

Generalisierte Auswahlkomponente für das
Web UI Toolkit "Kolibri"

Bachelor Thesis

Ramona Marti & Lea Burki

August 2024

Abstract

Die vorliegende Dokumentation beschreibt die Entwicklung einer plattformunabhängigen Dropdown-Komponente, die konsistent und benutzerfreundlich gestaltet ist. Die Hauptmotivation für diese Entwicklung ist, eine Komponente zu schaffen, die sowohl ästhetisch ansprechend als auch funktional ist. Sie lässt sich über verschiedene Webserver hinweg einheitlich nutzen. Die Komponente ist unabhängig von externen Bibliotheken entwickelt. Dazu basiert sie auf dem Kolibri-Designsystem, das als grundlegende Designrichtlinie dient.

Wöchentlichen Meetings und die intensive Zusammenarbeit mit den Betreuern sind Bestandteile der gewählten, agilen Entwicklungsmethode. Diese Vorgehensweise überprüft den Fortschritt und plant das weitere Vorgehen. Die Testergebnisse zeigen, dass der Kolibri-Codestil eine kurze Einarbeitungszeit erfordert. Danach ist jedoch eine einfache Integration in Projekte als auch Anpassung der Komponente möglich.

Ein besonderes Merkmal dieser Komponente ist die Möglichkeit, mittels mehrspaltiger Filterung eine gezielte Auswahl zu treffen. Die Vorauswahl eines Jahrzehnts ermöglicht die einfache Auswahl eines Geburtsjahrs. Die Reduktion der Jahreszahlen minimiert das Scrollen und Suchen. Dieses Prinzip lässt sich analog für andere Anwendungsfälle – wie die Auswahl von Kontinenten & Ländern oder spezifischen Mahlzeiten (z.B. vegetarische Gerichte) – anwenden.

Für die Zukunft bietet die Komponente eine flexible Basis, die leicht zu erweitern ist. Zudem bestehen Potenziale für die Entwicklung ähnlicher Komponenten für weitere Anwendungsbereiche.

Ehrlichkeitserklärung

Hiermit erklären wir, Ramona Marti und Lea Burki, dass wir unsere Bachelorarbeit mit dem Titel **24FS_IMVS17: Generalisierte Auswahlkomponente für das Web UI Toolkit "Kolibri"** selbstständig und ohne unerlaubte Hilfe angefertigt haben. Wir haben keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und alle wörtlich oder sinngemäss aus anderen Werken übernommenen Aussagen als solche gekennzeichnet. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum: _____

Unterschrift: _____

Unterschrift: _____

Danksagung

Unser erster Dank geht an Prof. Dierk König und Fabian Affolter für die wertvollen Tipps und durchgehende Unterstützung während der Bachelorarbeit **24FS_ IMVS17: Generalisierte Auswahlkomponente für das Web UI Toolkit "Kolibri"**. Ihr Fachwissen und ihre Anleitungen haben entschieden zu unserer Lösung und dem Erfolg beigetragen.

Zudem danken wir allen StudentInnen und FreundInnen, die sich als Test-Personen zu Verfügung gestellt haben. Sie haben uns mit konstruktiver Kritik und hilfreichen Anregungen Fehler und mögliche Erweiterungen aufgezeigt. Das Feedback hat uns inspiriert und Diskussionen mit unseren KommilitonInnen hat uns auf neue Ideen gebracht. Ohne sie hätte diese Arbeit nicht in dieser Form entstehen können.

Weiter sprechen wir unseren Dank an unseren Familien und FreundInnen für ihre ständige Unterstützung und ihr Verständnis aus. Sie haben uns während des Studiums und der Fertigstellung dieser Arbeit motiviert unser Bestes zu geben. Ihre Geduld und ihre Ermutigungen haben uns geholfen die Konzentration zu halten und Kraft zu tanken. Sie haben während der ganzen Zeit ein offenes Ohr geboten und durch das Korrekturlesen zum Erfolg der Arbeit beigetragen.

Zum Abschluss gilt unser Dank noch allen Personen, die uns im ganzen Studium als auch bei der Bachelorarbeit unterstützt haben. Ihre wertvollen Inputs, Hilfen und Motivationen beeinflussten den erfolgreichen Abschluss der Arbeit. Wir sind dankbar für diese Erfahrungen und all die Unterstützung und Anleitungen, die wir erhalten haben.

Inhaltsverzeichnis

Ehrlichkeitserklärung	II
Danksagung	III
1 Einleitung	1
1.1 Problemstellung	1
1.2 Ziel	1
1.3 Out of Scope	2
1.4 Leitfaden	2
2 Hintergrund	4
2.1 Ausgangslage	4
2.2 Master-Detail-Ansicht	4
2.3 Zustände in einer Auswahlkomponente	5
2.4 Browser & HTML Renderer	6
2.4.1 Rendering Prozess	6
2.4.2 Bekannte Implementationen	7
3 Existierende Auswahlkomponenten	9
3.1 HTML Datalist vs. Select	10
3.1.1 Option	10
3.1.2 Select	10
3.1.3 Datalist	12
3.2 Browser-Inkonsistenzen	15
3.2.1 UI Unterschiede	15
3.2.2 Edge Browser	17
3.2.3 Chrome Browser	18
3.2.4 Firefox Browser	19
3.2.5 Safari Browser	20
3.2.6 Browser auf Android & iOS	21
3.2.7 Undo / Redo	22
3.2.8 Fazit	24
3.3 Länderauswahl Komponente	24
3.3.1 Design	25
3.3.2 Funktionen & Interaktionen	25
3.3.3 Anwendung	25
3.4 Anwendungsfälle	27

4 Neue Auswahlkomponenten	28
4.1 Design	28
4.1.1 Designansatz	28
4.1.2 Mögliche Designoptionen eines Elements & deren Wahl	28
4.1.3 Figma-Prototypen	30
4.1.4 Farbpalette und Kontrast	31
4.1.5 Layout und Typografie	31
4.1.6 Implementation der Zustände	32
4.1.7 Optionen und Scrollbar-Styling	34
4.1.8 Prototyping und Benutzerfeedback	35
4.1.9 Implementationsresultat	35
4.2 Interaktionen	36
4.3 Prinzipien & Regeln	37
4.4 Patterns	38
4.4.1 Null Object Pattern	38
4.4.2 Projector Pattern	39
4.4.3 Decorator Pattern	40
4.5 Dropdown-Container	41
4.6 Performance	42
4.6.1 Performance Vergleich	43
4.7 Testing	44
4.7.1 Manuelle Tests	44
4.7.2 Automatisierte Tests	46
4.7.3 User Tests	47
4.8 Fazit	48
5 Diskussion	49
5.1 Future Features	50
5.1.1 Weitergehende User-Tests und Nutzerbefragungen	50
5.1.2 Weitere UI-Projektoren	50
5.1.3 Weitere Interaktions-Projektoren	50
5.1.4 Spezifische Auswahlkomponenten und Erweiterungen	51
5.1.5 Performance verbessern und mögliche Datenmenge erhöhen	51
5.1.6 Accessability verbessern	52
5.1.7 Abschliessende Bemerkungen	52
6 Glossar	53
A Aufgabenstellung	62
B Neue Komponente – Bilder	64
C Existierende Komponenten – Bilder	67
D User Test für Programmierer	78
E Personas	88
F Links	91

Kapitel 1

Einleitung

1.1 Problemstellung

Im Web existieren diverse Möglichkeiten zur Erstellung eines Auswahl-Inputs mit vorgegebenen Werten. Die HTML-Elemente bieten über die verschiedenen Browser keine konsistente Darstellung und Interaktion an. Dazu sind sie nicht schön anzusehen und nur geringfügig anpassbar. Die Werte sind begrenzt auf Text und Unicode-Symbole. Die Komponente ist nicht effizient bedienbar – besonders bei einer grossen Menge an Werten.

Als Alternative zu den Basis-Elementen existieren unzählige Bibliotheken, welche solche Eingabemöglichkeiten unterstützen. Diese besitzen externe Abhängigkeiten, welche das eigene Projekt unnötig aufblasen. Zudem benötigen viele dieser Libraries eine längere Einarbeitungszeit, um die Funktionalitäten verstehen und anwenden zu können.

Das Vorgängerprojekt deckt die Probleme ab, ist aber nur auf die Auswahl eines Landes zugeschnitten. Eine Anwendung dieser Komponente mit anderen Inhalten kann zu unerwünschtem Verhalten führen. Die oben genannten Probleme dienen als Basis für das folgende Kapitel.

1.2 Ziel

Die neue Komponente zielt ab, die Auswahl eines Wertes von einer begrenzten, vorgegebenen Menge zu ermöglichen. Dazu findet die Generalisierung und der Ausbau der vorangegangenen Länderauswahl statt. Die Eingabe soll weiterhin effizient bleiben und sich konsistent über alle Browser zeigen. Dabei liegt der Fokus auf Browser von Desktop-Computern bzw. Laptops. Die Komponente soll sich einfach anpassen lassen. Als Werte sind nebst Texten auch Bilder wünschenswert. Die Qualität ist auf dem Kolibri-Standard zu halten und durch Tests zu beweisen. User-Tests mit Programmierern als auch Endanwendern beweisen die gute Usability. Automatisierte Komponententests stellen die Korrektheit der Implementation sicher. Bei der Umsetzung sollen die Design Patterns des Kolibri ihre Anwendung finden. Dabei soll das Design und die Interaktion mit dem Toolkit synchronisiert sein. Hierbei ist zu beachten, die Ziele nicht ausserhalb des Projekts zu definieren.



1.3 Out of Scope

Dieses Projekt bezieht sich auf die Entwicklung einer Auswahlkomponente für Nutzer ohne Seh Einschränkungen. Daher spielt die Accessibility nur eine begrenzte Rolle. Screen-Reader sind nicht zu beachten, da dies zu umfangreich für diese Arbeit ist. Die effiziente Anzeige von über grossen Datenmengen mit mehr als 10'000 Werten ist nicht verlangt. Die Eingabe eines eigenen Wertes in die neue Auswahlkomponente ist ebenfalls ausserhalb der Anforderungen. Für die generalisierte Komponente reicht es, wenn die Auswahl eines einzelnen Wertes möglich ist. Die Auswahl mehrerer Werte im selben Eingabe-Element ist nicht gefordert. Die Komponente ist nicht speziell auf Mobile-Geräte auszurichten. Eine Undo- als auch eine Redo-Funktion der ausgewählten Werte ist ausserhalb des geforderten Rahmens.

Ein Bestandteil dieser Arbeit ist das Erweitern des `SimpleInputs` um ein Select und eine Datalist. Damit der Fokus des Projekts auf der generalisierten Auswahlkomponente bleibt, sind keine Änderungen ausserhalb der oben genannten Ziele gefordert. Anpassungen der Kern-Codebasis gehören nicht in den Rahmen dieses Projekts. Die Eingrenzung der Anforderungen stellt sicher, dass die Ressourcen auf die Ziele fokussiert bleiben.

1.4 Leitfaden

Dieser Bericht gliedert sich in die Teile **Hintergrund**, **existierende** und **neue Komponenten** sowie **Diskussion**. Jedes Kapitel baut auf dem vorherigen auf und führt den Leser Schritt für Schritt durch die Entwicklung und Optimierung der neuen Auswahlkomponente.

Im Kapitel **Hintergrund (2)** ist die **Ausgangslage (2.1)** des Kolibri-Toolkits und des Projekts erläutert. Es folgt eine Erklärung der **Master-Detail-View (2.2)** und der verschiedenen **Zustände (2.3)**, die eine Auswahlkomponente besitzen kann. Eine Beschreibung **der Browser und deren Rendering-Engines (2.4)** ist ebenfalls enthalten. Der **Rendering-Prozess** einer HTML-Seite sowie die wichtigsten **Browser-Implementationen** sind ebenfalls auf dem Plan.

Das Kapitel **existierende Komponenten (3)** vergleicht die **HTML-Elemente Datalist und Select (3.1)** und beschreibt die Nutzung und Unterschiede der verschiedenen Elemente. Dabei hebt es die **Browser-Inkonsistenzen (3.2)** hervor, die zu **unterschiedlichen UI-Erfahrungen** führen können. Die **Länderauswahl Komponente (3.3)** aus Projekt 5 ist nur auf die Auswahl eines Landes zugeschnitten. Mögliche **Anwendungsfälle (3.4)** der existierenden Komponenten zeigen die dabei entstehenden Probleme auf.

