgendes Format haben: *skc-100: comment*
        i. Zuerst kommt die Nummer aus YouTrack für die Zuordnung des Commits zum Task
        ii. Dann folgt ein Kommentar in englischer Sprache, welcher maximal eine Zeile lang sein soll
4. Den lokalen Feature-Branch mit den Änderungen pushen und einen Merge-Request erstellen.
    a. Wenn die Person, welche das Review durchführt, mit den Änderungen zufrieden ist, kann sie den Feature-Branch direkt in den «develop»-Branch mergen.
    b. Wenn die Person, welche das Review durchführt, mit den Änderungen nicht zufrieden ist, dann teilt sie dem Erstellenden des Merge-Requests ihre Verbesserungsvorschläge mit.
    c. Dieser kann dann die Verbesserungen anbringen und erneut zur Kontrolle freigeben. Dies wird so lange iterativ durchgeführt bis die Person, welche das Review durchführt, die Änderungen akzeptiert (Schritt a.)

Der «main»-Branch ist für die produktive Umgebung angedacht. In diesen wird vom «develop»-Branch aus gemerged, sobald neue Funktionalität in einem stabilen Zustand vorliegt.

7.8.2 Linting & Formatierung

Für das Linting wird ESLint und zur einheitlichen Formatierung Prettier eingesetzt. Linting bedeutet konkret, dass der Code statisch auf Fehler analysiert wird. Es werden Dinge, wie zum Beispiel nicht verwendete Funktionen, als fehlerhaft markiert. Was alles als Fehler angesehen wird, kann durch die Entwickelnden in einer Konfigurationsdatei definiert werden. Als Grundlage für die Konfiguration werden dabei die von ESLint empfohlenen Einstellungen verwendet.
Prettier sorgt, wie bereits erwähnt, für die einheitliche Formatierung des Codes. Dabei ist im Projekt eine Funktion vorhanden, welche alle Quelldateien durchgeht und das konfigurierte Format erzwingt. Ein Beispiel hierfür wäre, dass einfache Anführungszeichen statt doppelter Anführungszeichen verwendet werden, um Strings zu definieren. Syntaktisch wären beide Symbole korrekt, jedoch ist es von Vorteil sich auf ein Standardzeichen zu einigen, um sich auf die eigentliche Arbeit fokussieren zu können.

7.8.3 Continuous Integration

Im GitLab ist eine CI-Pipeline eingerichtet, welche für Merge-Requests aktiviert ist. Dies hat zur Folge, dass bei jedem neuen Merge-Request die Pipeline durchlaufen wird. Dasselbe passiert auch, wenn neue Commits zu einem bestehenden Merge-Request hinzukommen.

Die Pipeline besteht dabei aus drei Teilschritten:
1. Es wird geprüft, ob der Code in einem für Prettier akzeptierbaren Zustand vorliegt.
2. Es wird geprüft, ob der Code keine ESLint Fehler aufweist.
3. Es wird geprüft, ob alle Tests erfolgreich durchlaufen.

Falls eine dieser Überprüfungen fehlschlägt, schlägt die Pipeline fehl. Wenn dies eintrifft, dann ist es nicht möglich den Merge-Request zu mergen, bis die Probleme im Code behoben sind. All diese Checks können auch über im Projekt vorhandene Scripts lokal bei den Entwickelnden durchgeführt werden. Dies führt dazu, dass nicht jedes Mal zuerst gepushed werden muss, um zu prüfen, ob die Pipeline erfolgreich durchlaufen wird.

7.8.4 Testkonzept

Im Rahmen des Projekts kommen verschiedene Testarten zum Einsatz. Nach Absprache mit dem Betreuer wurde im Meeting am 15.05.2023 entschieden, dass auf Unit Tests verzichtet werden kann und an ihrer Stelle besser End-to-End Tests realisiert werden. Aus diesem Grund kommen drei Testarten zum Einsatz, welche der nachfolgenden Tabelle entnommen werden können.