Eine detaillierte Beschreibung der **neuen Komponente** findet sich im gleichnamigen Kapitel 4. Das **Design (4.1)** basiert auf dem Kolibri-Designsystem und erhält Unterstützung durch **Figma-Prototypen** zur Visualisierung und zum Testen. Die **Implementation der Zustände** optimiert die Benutzerführung für Maus- und Tastaturbenutzer. **Prototyping und Benutzerfeedback** tragen in einem iterativen Prozess kontinuierlich zur Verbesserung bei. Das **Implementationsresultat** visualisiert und beschreibt die neue Komponente in diversen Beispielen.

Im Abschnitt **Interaktionen (4.2)** sind Regeln für die Maus- und Tastaturinteraktion festgelegt. Die Einhaltung diverser **Prinzipien und Regeln (4.3)** sorgt für stabilen und verständlichen Code. Der Einsatz von **Patterns (4.4)** wie **Null-Object**, **Projector** und **Decorator** strukturiert die Implementation und erhöht die Wartbarkeit des Codes. Der **Dropdown-Container (4.5)** lässt sich auf verhschiedene Weise umsetzen.

Das Kapitel **Performance** (4.6) beschreibt Optimierungen zur Verbesserung der Laufzeit sowie Leistungspotenzial der neuen Auswahlkomponente. Das **Testing**-Kapitel (4.7) dokumentiert die Durchführung sowie die Auswertung von **manueller, automatisierter Tests** und **Usability**-Tests. Schliesslich fasst das **Fazit** (4.8) die wichtigsten Erkenntnisse zusammen.

Der letzte Teil – die **Diskussion** (5) – bietet einen Überblick über die Bedeutung der Erkenntnisse für die Entwicklung. Zudem beschreiben die **Future Features** (5.1) Ideen für eine Weiterentwicklung als auch Verbesserungsvorschläge.

Kapitel 2

Hintergrund

Die Grundlagen stellen ein gemeinsames Verständnis zur Auswahlkomponente her. Dazu zählen die Zustände, welche in diesem Zusammenhang auftreten können. Außerdem behandelt es eine Basis zu den Themen Browser und Rendering. Dieses Verständnis ermöglicht es, eine neue, konsistente Komponente zu entwickeln.

2.1 Ausgangslage

Studenten als auch Mitarbeiter entwickeln das Toolkit Kolibri laufend weiter. Entwickler können die Open Source Werkzeuge ganz einfach importieren und verwenden. Damit das Kolibri-Toolkit schlank bleibt, kommen keine externen Abhängigkeiten zur Verwendung. Mit einer VanillaJS Codebasis bietet das Tool eine breite Auswahl an, deckt aber noch nicht alle Interaktionen ab.

In diesem Projekt dient der in der Vorarbeit erstellte Fork als Ausgangslage. Ein Merge – des Branches *experimental* dieses Forks und des *Kolibri-16* der originalen Codebasis – stellt die Aktualität sicher. Das `SimpleInput` ist eines der Tools, welches sich bei der Implementation der neuen Auswahlkomponente als hilfreich erweisen kann.

Der Zugriff auf das Tool Figma ermöglicht das Verwenden des existierenden Designsystems und der bereits eingebundenen Elemente. Der Aufbau der Elemente als Komponenten vereinfacht die Wiederverwendung. Das Ergänzen des Icon-Sets ist bei Bedarf erlaubt.

Eine weitere, wichtige Ausgangslage ist ein gemeinsames Verständnis der verwendeten Begriffe. Dies sicherzustellen ist die Aufgabe des nächsten Kapitels mit der Beschreibung der Zustände einer Auswahlkomponente als auch derer Elemente. Diese Definitionen gelten im gesamten Bericht.

2.2 Master-Detail-Ansicht

Eine Auswahlkomponente lässt sich unkonventionell in eine Master-Detail-Ansicht einteilen. Dies ist in Abbildung 2.1 ersichtlich.

Typisch für die Master-Detail-View ist, dass die Detail-Ansicht mehr Daten anzeigt als der Master. Bei der Anwendung des Patterns auf eine Auswahlkomponente ist dies nicht der Fall. Anders als bei der typischen Master-Detail-View beinhaltet die Detail-Ansicht keine weiteren Informationen. Der Master listet alle möglichen Optionen auf. Dieser View-Baustein findet

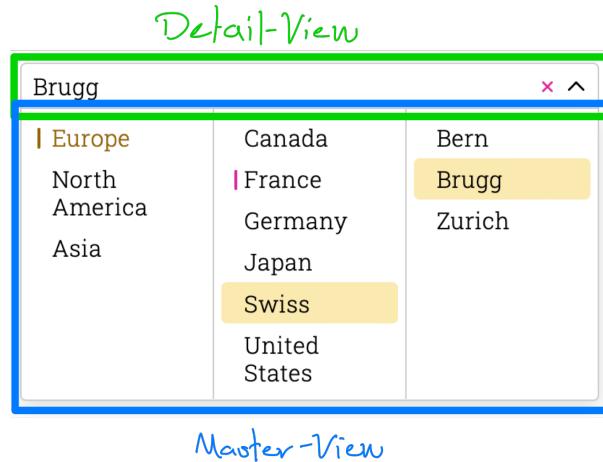


Abbildung 2.1: Master-Detail Aufteilung der Auswahlkomponente

sich im aufklappenden, scrollbaren Container wieder. Die dauerhaft sichtbare Komponente stellt die Detail-View dar. Dieser Container beinhaltet den aktuell selektierten Wert als auch das Dropdown-Icon. Weiteres zu den möglichen Zuständen ist im nachfolgenden Kapitel zu finden.

2.3 Zustände in einer Auswahlkomponente

Dieser Abschnitt erklärt die Zustände, welche in einer Auswahlkomponente auftreten können. Als Ausgangslage dient eine Eingabemöglichkeit, die eine Master-Detail-Ansicht aufweist. Zudem sind keine speziellen Voreinstellungen getroffen.



Abbildung 2.2: Geschlossene Komponente

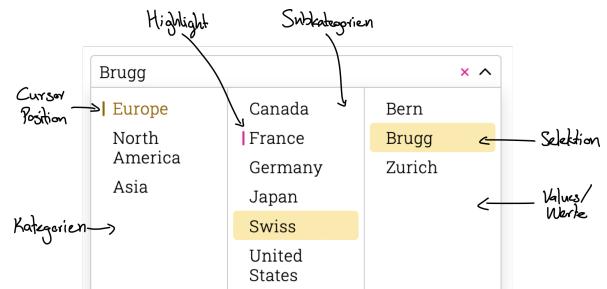


Abbildung 2.3: Offene Komponente

Je nach Darstellung der Komponente kann diese *offen* oder *geschlossen* sein. Im geschlossenen Status (Abbildung 2.2) zeigt das Erscheinungsbild nur die Detail-Ansicht an, welcher mindestens den selektierten Wert anzeigt. Eine offene Auswahlkomponente wie in Abbildung 2.3 stellt beide Elemente der Master-Detail-View dar. Die Master-Ansicht zeigt alle Optionen in einer Liste an.

Bei dem *normalen* bzw. nicht fokussierten Status ist die Komponente nicht an- oder ausgewählt. Wenn eine Webseite diese Komponente enthält, ist dies die standardmässige Darstellung. Das neue Element zeigt keine Reaktion auf Interaktionen, welche in diesem Zustand geschehen. Als einzige Ausnahme gilt Tab, welche den Fokus auf die Komponen-

te legen kann. In den meisten Erscheinungen ist nur die Detail-Ansicht sichtbar und der Master-Container ist ausgeblendet. Wählt der Nutzer die Komponente mit der Maus oder der Tastatur an, steht sie im *Fokus* bzw. ist sie *fokussiert*. Bedienungen über das Keyboard beziehen sich hierbei auf den Baustein. In den meisten Fällen ändert sich die Darstellung des Eingabefeldes z.B. durch einen blauen Rahmen.

Ist ein Wert in der Master-Ansicht ausgewählt und erscheint in der Detail-View, ist dieser *selektiert*. Das Formular enthält beim Versenden das Value der *Selektion*. Eine *Selektion* in einer Kategorie-Spalte findet in den Formulardaten keine Berücksichtigung. Stattdessen filtert die Kategorie-Selektion die Wert-Spalte. Durch das Hervorheben zeigt sich in der Liste aller Werte eine getätigte Auswahl an. In gewissen Situationen existiert noch der Zustand, dass in der Master-View ein Wert im *Highlight* steht. Bestätigt der Nutzer das *Highlight* mit der Maus, wechselt der Status auf selektiert. Das Hovern kann den Highlight-Wert ändern. Geschieht die Navigation durch die Werte mit der Tastatur, erhält genau ein Wert die *Cursor Position*. Durch das Betätigen gewisser vordefinierter Tasten ändert sich die *Cursor Position*. Bei einer Bestätigung mit der Tastatur ändert sich der Wert der *Cursor Position* auf selektiert. Die letzten zwei Zustandswerte haben keinen Einfluss auf das versendete Formular und sind nur im Master zu finden. Nachfolgend sind noch die Details zum Thema Browser erklärt.

2.4 Browser & HTML Renderer

Die wichtigsten Aspekte von der Theorie über die populären Browser bis hin zum Ablauf des Renderings sind hier aufgeführt. Dabei sind nur die bekanntesten Implementationen von Belang.

Ein Webbrowser dient als Zugang ins Internet und zur Anzeige von Webressourcen wie HTML, CSS und JavaScript. Er besteht aus einer Benutzeroberfläche, einer Browser- und einer Rendering-Engine. Für eine verständliche Darstellung der Inhalte auf dem UI verwendet die Browser-Engine einen sogenannten Renderer - mehr dazu später. Die Benutzeroberfläche dient als Schnittstelle zwischen dem Benutzer und der Datenschicht. Die Rendering-Engine interpretiert die Inhalte anhand des vorgegebenen Inhaltstyps. Einer der Engines ist der HTML-Renderer. Beim UI und der Bedienung zeigen sich die Uneinigkeiten zwischen den Browser-Herstellern, indem die Rendering-Engine den Code nicht gleich interpretiert. Anschliessend sind Informationen über den detaillierten Ablauf zur Anzeige eines HTML-Dokuments und die Rolle des Rendering dokumentiert.

2.4.1 Rendering Prozess

Der Aufruf einer Webseite beginnt mit dem HTTP-Request auf welchen eine HTTP-Response folgt. In diesem Bericht sind die gerade genannten Schritte vor dem eigentlichen erhalten der Daten nicht weiter wichtig. Der Response liefert schlussendlich die anzuseigenden Daten, welche in diesem Fall die tragende Rolle spielen.

Der Browser verarbeitet die erhaltenen Daten im HTML-Format weiter. Mehrere Schritte wandeln die einzelnen HTML-Elementen in sogenannte Nodes um. Aus den resultierenden Nodes entsteht durch Verknüpfungen eine Baumstruktur - der DOM. Das Document Object Model (DOM) (Abbildung 2.4 links oben) beschreibt die Eltern-Kind- und Geschwister-Beziehung der Nodes. Der Prozess bis zum CSS Object Model (CSSOM) gestaltet sich relativ ähnlich, ist aber nicht weiter wichtig.

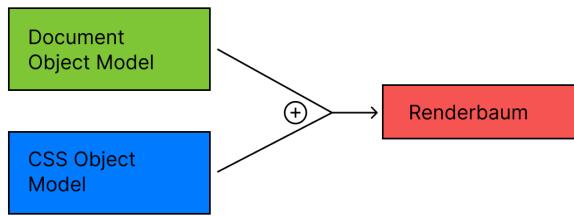


Abbildung 2.4: Rendering Prozess

Der DOM und der CSSOM sind unabhängig von einander. Der Browser kombiniert die beiden Bäume zu einem gemeinsamen Renderbaum (Abbildung 2.4 rechts). Der resultierende Baum repräsentiert nur sichtbare Elemente, wohingegen der DOM alle Elemente enthält. Der Renderbaum ist browserabhängig.

Dieses Wissen ist wichtig für das spätere Kapitel **Performance**. Der Grund ist, dass der Browser maximal 60 Mal pro Sekunde rendern kann. Jede Änderung am Browser-DOM¹ startet den kompletten Rendering Prozess von vorne. Zu viele Änderungen am DOM führen dazu, dass der Nutzer länger warten muss, bis die Webseite geladen ist.

Gründe für ein erneutes Rendern^[2]:

- Manipulation der Elemente des DOM
- Änderungen vom Inhalt (auch von Formularfeldern)
- Änderungen in den CSS-Eigenschaften
- Hinzufügen oder Entfernen von Stylesheets
- Ändern des Attributs `class`
- Größenänderung des Browserfensters
- Scrollen
- Pseudo-Klassenaktivierung

2.4.2 Bekannte Implementationen

Die neue Komponente soll in möglichst allen Browsern eine konsistente Erscheinung als auch Interaktion bieten. Durch die Erklärung aus Kapitel **Rendering Prozess** ist klar, dass die Renderer die Unterschiede im UI bewirken. Die Bedienungsabweichungen stammen grösstenteils vom zu Grunde liegenden Betriebssystem. Diese Erkenntnisse führen dazu, dass die neue Komponente auf Mac und Windows in den Browsern der jeweils geläufigsten Rendering-Implementationen zu testen ist. Zu den ausgewählten Webbrowsern zählen Google Chrome, Firefox (Mac und Windows), Safari (nur Mac) und Edge (nur Windows). Die Erläuterung dazu folgt im nächsten Abschnitt.

Die populärsten Browser mit 65% Marktdeckung basieren auf der Chromium-Basis und verwenden alle den HTML-Renderer Blink^[3]. Die Entwickler dieser Rendering-Engine sind

¹Im Renderbaum verwendeter und im Browser angezeigter DOM

²[Dev, 2020]

³[Wikipedia, 2024a]

die Open Source Community Chromium, Google, Intel und Samsung. Zu den Verwendern^[4] von Blink zählen unter anderem Google Chrome, Brave, Microsoft Edge, Opera und Vivaldi. WebKit^[5] – der Vorgänger des geläufigsten Renderer – findet sich im OSX Webbrowser Safari wieder. Diese von Apple, Google, KDE und Nokia entwickelte Rendering-Engine deckt 15% ab. Als bekanntester Vertreter des Restes zählt Firefox^[6] sowie weitere mozilla-basierte Browser, welche die Webseiten mit Gecko renderen. Es existieren noch weitaus mehr Renderer, welche aber eine sehr geringe Verbreitung aufweisen. Deswegen sind diese nicht weiter von Belang.

⁴[Ola und Markus, 2024a]

⁵[Wikipedia, 2024b]

⁶[Ola und Markus, 2024b]

Kapitel 3

Existierende Auswahlkomponenten

Hier sind die Gegenüberstellungen der bereits vorhandenen Möglichkeiten zur Darstellung einer Optionsauswahl aufgelistet. Eine Auflistung der möglichen Funktionalitäten zeigt die Grenzen dieser Elemente auf. Dabei spielen die UI- als auch die Interaktions-Unterschiede dieser Komponenten in verschiedenen Browsern eine tragende Rolle.

Als Basis dieser Arbeit dient das Country Select. Weiter existieren bereits die Elemente Buttons bzw. Links, `select` und `datalist`. Unter dem Country Select ist das Resultat des Projekt 5 zu verstehen. Die folgende Tabelle 3.1 zeigt einen Vergleich der genannten Möglichkeiten.

Tabelle 3.1: Vergleich der Auswahlmöglichkeiten

Kriterium	Buttons	Select	List	Country Input
Optimale Anzahl Elemente	1 – 3	7 ± 2	70 ± 20	ca. 250
Alternativ-Name	Links	Choice Input	Combo Box	keine
Falsche Auswahl	✗	✗	✓	✗
Multi-Auswahl	✓	✓	✗	✗
Readonly	✓	✗	✓	✗
Disabled	✓	✓	✓	✗
Werte-Typ	Skalar	Skalar	Skalar	Objekt
Interaktionsmöglichkeit	sehr begrenzt	gut	gut	gut & konsistent
Aktion bei Symbol-Eingabe	keine	Suche ²	Filter ¹	Suche ²
Merkmal	direkter Überblick	fixe Optionen	Eingabehilfe	Speziell für Länderauswahl
Anwendung	Navigationslink, Formularbuttons	Formularfeld mit begrenzter Auswahl	Filterbare Liste, Suchergebnis	Länderauswahl im Formular

¹ Filter: Liste verändern je nach Anzahl passender Werte

² Suche: Liste unverändert; erster zu der Eingabe passender Wert aus der Liste

Nebst den in der Tabelle aufgeführten Möglichkeiten existieren diverse Libraries und Frameworks. Viele dieser Lösungen für Auswahlkomponenten besitzen Abhängigkeiten zu weiteren Bibliothek-Komponenten. Durch das Einbinden der Frameworks bläst sich der eigene Code extrem auf. Diese Auswahlkomponenten bieten eine Menge an Funktionen an, welche jedoch die Anwendung sehr komplex machen. Die Einarbeitung dauert lange. Da das Ziel dieses Projekts eine schlanke und einfach verwendbare Komponente ist, sind die existierenden

Frameworks keine Lösung. Dieser Bericht legt keinen weiteren Fokus auf externe Bibliotheken.

3.1 HTML Datalist vs. Select

Die folgenden Unterkapitel zeigen die Möglichkeiten, welche die HTML-Elemente `input`, `option`, `datalist` und `select` bieten. Zudem zeigen tabellarische Gegenüberstellungen die Unterschiede und Inkonsistenzen dieser in verschiedenen Browsern und Betriebssystemen auf. Hierbei liegt der Fokus mehr auf der Interaktion mit den Komponenten als auf der Darstellung.

3.1.1 Option

Die beiden Auswahlkomponenten – `datalist` und `select` – verwenden `option`-Elemente. Die einzelnen Optionen wie in Code 3.1 Zeile 1 besitzen ein `value`-Attribut und einen Inhalt – das Label. Alternativ ist das `label`-Attribut zu nutzen und dafür das Value an die Stelle des Inhalts zu platzieren. Firefox unterstützt diese Variante jedoch nicht. Das Weglassen des `value`- und `label`-Attributes führt dazu, dass beide den Wert innerhalb des öffnenden und schliessenden Tags erhalten.

Code 3.1: Option Beispiel

```
1 <option value="dog"          > Dog    </option>
2 <option label="Cat" selected> cat    </option>
3 <option           disabled> Mouse </option>
```

Das HTML-Element kann das Attribut `disabled` oder `selected` erhalten. Code 3.1 Zeile 2 definiert durch eine `selected` Option den initial ausgefüllten Wert. Dies Attribut funktioniert nur im Zusammenhang mit dem `select`-Container. Bei einem `disabled` Element – wie in Code 3.1 Zeile 3 – erscheint das UI ausgegraut. Dieser Eintrag lässt keine Interaktion zu.

Das Designen von `options`^[1] beschränkt sich auf die Text-/ Hintergrundfarbe, welche jedoch nur in Firefox funktionieren. Die anderen Browser lassen kein Styling der Auswahl-Elemente zu bzw. zeigen diese nicht an. Es ist nur ein skalarer Text als Inhalt erlaubt. Das bedeutet, keine weiteren Verschachtelung von Elementen sind möglich. Als Eltern-Elemente sind `select`, `optgroup` und `datalist` erlaubt. Alle gängigen Browser unterstützen das `option`-Element.

3.1.2 Select

Das `select`-Element speichert Werte durch die einzelnen `option`-Kinder. Eine möglich Anwendung ist in Code 3.2 ersichtlich.

Code 3.2: Disabled Select Beispiel

```
1 <select name="animal" disabled>
2   <option value="dog"  > Dog    </option>
3   <option value="cat"  > Cat    </option>
4   <option value="mouse"> Mouse </option>
5 </select>
```

^[1] [contributors, 2023]

Die typischen Attribute für Eingabefelder wie `disabled`, `form`, `name` und `required` sind bei einem Select ebenfalls verwendbar. Durch das `disabled` zeigt sich das Element in einer ausgegrauten Erscheinung und bietet dem Nutzer keine Interaktionsmöglichkeiten mehr. Der Container – z.B. `fieldset` – vererbt dieses Attribut weiter an seine Kinder. Die `form`-Eigenschaft definiert das dazugehörige Formular und `name` den Key für das Key-Value Paar bzw. Name des Feldes im Formular. Das `required` erzwingt eine Eingabe, um das Senden des Formulars freizuschalten. Für eine bessere Accessibility benötigt das Select eine `id`, um das Label zuweisen zu können. Standardmäßig ist für dieses Feld nur eine einzelne Auswahl möglich. Durch die Ergänzung des Attributs `multiple` ist es möglich mehrere Werte zu markieren. Bei einer Vorauswahl mehrerer Listenelemente durch `selected` muss die Komponente ein Multiselect sein. Wenn jedoch bei einem Single-Select mehrere Optionen das `selected`-Attribut enthalten, gilt das zuletzt Definierte als Default. Das `autocomplete` funktioniert wie bei anderen Inputs, indem Vorschläge vom User-Agent-Feature auftauchen. Durch die `autofocus`-Eigenschaft sind Interaktionen mit dem Feld direkt nach dem Laden möglich. Die Definition des `size`-Attributs steuert die Anzahl sichtbarer Elemente, wobei der Default für Single-Selects bei eins und mehrfache Auswahl bei vier liegt. Die in Kapitel **Bekannte Implementationen** definierten Browser unterstützen alle Attribute des Selects – ausser der `size`. Die meisten Mobile-Browser supporten die Grösse nicht.

Abgesehen von der Grösse ist die Umgestaltung des Elements browserunabhängig kaum möglich. Es gibt jedoch aufwendige Wege, den Inhalt zu klonen und durch Wrapper neu zu stylen. In diesem Fall ist es notwendig, die Logik und die Interaktionen für die Accessibility neu zu implementieren. Gewisse Stylings lassen sich anwenden, wobei aber nicht alle Browser diese in der selben Weise übernehmen. Durch die komplexe Struktur des Selects ist eine eigene Darstellung schwierig zu kontrollieren.

Der Inhalt^[2] des Elements können `options` oder `optgroup` sein. Mehr zur `optgroup` kommt im nachfolgenden Unterkapitel. Die ARIA-Rolle gibt der Komponente die Funktion einer Combobox (ohne `size` & `multiple`) oder Listbox.

Die mehrfache Auswahl rein per Tastatur bietet in der Interaktion geringere Möglichkeiten als mit der Maus. Die Maus kann mit gedrückter Cmd (Mac) bzw. Ctrl (Windows) Taste weitere Elemente dazu selektieren. Mit dem Drücken der Shift-Taste lassen sich alle Elemente zwischen der ersten und der letzten Option markieren. Damit sich die zuvor markierte Auswahl nicht aufhebt, ist darauf zu achten, beim Anwenden beider Techniken den Bereich mit Shift zuerst auszuwählen. Für das Markieren mehrerer Werte mittels Tastatur ist es unabdingbar, zuerst zum ersten Element zu navigieren. Mit Shift und den Pfeiltasten ↑ und ↓ ist ein Bereich wählbar. Firefox unterstützt noch die einzelne Mehrfachauswahl über die Tastatur. Auf dem Mac sind die Tastenkombinationen Systembefehle. Daher gibt es keine Alternative als sie zuvor auszuschalten bzw. umzustellen.

Optgroup

Die `optgroup` dient als Zusatz-Element um in `selects` Optionen zu Gruppieren und mit einem Zwischentitel zu versehen. Der Titel lässt sich durch das `label`-Attribut – wie in Code 3.3 – setzen, ist aber nicht selektierbar.

Code 3.3: Optgroup Beispiel

```
1 <select name="animal">
2   <optgroup label="Big animal">
```

²[contributors, 2024c]

```
3     <option value="dog"> Dog </option>
4     <option value="cat"> Cat </option>
5   </optgroup>
6   <optgroup label="Small animal" disabled>
7     <option value="mouse" > Mouse </option>
8     <option value="hamster"> Hamster </option>
9   </optgroup>
10 </select>
```

Das `disabled` ermöglicht das Ausgrauen eines ganzen Blocks von Auswahloptionen. In Code 3.3 ist dies auf Zeile 6 zu sehen. Die Optionen, welche in diesem Element enthalten sind, erben das Attribut. Als Inhalt dienen `option`-Elemente und selbst besitzt es als Eltern-Element ein `select`. Die `optgroup` besitzt die ARIA-Rolle einer Gruppe. Alle Browser unterstützen dieses Element.

3.1.3 Datalist

Wie in Code 3.4 ersichtlich besteht die Datalist aus zwei Elementen – einem Input-Feld und einem Daten-Container. Die Datenliste besitzt keine speziellen eigenen Attribute. Das Element benötigt von den globalen Attributen zumindest eine `id`. Die Id dient zur Verknüpfung der Liste mit dem Input-Feld. Das folgende Unterkapitel **Input** beschreibt das Eingabefeld und dessen Typen im Detail.