Testart	Wann?	Wer?	Wie?
End-To-End Tests	Beim Erstellen des Merge-Requests	GitLab CI Pipeline	Automatisch
Usertests	Nach Fertigstellung des Prototyps in Figma	Teammitglied mit Drittperson	Manuell
Usability Tests	Nach Implementierung des MVP	Teammitglied mit Drittperson	Manuell

Tabelle 13: Testarten

7.8.4.1 End-To-End Tests

Um die End-To-End Tests zu implementieren, werden die in Stencil.js verfügbaren Hilfsmittel verwendet. Die von Stencil.js zur Verfügung gestellten Werkzeuge können für zwei Arten von Tests verwendet werden. Zum einen sind das Unit Tests und zum anderen End-To-End Tests. Da der entwickelte Prototyp vor allem Logik bzw. Funktionen im Zusammenhang mit der Benutzeroberfläche bietet, wird auf Unit Testing verzichtet.
Mit End-To-End Tests kann getestet werden, wie Komponente und ihre Unterkomponenten angezeigt werden und wie sie zusammenarbeiten. Um das Schreiben solcher Tests zu ermöglichen, verwendet Stencil.js im Hintergrund die Library «Jest». «Jest» ist eine der am meist verbreiteten Libraries zum Testen von JavaScript- und TypeScript-Anwendungen. Die Library wird unter anderem von Meta entwickelt, gewartet und empfohlen.
Bei der Ausführung von End-To-End Tests emuliert Stencil.js des Weiteren mit Hilfe von «Puppeteer» ein Browser-Fenster im Hintergrund. In diesem Fenster wird effektiv die Komponente und ihre Unterkomponenten geladen und verschiedene Operationen darauf durchgeführt, worauf geprüft werden kann, ob sich die Komponenten richtig verhalten.
Eine Übersicht über alle implementierten End-To-End Tests kann im Kapitel 5.5 eingesehen werden.

7.8.4.2 Usertests

Die Usertests werden anhand des in Figma erstellten Prototyps durchgeführt. Es geht dabei darum herauszufinden, wie gut die Nutzenden mit dem geplanten Prototyp zurechtkommen. Die Usertests werden, mit Hilfe eines vorab erstellten Testprotokolls, mit Mitgliedern der technischen Redaktion der STAR AG durchgeführt als auch weiteren Testpersonen, um ein möglichst breit abgestütztes Feedback zu erhalten.
Auf Grund der grossen geografischen Distanz zur STAR AG werden die Usertests mit den Mitgliedern deren technischen Redaktion via Teams durchgeführt. Die Testpersonen loggen sich dabei jeweils bei Figma ein und teilen ihren Bildschirm mit dem Moderator des Usertests. Somit kann sichergestellt werden, dass es keine verfälschten Testergebnisse gibt. Die Usertests mit den weiteren Testpersonen werden persönlich durchgeführt. Das Feedback der Nutzenden wird jeweils während des Tests vom Moderator notiert.
Die Resultate der Usertests sind dabei direkt in Kapitel 4 eingeflossen. Des Weiteren können alle ausgefüllten Testprotokolle zu den Usertests im Anhang eingesehen werden.

7.8.4.3 Usability Tests

Die Usability Tests werden durchgeführt, sobald ein MVP des Filter-Editors implementiert werden konnte. Es wird wiederum im Voraus ein Testprotokoll erstellt und das Feedback der Testpersonen wird erneut durch den Moderator des Usability Tests notiert. Die Usability Tests werden persönlich mit den Testpersonen durchgeführt.
Die ausgefüllten Testprotokolle zu den Usability Tests können im Anhang eingesehen werden. Eine Auswertung der Usability Tests ist in Kapitel 5.6 zu finden.

7.9 Zeiterfassung

Die Erfassung der geleisteten Arbeitsstunden erfolgt im YouTrack auf den einzelnen Tasks. Es wird dabei auf eine Genauigkeit von 15 Minuten rapportiert. Abbildung 34 zeigt die geleistete Arbeitszeit der beiden Teammitglieder pro Woche.

		Feb.		März				Apr.				Mai					Juni	
Benutzer		20-26	27-05	06-12	13-19	20-26	27-02	03-09	10-16	17-23	24-30	01-07	08-14	15-21	22-28	29-04	05-11	12-16
Gesamtzeit	**701h 45m**	**23h 45m**	**34h 30m**	**31h 00m**	**33h 15m**	**40h 45m**	**23h 00m**	**48h 30m**	**44h 45m**	**38h 00m**	**42h 30m**	**43h 00m**	**38h 15m**	**47h 15m**	**46h 00m**	**53h 45m**	**57h 45m**	**55h 45m**
Daniel Frick	354h 45m	16h 00m	18h 00m	13h 45m	17h 30m	20h 30m	20h 30m	21h 00m	20h 45m	19h 30m	23h 45m	22h 45m	18h 45m	24h 00m	23h 00m	22h 30m	27h 30m	25h 00m
Zvonimir Serkinic	347h 00m	7h 45m	16h 30m	17h 15m	15h 45m	20h 15m	2h 30m	27h 30m	24h 00m	18h 30m	18h 45m	20h 15m	19h 30m	23h 15m	23h 00m	31h 15m	30h 15m	30h 45m

Abbildung 34: Geleistete Arbeitszeit pro Woche
Quelle: Eigene Darstellung

Folgende Auswertungen stehen in YouTrack als Berichte zur Verfügung:

- Zeit pro Woche Gesamt (Eine Gesamtübersicht über die Arbeitszeit jedes einzelnen Teammitglieds (vgl. Abbildung 34))
- Zeit pro Woche nach Arbeitstyp (Detaillierte Übersicht über die Arbeitszeit gruppiert nach Arbeitstyp)
- Sprint X (Auswertung pro Sprint, hier sieht man an welchen Tickets ein Teammitglied während eines Sprints gearbeitet hat und welcher Zeitaufwand dafür aufgewendet wurde)

Die Berichte sind hier zugänglich.

Glossar

Angular ist ein TypeScript-basiertes Webapplikationsframework für das Erstellen von Mobile- und Web-Applikationen.

Branch ist eine Version des Git-Repository.

CI-Pipeline ist eine Abfolge von Schritten, welche nacheinander ausgeführt werden müssen, um eine neue Version der Software zu liefern.

Code-Review ist eine Qualitätsmassnahme, welche das Überprüfen der Änderungen in einem Merge-Request durch eine andere Person beschreibt.

Commit ist eine Operation, welche die letzten Änderungen am Quellcode an das Git-Repository sendet.

DevExpress ist ein amerikanisches Softwareunternehmen, welches UI-Controls für verschiedene Frameworks anbietet.

Drag & Drop ist ein Vorgang, welcher das Ziehen und Ablegen von einem Element beschreibt.

draw.io ist ein Webtool zur Erstellung von Diagrammen.

End-To-End Test ist ein Test, der ein reales Szenario vom Start bis zum Ende simuliert.

ESLint ist ein Werkzeug zur statischen Quellcode-Analyse.

Figma ist ein Webtool, welches es mehreren Personen erlaubt, gemeinsam Benutzeroberflächen zu entwickeln.

Git ist ein verteiltes Versionskontrollsystem zur Verwaltung von Quellcode.

Git-Repository enthält eine Sammlung von Dateien mit verschiedenen Versionen eines Projekts.

GitLab ist eine webbasierte DevOps-Plattform, die neben der Verwaltung von Git-Repositories weitere Funktionalitäten wie CI-Pipelines anbietet.

GRIPS ist ein strukturiertes Informationsmanagementsystem, welches es ermöglicht grosse Mengen an Informationen schnell und wirtschaftlich zu erstellen und zu veröffentlichen.