Code 3.4: Datalist Beispiel

```
1 <input type="text" name="animal"
2   list="data" placeholder="Animal" />
3 <datalist id="data">
4   <option> Dog </option>
5   <option> Cat </option>
6   <option> Mouse </option>
7 </datalist>
```

Das Stylen der `datalist`^[3] ist sehr begrenzt bzw. nicht möglich. Der Daten-Container reagiert nicht auf den Zoom des Browsers. Gewisse Screenreader ignorieren die Vorschlagsliste und lesen diese dadurch auch nicht vor. Das Element erhält die ARIA-Rolle einer Listbox.

Die `options` einer `datalist` besitzen normalerweise nur ein `value`-Attribut und kein Label. Bei einer zusätzlichen Label-Definition kann das je nach Browser zu abweichenden Darstellungen kommen. Während Firefox nur das Label in der Liste anzeigt, visualisieren die anderen Browser das Label und das Value gemeinsam. Die Browser, welche Label und Value anzeigen, heben den Wert ein wenig hervor.

Die Darstellung kann zu Missverständnissen führen, da das Eingabefeld bei einer Auswahl nur das Value anzeigt. Die `options`-Werte können sich dem Typ des Inputs anpassen. Generell unterstützen alle Browser die `datalist`, aber Firefox nur begrenzt.

Nicht alle Browser unterstützen die Datenliste für jeden Eingabe-Typ^[4]. Der textuelle Typ funktioniert in allen Browsern und die Liste öffnet sich nach einem Klick bzw. Doppelklick auf das Feld. Vordefinierte Datum- und Zeit-Typen funktionieren nur in den Chromium-basierten Browsern. Firefox und Safari zeigen den normalen Eingabe-Container, als ob die Liste nicht verknüpft ist. Bei der Verwendung einer Liste mit einer Range zeigen alle Browser die Optionen durch Markierungen auf dem Slider an. Die Kombination einer

³[contributors, 2024a]

⁴[contributors, 2024a]

`datalist` mit einer Farbpalette zeigt eine breite, aber unterschiedliche Unterstützung. Unter den gängigen Browsern ist Firefox auf OSX der Einzige, welcher die Liste nicht darstellt. Zusammengefasst bieten für Listen mit den unterschiedlichsten Typen Chromium-basierte Browser die beste Unterstützung.

Input

Dieser Abschnitt behandelt nur den Teil, welcher im Bezug auf die `datalist` von Belang ist. Das `list`-Attribut verknüpft das Input – in Code 3.4 auf Zeile 1 und 2 – mit den Auswahloptionen der Liste. Bei der Verwendung zusammen mit der `datalist` ist dieses Attribut Pflicht. Dadurch erscheinen die Auswahloptionen während der Bedienung des Input-Feldes.

Dieser Paragraph behandelt die in diesem Kontext für alle Typen geltenden Attribute. Das `autocomplete`-Attribut dient zur Anzeige der Hinweise für das Autofill-Feature der Browser. Das `disabled` schaltet die Interaktionsmöglichkeiten aus und deaktiviert somit die Komponente. Ist das Eingabefeld ausserhalb eines Formulars platziert, kann das Attribut `form` eine Verknüpfung zu einem auf der Seite existierenden Formular herstellen. Im abgesendeten Formular dient die Eigenschaft `name` zur Identifizierung des Wertes im `value`-Attribut. Der Typ des Eingabefeldes – angegeben durch das `type` – definiert welche UI-Erscheinung das Input im Browser erhält und welche Werte zulässig sind. Als Standard-Typ ist `text` definiert, wodurch dieses Attribut optional ist. Mehr zu den für diese Arbeit wichtigen Typen ist im Unterkapitel **Input-Typen mit Datalist-Unterstützung** zu lesen. Die `id` als globale Eigenschaft dient bei den Eingabefeldern zusätzlich noch zur Verknüpfung mit einem Label-Element und somit einer besseren Accessibility.

Weiter existieren noch die zwei Attribute `readonly` und `required`, welche bei allen Typen ausser `range` und `color` definiert sind. Durch die Ergänzung der ersten Eigenschaft ist der Wert nicht mehr änderbar, aber es bleibt im Verlauf der fokussierbaren Komponenten erhalten. Das `required` erzwingt eine Eingabe. Die Eigenschaften `min`, `max` und `step` unterstützen die Konfiguration der numerischen Typen⁵. Die ersten beiden begrenzen die Werte auf einen Bereich oder ein Intervall. Mit `step` lässt sich die Grösse eines Schrittes einstellen. Die textlichen Felder⁶ besitzen die Attribute `maxlength`, `minlength`, `pattern` und `size`. Diese Texttypen zusammen mit `number` können durch `placeholder` einen Platzhaltertext erhalten. Die Min- und Max-Länge schränken die Textlänge der Eingabe ein. Die Vorgabe eines Regex-Musters im `pattern`-Attribut kann die Eingabe weiter eingrenzen. Das Formular validiert die Felder vor dem Senden anhand des Patterns und markiert fehlerhafte Einhaben als invalide. Gewisse Typen – wie z.B. `url`, `tel` und `email` – haben bereits eine Pattern hinterlegt. Mit `size` lässt sich die Grösse bzw. Anzahl sichtbarer Zeichen angeben. Weitere Informationen zum Styling eines Input-Feldes stehen im Unterkapitel **CSS für Input-Felder**.

Das Input besteht aus einem selbst-schliessenden Tag. Die ARIA-Rolle ist vom Typ abhängig und kann als Textbox, Combobox, Spinbutton, Slider, Searchbox, Telbox oder keiner speziellen Rolle zugeordnet sein. In den hier erwähnten Möglichkeiten der Komponente existiert eine grossflächige Browserunterstützung. Die einzigen Ausnahmen beziehen sich auf die Typen `week` und `month`, welche im Firefox und Safari nicht funktionieren.

⁵`date`, `month`, `week`, `time`, `datetime-local`, `number`, `range`

⁶in diesem Zusammenhang: `text`, `search`, `url`, `tel`, `email`, `password`

Input-Typen mit Datalist-Unterstützung

Bei den textuellen Typen^[7] existieren `text`, `search`, `number`, `email`, `url` und `tel`. Diese Typen erscheinen mehr oder weniger als normales, einzeiliges Texteingabefeld. Die ersten zwei der Auflistung verwalten einen Text ohne spezielle Anforderungen. Der Typ `number` dient zum Speichern der Werte als Zahlen und zeigt dies durch die Ergänzung zweier Buttons im UI-Feld. Die letzten drei Typen sind Textfelder, welche bereits ein passendes Pattern für E-Mail, URL oder Telefonnummer hinterlegt haben. Zudem zeigen diese Felder bei dynamischen Tastaturen eine angepasste Layout⁸. Typen in der Kategorie Date-Time sind `month`, `week`, `date`, `time` und `datetime-local`.

Bei einer ungefähren Eingabe einer Zahl auf einem Intervall bietet sich `range` an. Im UI zeigt sich diese Komponente als Slider, wobei sich der Standard Wert in der Mitte findet. Zur Festlegung der Limiten dienen `min` und `max`. Für eine Farbauswahl kann der Typ `color` zur Anwendung kommen. Die gängigen Browser zeigen nach dem Aufklappen Farbpaletten, welche sich aber in der Darstellung unterscheiden. Firefox beispielsweise greift auf die vom System gestellte Farbpalette zurück, wo Chromium-basierte Browser eine eigene bieten.

CSS für Input-Felder

Bei einem Input^[9] bestehen mehrere Möglichkeiten dieses umzugestalten.

Es existieren eigene CSS-Pseudo-Klassen, wobei die folgende Aufzählung nur einen Ausschnitt aufzeigt.

- `:enabled` bzw. `:disabled` – reagiert auf das `disabled`-Attribut
- `:read-write` bzw. `:read-only` – reagiert auf das `readonly`-Attribut
- `:valid` bzw. `:invalid` – reagiert beim Abschicken des Formulars auf die Client-Side Validität¹⁰ des Felds
- `:user-invalid` – reagiert beim Verlassen des Inputs auf die Client-Side Validität des Felds
- `:in-range` bzw. `:out-of-range` – reagiert auf das `min`- und `max`-Attribut
- `:optional` bzw. `:required` – reagiert auf das `required`-Attribut
- `:blank` – reagiert wenn ein Feld leer ist

Weiter gibt es noch das Pseudo-Element `::placeholder`, welches das Stylen des Platzhalters erlaubt. Das CSS-Property `caret-color` färbt den Cursor¹¹ in Inputs um. Im Gegensatz zur `datalist` und dem `select` lässt sich das normale Eingabefeld relativ gut designen.

⁷[contributors, 2024a]

⁸z.B.: @ und . ist bei `email` immer sichtbar, oder bei `tel` sind die Zahlen besser dargestellt

⁹[contributors, 2024b]

¹⁰Einhaltung der Regeln für das Input – z.B. ein gegebenes `pattern`

¹¹blinkender Strich in einem Eingabefeld

3.2 Browser-Inkonsistenzen

Die Komponenten Select & Datalist variieren in der Ansicht als auch in der Interaktion. Die Unterschiede sind system-, browser- und komponentenabhängig. Nachfolgend sind die Inkonsistenzen genauer erläutert.

3.2.1 UI Unterschiede

Das Select ist in der geschlossenen Form auf der rechten Seite immer mit mindestens einem Pfeil nach unten ausgestattet. Safari (OSX und iOS) verwendet als Icon einen Doppelpfeil (Abbildung 3.1). Firefox zeigt das Feld in grau (Abbildung 3.2), während Chrome und Edge einen weißen Hintergrund (Abbildung 3.3) verwenden. Safari auf iOS stellt die Komponente ohne Rahmen dar und deswegen ist der Hintergrund in einem hellen Grau gehalten. Da der Fokus dieser Arbeit auf der Desktop-Anwendung liegt, finden sich die Bilder der Mobile-Browser im Anhang C.

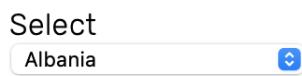


Abbildung 3.1: Geschlossenes Select OSX Safari



Abbildung 3.2: Geschlossenes Select OSX Firefox



Abbildung 3.3: Geschlossenes Select Windows Chrome

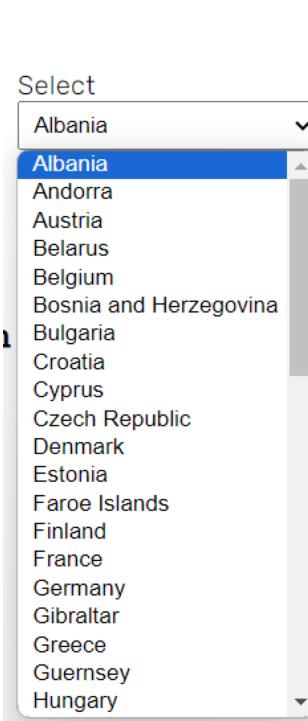


Abbildung 3.4: Offenes Select Windows Chrome

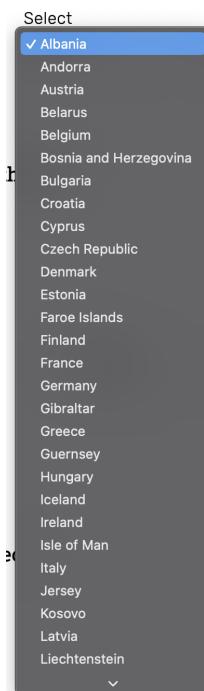


Abbildung 3.5: Offenes Select OSX Chrome

Die geöffnete Liste ist bei Firefox als einziges relativ konsistent. Sie erscheint in einem mittel- bis hellgrauen Container. Die anderen Browser sind einstellungsabhängig und zeigen den Container weiss oder dunkelgrau (Abbildungen 3.4 und 3.5) an. Je nach Anzahl der enthaltenen Elementen erscheint die Liste darunter (darüber) oder überdeckt den Container des ausgewählten Wertes. Am Wenigsten lässt sich das Element in Safari stylen.

Da nicht jeder Browser ein Icon anzeigt, ist die Datalist weniger konsistent. Safari (Mac) und Firefox stellen keinen visuellen Hinweis (Abbildungen 3.6 und 3.7) auf einen Listen-Container dar. Safari auf iOS zeigt hingegen in jedem Fall – bezogen auf den textuellen Typ

– einen nach unten zeigenden Pfeil an. Andere Browser blenden auf der rechten Seite beim Hovern oder beim Besitzen des Fokus das Icon (Dreieck nach unten zeigend) ein.