HTML-Template ist ein Mechanismus, um HTML zu halten, welches nicht sofort gerendert wird, sondern erst nachträglich zur Laufzeit über JavaScript.

Hybrids ist ein Framework mit dem Client-Side Web-Applikationen, UI-Komponenten Libraries oder einzelne Web Components entwickelt werden können.

JavaScript ist eine Programmiersprache für das Web.

Jest ist ein JavaScript Testing Framework.

JetBrains WebStorm ist eine Entwicklungsumgebung für JavaScript und verwandte Technologien.

Lit ist ein Framework, welches es erlaubt Web Components zu entwickeln.

Merge-Request ist eine Anfrage, ob ein Branch gemerged werden darf.

Meta ist ein amerikanisches Technologie-Konglomerat.

Mockup ist ein digital gestalteter Entwurf einer Applikation.

OneDrive ist ein Cloud-Speicher für Dateien.

OneNote ist ein digitales Notizbuch.

Paper-Prototype ist eine Sammlung von handgezeichneten Skizzen, welche Konzepte für eine Applikation enthalten.

Polymer ist eine Library mit welcher Web-Applikationen unter Verwendung von Web Components entwickelt werden können.

Prettier ist ein Tool zur Codeformatierung.

Puppeteer ist eine Library, welche eine High-Level API zur Kontrolle von Chrome / Chromium, über das DevTools Protokoll, bietet.

React ist ein JavaScript Framework für das Erstellen von User Interfaces.

Rendering ist das Aufbereiten einer grafischen Oberfläche.

Salesforce ist ein amerikanisches Softwareunternehmen.

Scrum ist ein Vorgehensmodell für agile Softwareentwicklung.

Shadow DOM ist ein abgekapselter Teil im regulären DOM-Baum.

Skate.js ist eine Abstraktion der Web Component Standards in Form von Paketen, welche es erlauben Web Components zu entwickeln.

Slim.js ist ein schnelles und robustes Micro-Framework, welches es ermöglicht Web Components zu entwickeln.

STAR AG ist der Industriepartner dieser Arbeit.

Stencil.js ist eine Library mit der wiederverwendbare und skalierbare Web Component Libraries entwickelt werden können.

Strategy-Pattern ist ein Pattern, bei welchem eine Familie austauschbarer Algorithmen definiert wird.

Teams ist eine Kollaborations-Software.

TypeScript ist eine stark typisierte Programmiersprache, die auf JavaScript aufbaut.

Usability Test ist ein Test, um die Benutzungsfreundlichkeit einer Applikation mit potenziellen Nutzenden zu testen.

Usertest ist ein Test, bei dem eine Website, ein Dienst, ein Produkt oder ein Prototyp von realen Personen getestet wird.

Visitor-Pattern ist ein Pattern, bei welchem Operationen auf Elementen einer Objektstruktur ausgeführt werden können.

Vue.js ist ein JavaScript Framework für das Erstellen von User Interfaces.

Web Component ist eine Sammlung von Webstandards, welche es im Verbund erlauben, neue eigene HTML-Tags zu definieren.

WhatsApp ist ein Instant-Messaging-Dienst.

Wireframe ist eine grobe Visualisierung, welche typischerweise verwendet wird, um die Struktur und die Funktionalitäten einer Applikation zu planen.

YouTrack ist ein Issue Tracking System, welches unter anderem die Arbeit mit agilen Boards ermöglicht.

Abkürzungsverzeichnis

API	Application Programming Interface
AVT	Arbeitsverwaltungstool
CI	Continuous Integration
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
JSX	JavaScript XML
NPM	Node Package Manager
MVP	Minimum Viable Product
RABBIT	Research users, Assess the situation, Balance needs, Build an operative Image, Test the product
RUP	Rational Unified Process
SEO	Search Engine Optimization
SQL	Structured Query Language
TS	TypeScript
UCD	User-Centered Design
UI	User Interface
XML	Extensible Markup Language
YAML	YAML Ain’t Markup Language

Literaturverzeichnis

Borodanov, Sergey, 2021, Overview of the Web components in 2021 von der Webseite, https://exyte.com/blog/web-components-technology, 27.05.2023.