Abbildung 3.6:
Geschlossene Datalist
OSX Safari



Abbildung 3.7:
Geschlossene Datalist
OSX Firefox

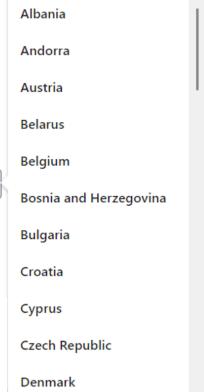


Abbildung 3.8: Offene Datalist
Windows Chrome

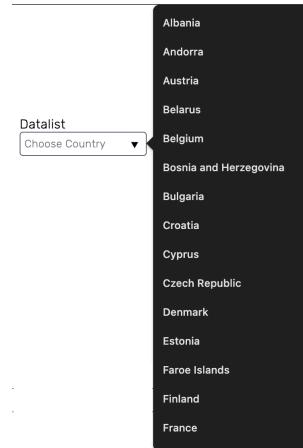


Abbildung 3.9: Offene Datalist OSX Chrome

Bei Firefox unterscheidet sich das Öffnen der Liste ebenfalls. Wenn das Feld den Fokus noch nicht besitzt, benötigt es zwei Klicks. Die anderen Browser lassen die Liste bereits beim ersten Kick erscheinen. Die Liste selbst verhält sich je nach Browser und Inhalt verschieden, indem sie seitlich oder darunter (darüber) erscheint. Die Übernahme des Dark- bzw. Light-Modes als Container-Farbe hängt vom Browser und dem Anwendungskontext ab. Die Liste überdeckt das Eingabefeld nie. Die Abbildungen 3.8 und 3.9 zeigen Beispiele der geöffneten Datalist.

Ein Blick auf die mobilen Browser wie iOS-Safari, Android-Firefox und -DuckDuckGo zeigen weitere UI-Unterschiede. Für eine bessere Bedienbarkeit zeigen sich die geöffneten Selects auf Android – gegenüber der Desktop-Versionen – in einem anderen Design. Die Liste öffnet sich als Dialog-Popup und füllt je nach Anzahl der Werte fast den kompletten Display aus. Das selektierte Element besitzt einen ausgefüllten Radio-Button auf der rechten Seite. Auf iOS-Browser erhält die ausgewählte Option auf der linken Seite ein Check-Icon (✓). Der Listen-Container erscheint ähnlich zum OSX-Browser als Dropdown-Liste. Nach einer Auswahl schliessen die mobilen Selects automatisch.

Die Datalist zeigt ihren Inhalt nur bei der Interaktion mit Pfeil auf der rechten Seite. Abgesehen von der Grösse der einzelnen Einträge für die Bedienbarkeit unterscheiden diese sich kaum von den Desktop-Versionen. Die Selektionen erscheinen gegenüber der anderen Optionen in keinem speziellen Design.

Weitere Inkonsistenzen sind in den Bildern im Anhang C zu sehen. Die nachfolgenden Abschnitte **3.2.2** bis **3.2.6** beschreiben die möglichen Interaktionen. Die Tabellen stellen das Datalist, Single- als auch Multiselect einander gegenüber. Im Anschluss an die Gegenüberstellungen der diversen Browser weist das Kapitel **Fazit** detaillierter auf die Inkonsistenzen hin.

3.2.2 Edge Browser

Die folgende Tabelle 3.2 zeigt den Vergleich der typischen Interaktionen zwischen Datalist, Single- und Multiselect in Edge auf Windows. Auf Windows verhält sich Edge sehr ähnlich wie Chrome, da die Codebasis beider Browser Chromium ist.

Tabelle 3.2: Vergleich Interaktion Datalist & Select in Edge (Windows)

Kriterium	Datalist	Select	Multiselect
geschlossen	geschlossen	-	
offen	offen	offen	
↑ / ↓	Liste öffnen Highlight ändern	Selektion ändern Selektion ändern	Selektion ändern
← / →	Cursor ¹ bewegen Cursor ¹ bewegen	Selektion ändern -	-
Buchstaben	Schreiben & Liste öffnen Schreiben & Liste filtern ²	Selektion auf Suchergebnis ³ ändern Selektion auf Suchergebnis ³ ändern	Selektion aufheben & Selektion auf Suchergebnis ³ ändern
Leerschlag	Schreiben & Liste öffnen Schreiben & Liste filtern ²	Liste öffnen -	-
Backspace	Löschen & Liste öffnen (wenn Feld nicht leer) Löschen & Liste filtern ²	- -	-
Esc	- Liste schliessen	- Liste schliessen	-
Enter	<i>in Formular:</i> senden <i>sonst:</i> - <i>mit Highlight:</i> ändern <i>ohne:</i> Form senden / -	Liste öffnen Selektion ändern & Liste schliessen	-
Tab	Input-Feld verlassen Input-Feld verlassen	Input-Feld verlassen Liste schliessen	-
PageUp / PageDown	Fenster scrollen Highlight auf View-Rand dann seitenweise ändern	Selektion auf jeden 3. Wert ändern Selektion auf View-Rand dann seitenweise ändern	Selektion auf vorherige/ nächste size ⁴ Stelle ändern
Home / End	Highlight auf ersten/ letzten Wert ändern Highlight auf ersten/ letzten Wert ändern	Selektion auf ersten/ letzten Wert ändern Selektion auf ersten/ letzten Wert ändern	Selektion auf ersten/ letzten Wert ändern
Scroll	Fenster scrollen <i>Aussen:</i> Liste fixed ⁵ offen <i>Innen:</i> Liste scrollen	Fenster scrollen <i>Aussen:</i> Liste schliessen <i>Innen:</i> Liste scrollen	<i>Aussen:</i> Fenster scrollen <i>Innen:</i> Liste scrollen
Hover	- Highlight ändern	UI-Anpassung Highlight ändern	-
Click	Liste öffnen Selektion ändern	Liste öffnen Selektion ändern & Liste schliessen	Selektion aufheben & Selektion ändern

¹ Blinkender Strich in einem Eingabefeld

² Filter: Input-Begriffe and-verknüpft enthalten; Liste verändern je nach Anzahl passender Werte

³ Suche: Erster zu der Eingabe passender Wert aus der Liste, wenn Eingabe nicht passend ⇒ letzter noch übereinstimmender Wert; Liste unverändert; nach Pause / Debounce-Ablauf ⇒ neue Suche

⁴ Wert von dem size-Attribut

⁵ Verhalten der CSS-Position fixed

3.2.3 Chrome Browser

Die Gegenüberstellung 3.3 beschreibt die bekannten Bedienungsmöglichkeiten der drei existierenden Auswahlkomponenten in Chrome. Dabei spielt das System (Windows oder Mac) nur bei der Tastenkombination und nicht bei der Reaktion eine Rolle. Auf dem Mac verhält sich Chrome ähnlich wie auf Windows, Designaspekte können sich unterscheiden.

Tabelle 3.3: Vergleich Interaktion Datalist & Select in Chrome (Mac / Windows)

Kriterium	Datalist	Select	Multiselect
	geschlossen offen	geschlossen offen	- offen
↑ / ↓	Liste öffnen Highlight ändern	Selektion ändern Selektion ändern	Selektion ändern
← / →	Cursor ¹ bewegen Cursor ¹ bewegen	Selektion ändern -	-
Buchstaben	Schreiben & Liste öffnen Schreiben & Liste filtern ²	Selektion auf Suchergebnis ³ ändern Selektion auf Suchergebnis ³ ändern	Selektion aufheben & Selektion auf Suchergebnis ³ ändern
Leerschlag	Schreiben & Liste öffnen Schreiben & Liste filtern ²	Liste öffnen -	-
Backspace	Löschen & Liste öffnen (wenn Feld nicht leer) Löschen & Liste filtern ²	- -	-
Esc	- Liste schliessen	- Liste schliessen	-
Enter	<i>in Formular:</i> senden <i>sonst:</i> - <i>mit Highlight:</i> ändern <i>ohne:</i> Form senden / -	Liste öffnen Selektion ändern & Liste schliessen	-
Tab	Input-Feld verlassen Input-Feld verlassen	Input-Feld verlassen Liste schliessen	-
PageUp / PageDown	Fenster scrollen Highlight auf View-Rand dann seitenweise ändern	Selektion auf jeden 3. Wert ändern Selektion auf View-Rand dann seitenweise ändern	Selektion auf vorherige/ nächste size ⁴ Stelle ändern
Home / End	Highlight auf ersten/ letzten Wert ändern Highlight auf ersten/ letzten Wert ändern	Selektion auf ersten/ letzten Wert ändern Selektion auf ersten/ letzten Wert ändern	Selektion auf ersten/ letzten Wert ändern
Scroll	Fenster scrollen <i>Aussen:</i> Liste fixed ⁵ offen <i>Innen:</i> Liste scrollen	Fenster scrollen <i>Aussen:</i> Liste schliessen <i>Innen:</i> Liste scrollen	<i>Aussen:</i> Fenster scrollen <i>Innen:</i> Liste scrollen
Hover	- Highlight ändern	UI-Anpassung Highlight ändern	-
Click	Liste öffnen Selektion ändern	Liste öffnen Selektion ändern & Liste schliessen	Selektion aufheben & Selektion ändern

¹ Blinkender Strich in einem Eingabefeld

² Filter: Input-Begriffe and-verknüpft enthalten; Liste verändern je nach Anzahl passender Werte

³ Suche: Erster zu der Eingabe passender Wert aus der Liste, wenn Eingabe nicht passend ⇒ letzter noch übereinstimmender Wert; Liste unverändert; nach Pause / Debounce-Ablauf ⇒ neue Suche

⁴ Wert von dem size-Attribut

⁵ Verhalten der CSS-Position fixed

3.2.4 Firefox Browser

Die nachfolgende Aufstellung 3.4 zeigt die Reaktionen auf gewisse Benutzerinteraktionen in Firefox. Wie auch auf Windows zeigt Firefox auf dem Mac konsistentes Verhalten, jedoch mit typischen OSX Designanpassungen. Die Interaktions-Feedbacks auf Mac können sich leicht von der Windows-Version unterscheiden.

Tabelle 3.4: Vergleich Interaktion Datalist & Select in Firefox (Mac / Windows)

Kriterium	Datalist	Select	Multiselect
	geschlossen offen	geschlossen offen	- offen
↑ / ↓	Liste öffnen Highlight ändern	Selektion ändern Selektion ändern	Selektion ändern
← / →	Cursor ¹ bewegen <i>in Liste</i> : Highlight wählen <i>in Wertefeld</i> : Cursor ¹ bewegen	Selektion ändern	Selektion ändern
Buchstaben	Schreiben & Liste öffnen Schreiben & Liste filtern ²	Selektion auf Suchergebnis ³ ändern Selektion auf Suchergebnis ³ ändern	Selektion aufheben & Selektion auf Suchergebnis ³ ändern
Leerschlag	Schreiben & Liste öffnen Schreiben & Liste filtern ²	Liste öffnen -	-
Backspace	Löschen & Liste öffnen (wenn Feld nicht leer) Löschen & Liste filtern ²	- -	-
Esc	- Liste schliessen	- Liste schliessen	-
Enter	<i>in Formular</i> : senden <i>sonst</i> : - <i>mit Highlight</i> : ändern <i>ohne</i> : Form senden / -	- Selektion ändern & Liste schliessen	-
Tab	Input-Feld verlassen Input-Feld verlassen	Input-Feld verlassen Liste schliessen	Input-Feld verlassen
PageUp / PageDown	Liste öffnen Highlight auf View-Rand dann seitenweise ändern	Selektion auf jeden 20. Wert ändern Selektion auf View-Rand dann seitenweise ändern	Selektion auf View-Rand ändern
Home / End	Highlight auf ersten/ letzten Wert ändern Highlight auf ersten/ letzten Wert ändern	Selektion auf ersten/ letzten Wert ändern Selektion auf ersten/ letzten Wert ändern	Selektion auf ersten/ letzten Wert ändern
Scroll	Fenster scrollen <i>Aussen</i> : Liste schliessen <i>Innen</i> : Liste scrollen & Highlight ändert am Ende	Fenster scrollen <i>Aussen</i> : Liste schliessen <i>Innen</i> : Liste scrollen	<i>Aussen</i> : Fenster scrollen <i>Innen</i> : Liste scrollen
Hover	UI-Anpassung Highlight ändern	- Highlight ändern	-
Click	Liste öffnen Selektion ändern	Liste öffnen Selektion ändern & Liste schliessen	Selektion aufheben & Selektion ändern

¹ Blinkender Strich in einem Eingabefeld

² Filter: Leerschlag als normales Symbol gezählt; Liste verändern je nach Anzahl passender Werte

³ Suche: Erster zu der Eingabe passender Wert aus der Liste, wenn Eingabe nicht passend ⇒ letzter noch übereinstimmender Wert; Liste unverändert; nach Pause / Debounce-Ablauf ⇒ neue Suche

3.2.5 Safari Browser

Die Interaktionen auf Safari sind in der Tabelle 3.5 einander gegenübergestellt. Der Mac Standard-Browser zeigt zu den bisher gesehenen Vergleichen die grösste Abweichung.

Tabelle 3.5: Vergleich Interaktion Datalist & Select in Safari (Mac)

Kriterium	Datalist	Select	Multiselect
	geschlossen	geschlossen	-
	offen	offen	offen
↑ / ↓	- Highlight ändern	Liste öffnen Highlight ändern	Selektion ändern
← / →	Cursor ¹ bewegen Cursor ¹ bewegen	- -	-
Buchstaben	Schreiben & Liste öffnen Schreiben & Liste filtern ²	Selektion auf Suchergebnis ³ ändern Highlight auf Suchergebnis ³ ändern	Selektion aufheben & Selektion auf Suchergebnis ³ ändern
Leerschlag	Schreiben & Liste öffnen Schreiben & Liste filtern ²	Liste öffnen Selektion ändern	-
Backspace	Löschen & Liste öffnen Löschen & Liste filtern ²	- Highlight auf ersten Wert ändern	-
Esc	- -	- Liste schliessen	-
Enter	<i>in Formular:</i> senden <i>sonst:</i> - Highlight wählen & Liste schliessen	- Selektion ändern & Liste schliessen	-
Tab	Input-Feld verlassen Input-Feld verlassen	Input-Feld verlassen -	-
PageUp/-Down (fn & ↑ / ↓)	Fenster scrollen Fenster scrollen	Fenster scrollen Selektion auf ersten/letzten Wert ändern	Selektion auf vorherige/nächste <i>size</i> ⁴ Stelle ändern
Home / End (fn & ← / →)	Fenster scrollen Fenster scrollen	Fenster scrollen Selektion auf ersten/letzten Wert ändern	Selektion auf ersten/letzten Wert ändern
Scroll	Fenster scrollen <i>Aussen:</i> Liste schliessen <i>Innen:</i> Liste scrollen	Fenster scrollen <i>Aussen:</i> - <i>Innen:</i> Liste scrollen & Highlight ändern	<i>Aussen:</i> Fenster scrollen <i>Innen:</i> Liste scrollen
Hover	- -	- Highlight ändern	-
Click	Liste öffnen Selektion ändern	Liste öffnen Selektion ändern & Liste schliessen	Selektion aufheben & Selektion ändern

¹ Blinkender Strich in einem Eingabefeld

² Filter: Leerschlag als normales Symbol gezählt; Liste verändern je nach Anzahl passender Werte

³ Suche: Erster zu der Eingabe passender Wert aus der Liste, wenn Eingabe nicht passend ⇒ sortiert nach folgender Wert; Liste unverändert; nach Pause / Debounce-Ablauf ⇒ neue Suche

⁴ Wert von dem size-Attribut

3.2.6 Browser auf Android & iOS

Der Vergleich der Tabellen 3.6 und 3.7 zeigt, dass auf dem selben Android-Gerät Abweichungen existieren. Nicht alle Bedienungsmöglichkeiten führen zur selben Reaktion.

Tabelle 3.6: Vergleich Interaktion Datalist & Select in Firefox (Android, mobile)

Kriterium	Datalist	Select	Multiselect
geschlossen	geschlossen	geschlossen	geschlossen
offen	offen	offen	offen
Buchstaben	Schreiben nicht möglich	nicht möglich nicht möglich	nicht möglich nicht möglich
Leerschlag	Schreiben nicht möglich	nicht möglich nicht möglich	nicht möglich nicht möglich
Backspace	Löschen nicht möglich	nicht möglich nicht möglich	nicht möglich nicht möglich
Enter	<i>in Formular:</i> senden <i>ohne:</i> - nicht möglich	nicht möglich nicht möglich	nicht möglich nicht möglich
Scroll	Fenster scrollen - <i>Aussen:</i> - <i>Innen:</i> Liste scrollen	Fenster scrollen <i>Aussen:</i> - <i>Innen:</i> Liste scrollen	Liste scrollen <i>Aussen:</i> - <i>Innen:</i> Liste scrollen
Click	- <i>Aussen:</i> - <i>Innen:</i> Selektion ändern & Liste schliessen	Liste öffnen <i>Aussen:</i> - <i>Innen:</i> Selektion ändern & Liste schliessen	Liste öffnen <i>Aussen:</i> - <i>Innen:</i> Selektion ändern

* nicht möglich: Virtuelle Tastatur ist nicht sichtbar, somit kann keine Interaktion stattfinden

Tabelle 3.7: Vergleich Interaktion Datalist & Select in DuckDuckGo (Android, mobile)

Kriterium	Datalist	Select	Multiselect
geschlossen	geschlossen	geschlossen	geschlossen
offen	offen	offen	offen
Buchstaben	Schreiben & Liste öffnen Schreiben & Liste filtern ¹	nicht möglich nicht möglich	nicht möglich nicht möglich
Leerschlag	Schreiben & Liste öffnen Schreiben & Liste filtern ¹	nicht möglich nicht möglich	nicht möglich nicht möglich
Backspace	Löschen & Liste öffnen (wenn Feld nicht leer) Löschen & Liste filtern ¹	nicht möglich nicht möglich	nicht möglich nicht möglich
Enter	<i>in Formular:</i> senden <i>ohne:</i> Input-Feld verlassen <i>in Formular:</i> senden <i>ohne:</i> Input-Feld verlassen	nicht möglich nicht möglich	nicht möglich nicht möglich
Scroll	Fenster scrollen <i>Aussen:</i> Liste bleibt offen <i>Innen:</i> Liste scrollen	Fenster scrollen <i>Aussen:</i> - <i>Innen:</i> Liste scrollen	Fenster scrollen <i>Aussen:</i> - <i>Innen:</i> Liste scrollen
Click	<i>in Feld:</i> - <i>Pfeil:</i> Liste öffnen <i>Aussen:</i> Liste schliessen <i>Innen:</i> Selektion ändern & Liste schliessen	Liste öffnen <i>Aussen:</i> - <i>Innen:</i> Selektion ändern & Liste schliessen	Liste öffnen <i>Aussen:</i> - <i>Innen:</i> Selektion ändern

* nicht möglich: Virtuelle Tastatur ist nicht sichtbar, somit kann keine Interaktion stattfinden

¹ Filter: Leerschlag als normales Symbol gezählt; Liste verändern je nach Anzahl passender Werte; nicht scrollbar

Im Gegensatz zu den oben gezeigten Andriod-Browser zeigen die meisten iOS-Browser die `datalist` und die `selects` auf der selben Basis an. Der iOS-Safari als Stellvertreter zeigt den Vergleich in Tabelle 3.8 auf.

Tabelle 3.8: Vergleich Interaktion Datalist & Select in Safari (iOS, mobile)

Kriterium	Datalist	Select	Multiselect
	geschlossen	geschlossen	geschlossen
	offen	offen	offen
Buchstaben	Schreiben nicht möglich	nicht möglich nicht möglich	nicht möglich nicht möglich
Leerschlag	Schreiben nicht möglich	nicht möglich nicht möglich	nicht möglich nicht möglich
Backspace	Löschen nicht möglich	nicht möglich nicht möglich	nicht möglich nicht möglich
Enter	<i>in Formular:</i> senden <i>ohne:</i> - nicht möglich	nicht möglich nicht möglich	nicht möglich nicht möglich
Scroll	Fenster scrollen <i>Aussen:</i> Liste schliessen <i>Innen:</i> Liste scrollen	Fenster scrollen <i>Aussen:</i> - <i>Innen:</i> Liste scrollen	Fenster scrollen <i>Aussen:</i> Fenster scrollen <i>Innen:</i> Liste scrollen
Click	Pfeil: Liste öffnen Selektion ändern & Liste schliessen	Liste öffnen Selektion ändern & Liste schliessen	Liste öffnen <i>Aussen:</i> Liste schliessen <i>Innen:</i> Selektion ändern

* nicht möglich: Virtuelle Tastatur ist nicht sichtbar, somit kann keine Interaktion stattfinden

Wie in den Tabellen 3.6 bis 3.8 zu sehen ist, erlauben Mobilgeräte weniger Interaktionen als Desktop-Computer. Zum einen stellen die verschiedenen virtuellen Tastaturen eine geringere Auswahl an Interaktionen an. Auf der anderen Seite öffnet sich die virtuelle Tastatur nicht in jeder Situation. In Bezug auf die Komponenten Select und Datalist öffnet sich die dynamische Eingabemöglichkeit nur durch das Input der Datalist. Dies ist der Grund, wieso in den drei oben dargestellten Tabellen keine Tastatur-Interaktionen bei Selects möglich sind. Die Unterschiede bei der Datalist entstehen dadurch, dass nicht jeder mobile Browser die Liste bei Tastatur-Eingaben offen lässt bzw. öffnet. Dadurch erklären sich die unterschiedlichen Verhaltensweisen, welche in den Tabellen ersichtlich sind.

Die Touch-Bedienungen auf der Datalist weisen kaum eine Übereinstimmung auf. Das Scrollen als auch das Klicken verhalten sich bei den ausgewählten Mobile-Browsern Safari, Firefox und DuckDuckGo unterschiedlich. Das Select zeigt sich in der Interaktion mit dem Finger konsistenter.

3.2.7 Undo / Redo

Die Tastatur-Interaktionen Undo¹² und Redo¹³ verhalten sich in allen Desktop-Browsern gleich. Beim Select – Single als auch Multi – passiert nichts. Im Input-Feld mit der Datalist regieren die Aktionen auf den geschriebenen Text. Die einzige Ausnahme ist Safari, welcher das Undo / Redo auf Browserebene ausführt. Das bedeutet, die Interaktion betrifft nicht nur das fokussierte Eingabefeld, sondern alle Änderungen im Browser wie das Schliessen von Tabs.

¹²Ctrl & Z auf Windows; Cmd & Z auf Mac

¹³Ctrl & Y auf Windows; Cmd & Shift & Z auf Mac

Von den Mobilgeräten unterstützt nur das iOS-System via Schüttel-Bewegung das Undo bzw. Redo. Der iOS-Safari Browser zeigt dieselbe Reaktion wie Safari auf Desktop.

3.2.8 Fazit

Die Gegenüberstellungen der existierenden HTML-Elemente zeigen einige Gemeinsamkeiten, aber auch viele Inkonsistenzen. Da die Unterschiede ein Grund für die Entwicklung einer neuen Auswahlkomponente sind, fasst dieser Abschnitt diese nochmals zusammen. Im UI existieren über die vier Browser Edge, Firefox, Chrome und Safari als auch den Systemen Mac und Windows teilweise grosse Inkonsistenzen. Das System-Theme¹⁴ spielt für die Darstellung ebenfalls eine grosse Rolle. Hierbei spielen das betriebssystemspezifische Styling und die individuellen Implementierungen der Browser eine Rolle. Der Fokus dieses Abschnitts liegt aber auf der Interaktion, da das UI bereits im Kapitel 3.2.1 genauer beschrieben ist.

Zwischen Edge und Chrome bestehen keine gravierenden Inkonsistenzen. Dies liegt daran, dass die beiden Browser mit der selben Rendering Engine¹⁵ arbeiten. Zwischen Mac und Windows bestehen nur die systemspezifischen Unterschiede wie die zu drückenden Tasten (z.B. Home vs. fn & ←).