Bosworth, Edward Lewis, 2010, Logical Adjacency von der Webseite, http://www.edwardbosworth.com/My5155_Slides/Chapter04/KarnaughMaps.htm, 14.06.2023.

Foo, 2022, Will Web Components Replace React? von der Webseite, https://www.foo.software/posts/will-web-components-replace-react, 27.05.2023.

Garrett, Jesse James, 2011, *The Elements of user Experience: User-centered Design for the Web and Beyond* (New Riders, Berkeley).

Knight, Westley, 2019, *UX for Developers: How to Integrate User-Centered Design Principles Into Your Day-to-Day Development Work* (Apress Media LLC, Northampton).

Lowdermilk, Travis, 2013, *User-Centered Design* (O'Reilly Media, Sebastopol).

Nielsen, Jakob, 1993, *Usability Engineering* (Morgan Kaufmann, San Francisco).

Norman, Donald und Stephen Draper, 1986, *User Centered System Design: New Perspectives on Human-Computer Interaction* (Larence Erlbaum Associates, Hillsdale).

Ramirez, Carolina, 2022, Web Components. What are they? Pros, Cons and more von der Webseite, https://medium.com/geekculture/web-components-what-are-they-pros-cons-and-more-77ad56711e49, 27.05.2023.

Shneiderman, Ben, 1996, The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations, IEEE 0-8186-7469-5/96, 336-343.

Shneiderman, Ben, Catherine Plaisant, Maxine Cohen, Steven Jabobs und Niklas Elmqvist, 2018, *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (Pearson Education Limited, Harlow).

Still, Brian und Kate Crane, 2017, *Fundamentals of User-Centered Design: A Practical Approach* (Tylor & Francis Group, Boca Raton).

Tietje, Olaf, 2019, Diskrete Mathematik Herbstsemester 2019, OST.

Thehos, Andreas, 2019, Risikomatrix in Excel von der Webseite, https://thehosblog.com/2019/02/22/risikomatrix-in-excel/, 28.02.2023.

Webcomponents.org, 2023, Libraries von der Webseite, https://www.webcomponents.org/libraries, 27.05.2023.

Weinschenk, Susan, 2020, *100 Things Every Designer Needs to Know About People* (Peachpit Press, Hoboken).

Abbildungsverzeichnis

Tabellenverzeichnis

Anhangsverzeichnis

Der Anhang wurde aufgrund der grossen Anzahl an zusätzlichen Dokumenten in Ordner gegliedert und die Dokumente separat darin abgelegt. Die Ordnerstruktur für die Anhänge sieht wie folgt aus:

A_Aufgabenstellung
Dieser Ordner enthält die initiale Aufgabenbeschreibung des Betreuers.

B_Projektplan
Dieser Ordner enthält den initialen sowie den finalen (ausgefüllten) Projektplan.

C_Prototyp_Figma
Dieser Ordner enthält die einzelnen Figma Prototypen, welche im Projektverlauf entwickelt wurden.

D_Prototyp_Testprotokolle
Dieser Ordner enthält die Testprotokolle der durchgeführten Usertests.

E_UsabilityTest_Testprotokolle
Dieser Ordner enthält die Testprotokolle der durchgeführten Usability Tests.

F_Sitzungsprotokolle
Dieser Ordner enthält die Sitzungsprotokolle von den Meetings mit dem Betreuer.

G_Persoenliche_Berichte
Dieser Ordner enthält die persönlichen Berichte der Teammitglieder zur Bachelorarbeit.

H_Zitierstandard
Dieser Ordner enthält den verwendeten Zitierstandard.