Firefox unterscheidet sich von Chrome in mehreren Interaktionen. Während die offene Datalist auf Firefox mit den seitlichen Pfeiltasten zwei unterschiedliche Reaktionen zeigt, reagiert sie im Chrome-Browser immer auf dieselbe Weise. Das Multiselect zeigt in den beiden Browsern bei ← und → ebenfalls inkonsistentes Verhalten. Die Enter-Taste löst auf Firefox im geschlossenen Single-Select nichts aus, während Chrome die Liste öffnet. Dafür reagiert Firefox im Multiselect auf Tab (Input verlassen), wo Chrome kein Verhalten zeigt. Die Interaktionen PageUp/-Down zeigen sich bei Datalist als auch dem Select unterschiedlich. Der eine Brower verwendet die View-Höhe als Mass zum Überspringen der Elemente und der andere das size-Attribut. Im geschlossenen Zustand des Single-Select überspringt die Selektion ebenfalls eine unterschiedliche Anzahl Elemente. Bei der Interaktion mit der Maus weist das Scrollen auf beiden geöffneten Elementen Inkonsistenzen im Verhalten auf.

Safari zeigt gegenüber den oben angesprochenen Browsern die grösste Abweichung. Einige Aktionen ändern anstelle der Selektion nur das Highlight. Zu den Chromium-basierten Browsern - wie Chrome - ist der Apple-Browser am ähnlichsten. PageUp/-Down zeigt bei den Einzelauswahl-Elementen ein komplett anderes Verhalten. Beim Single-Select ist die Reaktion dieselbe wie beim Drücken von Home/ End. Die einzige Interaktion, welche nicht von den anderen Browsern abweicht, ist das Klicken auf ein Element.

Die inkonsistente Darstellung und Funktionalität dieser Komponenten erschwert eine einheitliche Benutzererfahrung über verschiedene Plattformen hinweg. Für eine optimale User-Experience ist die Konsistenzprüfung unabdingbar. Diese Erkenntnisse legen den Grundstein für die Entwicklung einer neuen, optimierten Auswahlkomponente, die die bestehenden Probleme adressieren soll. Die bereits erstellte Länderkomponente löst die Inkonsistenzen. Details dazu folgen im nächsten Kapitel.

3.3 Länderauswahl Komponente

Alternativ zu den bekannten HTML-Elementen `datalist` und `select` existiert eine weitere, spezielle Auswahlkomponente – die Länderauswahl^[16]. Diese vereinfacht die Selektion eines Landes in einer vorgegebenen Liste. Die Komponente ist jedoch nur auf den spezifischen Anwendungsfall – der Wahl eines Landes – getestet. Das Design, die Funktionen und die Anwendung unterscheiden sich aber zu den oben genannten HTML-Elementen.

¹⁴Dark- bzw. Light-Mode; auf Systemebene oder Browserebene einstellbar

¹⁵Mehr dazu steht im Kapitel Browser & HTML Renderer

¹⁶[Marti und Burki, 2024]

3.3.1 Design

Die Länderauswahl erscheint moderner als die alten HTML-Elemente. Die Komponente ist in geschlossenem als auch geöffnetem Zustand durch einen immer sichtbaren Container mit dem gewählten Land (Abbildung 3.10) visualisiert. Ist die Länderauswahl geöffnet, erscheint ein Listencontainer mit allen Ländern (Abbildung 3.11). Der Aufbau zeigt sich in einem Spaltendesign – Kontinente links und Länder rechts.

Abbildung 3.10: Geschlossene Länderauswahl

Country	
Choose a country	
All	Afghanistan
Africa	Albania
Antarctica	Algeria
Asia	American Samoa
Europe	Andorra
North America	Angola
Oceania	Anguilla
South America	Antarctica

Abbildung 3.11: Offene Länderauswahl

Das Design lässt sich einfacher und vielfältiger als bei den Standard HTML-Elementen anpassen. Die einzelnen Subkomponenten besitzen Klassen, welche ein einfaches Umstyling ermöglicht. Funktionell zeigen sich Übereinstimmungen als auch Unterschiede.

3.3.2 Funktionen & Interaktionen

Ein grosser Unterschied findet sich in der Funktion des Kontinents. Durch die Selektion eines Kontinents reduzieren sich die zur Auswahl stehenden Länder. Ist kein spezifischer Kontinent ausgewählt, fällt die Selektion auf *All*. Die Komponente besitzt kein integriertes Formular-Feld und benötigt deswegen in dieser Anwendung zusätzlichen Aufwand.

Die Länderauswahl bietet viele Interaktionen, welche sich bei den HTML-Elementen wiederfinden. Die folgende Tabelle 3.9 zeigt die Interaktionen, welche auf der Komponente implementiert sind.

In der Bedienung existieren keine Browser-Inkonsistenzen. Die Interaktionen lehnen mehr an denen des Selects als denen der Datalist an. Zudem betreffen die Änderungen in der Länder-Spalte nur das Highlight¹⁷ und nicht die Selektion. Die Selektion benötigt eine Bestätigung mit Enter oder Space.

3.3.3 Anwendung

Diese Komponente benötigt in der Anwendung kein HTML. Die Einbindung findet via JavaScript statt. Der folgende Code 3.5 zeigt einen möglichen Anwendungsfall.

Code 3.5: Länderauswahl Beispiel

```
1 const container = document.querySelector("#inputContainer");
2 if (null != container) {
3     const countryList = [
```

¹⁷Die Länderauswahl unterscheidet nicht zwischen Highlight und Cursor Position.

Tabelle 3.9: Aktionen bei der Länderauswahl Komponente

Kriterium	geschlossen	offen
↑ / ↓	↓ : Liste öffnen	<i>Kontinent</i> : Selektion ändern <i>Land</i> : Highlight ändern
← / →	-	Highlight ändern
Buchstaben	Selektion auf Suchergebnis ¹ ändern	Highlight auf Suchergebnis ¹ ändern
Leerschlag	Liste öffnen	Selektion ändern
Backspace	Selektion löschen	Selektion löschen
Esc	-	Liste schliessen
Enter	Liste öffnen	Selektion ändern
Tab	Input-Feld verlassen	Liste schliessen & Input-Feld verlassen
Scroll	Fenster scrollen	<i>Aussen</i> : Liste bleibt offen <i>Innen</i> : Liste scrollen & Highlight ändern
Hover	-	Highlight ändern
Click	Liste öffnen	<i>in Liste</i> : Selektion ändern <i>in Wertefeld</i> : Liste schliessen

* Änderung der Selektion bewirkt Änderung des Highlights auf den selben Wert

* Highlight und Cursor Position besitzen selben Wert; kein Unterschied

¹ Suche: Erster mit dem eingegebenen Symbol passender Wert aus der Liste, wenn Eingabe nicht passend ⇒ keine Aktion; Liste unverändert; nach Pause / Debounce-Ablauf ⇒ neue Suche

```

4      {
5          country : "Austria",
6          continent: "Europe",
7      },
8      /* more country objects */
9  ];
10     const structureDetail = {
11         value      : "",
12         placeholder: "Choose a country",
13         label      : "Country",
14         name       : "country",
15     };
16     const structureMaster = {
17         elementList   : countryList,
18         sectionElement: { continent: "All" },
19         focusObject   : {},
20     };
21     const detailController = ChoiceDetailController(structureDetail);
22     const masterController = ChoiceMasterController("continent", "country"
23             )(structureMaster);
24     const view = projectChoiceInput(detailController, masterController, "
25         countryInput");
26     container.append(...view);
27 }
```

Wie im Code 3.5 ersichtlich sind mehrere Zeilen für die Erstellung der Komponente notwendig. Zudem befindet sich die Länderauswahl noch im experimentellen Status. Wie das Unterkapitel Future Features aus dem Bericht^[18] der Vorarbeit (IP5) beschreibt, gibt es

¹⁸[Marti und Burki, 2024]

noch Verbesserungspotenzial in der Implementation. Da diese Komponente noch nicht lange existiert, ist sie noch in keinem realen Anwendungsfall im Einsatz.

3.4 Anwendungsfälle

Die Standard HTML-Elemente finden in vielen Webapplikationen ihre Anwendung. Das bekannteste Beispiel eines Selects ist die Auswahl des Geschlechts. Diese Verwendung ist abgesehen vom Design unproblematisch. Ein anderer Anwendungsfall ist die Auswahl eines Jahres oder Geburtstags mit drei Selects. Das Ausfüllen dieser Situation gestaltet sich schon unangenehmer. Dies liegt daran, dass die Suche nach dem gewünschten Wert besonders beim Jahr eher lange dauern kann. Besonders mühsam gestaltet sich für Nutzer die Suche nach dem Herkunfts- bzw. Zielland aus einer Liste von ca. 250 weltweit. Aber auch den Wohn- oder Destinationsort aus mehreren 100 bis 1'000 (je nach Anwendungsgebiet) auswählen zu müssen, ist sehr zeitaufwendig. Die Datalist mit der eingebauten Filterfunktion bietet auf den ersten Blick eine angenehmere Anwendung. Das Problem ist jedoch, dass in den meisten Fällen eigene Eingaben unerwünscht sind. Als ebenfalls schlechte Alternative bietet sich das Select als Auswahlkomponente an. Damit zieht sich die Suche nach einem bestimmten Wert in die Länge – speziell wenn die Optionen nicht in alphabetischer Reihenfolge vorliegen. Selbst die Zuhilfenahme der Select-Suche¹⁹ ist nicht in jeder Situation hilfreich. Ein weiteres Anwendungsgebiet findet sich in gewissen Webshop bei den Filter- und Sortierfunktionen wieder. Die Komponente zeigt sich unter anderem bei der Auswahl einer Grösse, Farbe oder Kategorie. In diesem Beispiel können viele Optionen bzw. Werte ohne klare Reihenfolge in einer Liste auftauchen. Dies führt dazu, dass das Auffinden des gewünschten Werts eher schwer fällt. All die oben genannten und noch weitere Fälle zeigen verschiedene Probleme auf. Die neue Komponente ermöglicht es, die unangenehmen Situationen aufzulösen. Mehr Informationen dazu ist im nachfolgenden Kapitel zu lesen.

¹⁹Durch das Drücken der Anfangsbuchstaben des gesuchten Werte springt die Selektion zur passenden Option



Kapitel 4

Neue Auswahlkomponenten

Dieses Kapitel beschreibt die Bausteine der neuen Komponente in verschiedenen Ebenen. Als erstes zeigt das Design die visuelle Darstellungsmöglichkeiten über die getroffene Auswahl bis hin zu den Prototypen. Dazu gesellen sich die Implementation der Zustände als auch die Umsetzung des Designs. Darauffolgend findet sich die Definition der Interaktion mit der Maus als auch der Tastatur. Die für einen stabilen Code wichtigen Regeln und Prinzipien sind ebenfalls festgelegt. Die Anwendung diverser Patterns führt zu einer klaren Struktur und hoher Wiederverwendbarkeit. Als spezielles Element in der View zählt der Dropdown Container, welcher sich unterschiedlich implementieren lässt. Für ein angenehmes Nutzerfeeling spielt eine gute Performance eine tragende Rolle. Dies zeigen auch die ausgeführten User Tests mit Endanwendern als auch Programmierern. Automatisierte Tests für jede Komponente beweisen eine gute Codequalität.

4.1 Design

Der Fokus liegt darauf, eine konsistente und ansprechende Benutzererfahrung zu schaffen, die sich über alle modernen Browser hinweg hält.

4.1.1 Designansatz

Das Design der neuen Auswahlkomponente ist stark vom Kolibri-Designsystem inspiriert, jedoch mit einigen Anpassungen, um die Lesbarkeit und Benutzerfreundlichkeit zu optimieren. Das Designsystem bietet bereits ein umfassendes Set von Richtlinien und Komponenten, die es ermöglichen, Anwendungen einheitlich und benutzerfreundlich zu gestalten.

4.1.2 Mögliche Designoptionen eines Elements & deren Wahl

Elemente lassen sich durch diverse Eigenschaften stylen. Die Anpassung der Hintergrundfarbe fällt am schnellsten ins Auge. Diese ist hell zu gestalten, damit der Kontrast zur dunklen Default-Schriftfarbe erhalten bleibt. Der Rahmen bietet eine weitere Designmöglichkeit. Seine Farbe, Dicke oder Struktur kann variiert werden. Eine andere Alternative ist nur eine Seite des Rahmens zu verwenden. Als Beispiel dafür gilt der sogenannte Spiegelstrich, welcher auf der linken Seite platziert ist. Für eine gute Erkennbarkeit sollte die Färbung Rund um den Rahmen in einer eher dunklen Schattierung Anwendung finden. Als weitere Style-Eigenschaften

bietet sich Änderungen der Schrift an. Bei der Farbe ist der Kontrast zu beachten, weswegen eine eher Dunkle zu wählen ist. Alternativ lässt sich die Dicke, Schriftart, Neigung, Grösse oder Dekoration ändern. All diese Style-Anpassungen lassen sich auf unterschiedliche Art und Weise kombinieren.

Bei den Farben existiert eine grosse Bandbreite. Da Kolibri bereits ein Designssystem besitzt, schränkt sich die Menge ein. Unter den vorhandenen Farbwerten, bieten sich die Folgenden am besten an:

- Kolibri-Light/Yellow/100 → helles Weiss-Gelb
- Kolibri-Light/Yellow/300 (-kb-color-select) → helles Gelb
- Kolibri-Light/Warning/-kb-warning-dark → dunkles Gelb
- Kolibri-Light/Danger/-kb-danger-accent (-kolibri-color-accent) → mittleres Rosa
- Kolibri-Light/Success/-kb-success-accent → mittleres Grün
- Kolibri-Light/Success/-kb-success-light → helles Grün
- Kolibri-Light/Primary/-kb-primary-accent → mittleres Violett
- Kolibri-Light/Primary/-kb-primary-light → helles Violett
- Kolibri-Light/Secondary/-kb-secondary-accent → mittleres Blau
- Kolibri-Light/Secondary/-kb-secondary-light → helles Blau
- Kolibri-Light/Monochrome/-kb-color-body → mittleres Grau
- Kolibri-Light/Monochrome/-kb-color-line → helles Grau

Da die Elemente drei verschiedene Zustände gleichzeitig erhalten können, müssen die Styles kombinierbar sein. Nicht jede der erwähnten Eigenschaften und Farben eignen sich in gleichem Masse. *Yellow-100* fällt schnell wieder weg, da je nach Display der Kontrast zu gering ist. Unter den restlichen hellen Schattierungen passt *Yellow-300* am besten, da diese Farbe bereits als Selektionsfarbe im Code hinterlegt ist. Als Hintergrundfarbe eignet sich am ehesten das vordefinierte `-kb-color-select`. Dadurch ist klar, dass die anderen beiden Zustände eine eher kräftige bzw. dunkle Färbung benötigen. Der Blick auf die Namen im Designssystem zeigt eine weitere Farbe zur Hervorhebung eines Elements. Die Benennung des mittleren Rosa mit Accent (`-kolibri-color-accent`) bietet einen guten Kontrast zur Selektion und ist ebenfalls im Code vordefiniert. Da die dritte Farbe nicht zu viel Unruhe in das Design bringen soll, schränkt sich die Farbauswahl weiter ein. Mit einem gut erkennbaren Kontrast zur Selektion bietet sich das dunkle Gelb `-kb-warning-dark` an.

Wie erwähnt ist die Eigenschaft Hintergrundfarbe durch die Wahl der Farbe bereits festgelegt. Die Änderung der Schrift sollte maximal die Farbe betreffen. Die anderen Font-Stylings sind schwer erkennbar oder zerstören das Bild. Der komplette Rahmen passt nicht in das Design und fällt somit ebenfalls weg. Da nicht beide Zustände auf die Schriftfarbe zugreifen können, bietet sich der oben genannte Spiegelstrich an. Das Rosa mit einem guten Kontrast zum Gelb findet sich im linksseitigen Rahmen wieder. Das dunkle Gelb färbt – nebst dem linksseitigen Strich – die Schrift. Die Zuordnung der Designwahl zu den fehlenden Zuständen steht im Kapitel **Farbpalette und Kontrast**.

4.1.3 Figma-Prototypen

Zur Visualisierung und zum interaktiven Testing des Designs kommt Figma zum Einsatz. Figma ist ein webbasiertes Tool zur Erstellung von UI/UX-Designs, das Echtzeit-Kollaboration ermöglicht. Das Sammeln der Feedbacks von Stakeholdern und Nutzern unterstützt eine effiziente Entwicklung. Dafür ist es unabdingbar, im Vorfeld mit Figma schnell Prototypen zu erstellen und mit den Probanden zu teilen.

Die folgenden Screenshots 4.1 bis 4.3 zeigen die in Figma erstellten Prototypen der Dropdown-Komponente:

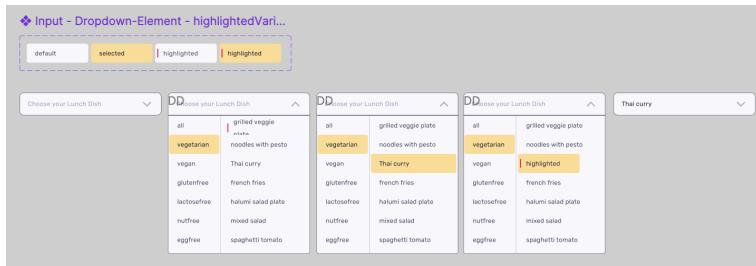


Abbildung 4.1: Figma Prototyp – Dropdown Komponente 1



Abbildung 4.2: Figma Prototyp – Dropdown Komponente 2

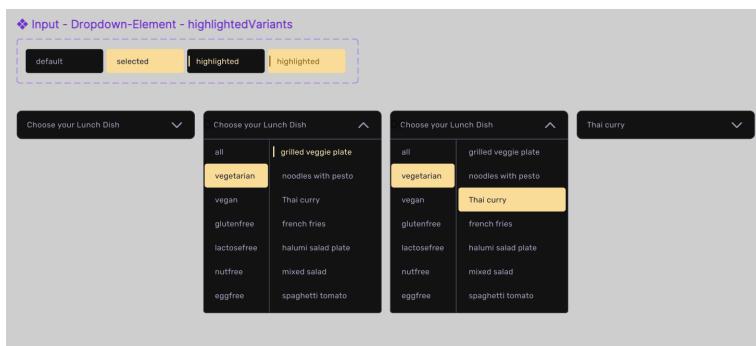


Abbildung 4.3: Figma Prototyp – Dropdown Komponente 3

4.1.4 Farbpalette und Kontrast

Das Kolibri-Designsystem bietet eine Vielzahl von Farben. Der Figma-Prototyp enthält spezifische Anpassungen, so dass die Dropdown-Komponente gut lesbar ist. Für den Erhalt einer besseren Nutzerfreundlichkeit gestaltet sich die Farbauswahl aus Abbildung 4.4. Diese bietet hohen Kontrast und verbessert somit die Barrierefreiheit der Anwendung.



Abbildung 4.4: Kolibri Design System – Farbpalette

Die Kolibri-Base Farben erhalten im neuen Dropdown eine angepasste Verwendung:

- Original → Neu
- Kolibri-Light/Yellow/300 → selected
- Kolibri-Light/Danger/-kb-danger-accent → highlighted
- Kolibri-Light/Warning/-kb-warning-dark → cursor position

Der Code 4.1 zeigt die importierten CSS-Dateien, die als Grundlage für dieses Styling dienen:

Code 4.1: CSS Imports

```
1 @import "../.../css/kolibri-base.css";
2 @import "../.../css/kolibri-light-colors.css";
```

4.1.5 Layout und Typografie

Die Umsetzung der Dropdown-Komponente des in Figma entworfenen Designs erhält eigenes CSS. Nachfolgender Codeausschnitt 4.2 zeigt den relevanten CSS-Code, der bei der Gestaltung des Popover- und Option-Styles zur Verwendung kommt:

Code 4.2: CSS für das Popover-Element

```
1 @keyframes open {
2     0% {
3         transform: scaleY(0);
4     }
5     100% {
6         transform: scaleY(1);
7     }
8 }
9 @keyframes close {
10    0% {
11        transform: scaleY(1);
12    }
}
```

```
13    100% {
14        transform: scaleY(0);
15    }
16 }
17 .options-component[popover] {
18     position: absolute;
19     z-index: 20;
20     max-height: 240px;
21     border-radius: 0 0 4px 4px;
22     border: 1px solid #ccc;
23     border-top: none;
24     background: #fff;
25     overflow: hidden;
26     align-items: stretch;
27     flex-wrap: nowrap;
28     padding: 0;
29     margin: 0;
30     box-shadow: 0px 5px 15px #0002;
31     animation: open 300ms ease-in-out;
32     transform-origin: top center;
33 }
34 .options-component[popover]:popover-open {
35     display: flex;
36     height: fit-content;
37 }
```

Diese CSS-Regeln definieren die Animationen für das Öffnen und Schliessen des Popovers und das grundlegende Styling des Popover-Containers. Die `@keyframes` regeln die Transformation, während `.options-component[popover]` das Layout und das Erscheinungsbild des Popovers steuert.

4.1.6 Implementation der Zustände

Die Dropdown-Komponente ist so gestaltet, dass sie sowohl für Maus- als auch Tastaturlbenutzer optimal funktioniert. Das Design der Interaktionen bietet eine intuitive und leicht zugängliche Bedienung der Komponente. Spezifische CSS-Klassen erleichtern die Benutzerführung. Sie definieren die Styles der hervorgehobenen (highlighted) bzw. ausgewählten (selected) Optionen. Der CSS-Code 4.3 zeigt einen Style-Ausschnitt auf ein aktuelles¹ und selektiertes Element.

Code 4.3: Aktuelle (`cursor-position`) und ausgewählte (`selected`) Option

```
1 .cursor-position {
2     color: var(--kb-hsla-warning-dark);
3
4     &:not(.disabled)::before {
5         content: '>';
6         position: absolute;
7         left: 7px;
8         top: 0.5em;
9         bottom: 0.4em;
10        transform: translateX(-50%);
11
12        background: var(--kb-hsla-warning-dark);
13        border-radius: 1px;
```

¹Element, welches sich unter der Cursor Position befindet

```

14         width:      2px;
15     }
16 }
17 .selected {
18     background: var(--kolibri-color-select);
19     border-radius: 4px;
20 }

```

cursor-position: Definiert die Hervorhebung der Option, die gerade durch den Cursor² fokussiert ist. Eine dunkle Farbe und eine linksseitige Markierung betonen das Element.

selected: Stellt die ausgewählte Option durch einen speziellen Hintergrund und abgerundete Ecken dar.

Im `columnOptionsProjector.js` gibt es Event-Handler, die das Setzen und Entfernen der Highlighting-Klassen steuern:

Code 4.4: Steuerung der Highlighting-Klassen

```

1 const selectOptionItem = (root) => (newOption, oldOption) => {
2   const oldItem = getHtmlElementByOption(oldOption, root);
3   if (oldItem) {
4     oldItem.classList.remove("selected");
5   }
6
7   const newItem = getHtmlElementByOption(newOption, root);
8   if (newItem) {
9     newItem.classList.add("selected");
10  }
11 };
12 const cursorPositionItem = (root) => (newOption, oldOption) => {
13   // same structure as selectOptionItem for the cursor-position class
14 };

```

selectOptionItem: Verschiebt die Klasse `.selected` und somit die Hervorhebung auf das neue Element.

cursorPositionItem: Entfernt bzw. setzt die Klasse `.cursor-position` – ähnlich wie beim `selectOptionItem`.

Im Codeausschnitt 4.5 aus der Datei `selectProjector.js` lässt sich das Highlighting durch die Verknüpfung der Optionen und ihrer Events mit den entsprechenden CSS-Klassen steuern.

Code 4.5: Event-Handling für die Auswahl einer Option

```

1 const projectSelectedValueOptionView = (selectController, popoverElement)
2   => {
3   const rootElement = document.createElement("div");
4   rootElement.id    = selectController.getId() + "-selected-option";
5   rootElement.classList.add("selected-option-component");
6   rootElement.setAttribute("data-id", selectController.getId());
7
8   const selectedOptionContainer = document.createElement("div");
9   selectedOptionContainer.innerHTML =
10    selectController.getSelectedValueOption().getLabel();
11   selectedOptionContainer.onclick = togglePopover;
12   selectedOptionContainer.classList.add("selected-value");
13
14   rootElement.append(selectedOptionContainer);

```

²Tastatur-Position-Anker (nicht Maus-Cursor)



```
13     return [rootElement, selectedOptionContainer];
14 }
```

`projectSelectedValueOptionView`: Erstellt die Ansicht für die ausgewählte Option und verknüpft die Klick-Events mit dem Umschalten des Popovers. Diese Kombination von CSS und JavaScript bietet dem Highlighting und der Auswahl von Optionen in der Dropdown-Komponente eine effektive und visuell ansprechende Umsetzung.

4.1.7 Optionen und Scrollbar-Styling

Diese CSS-Regeln gestalten die Optionen einer einzelnen Spalte sowie das Scroll-Verhalten des Containers. Das `.options-column` aus Code 4.6 definiert das Layout sowie weitere listen-abhängige Eigenschaften, während das `.option-item` das Erscheinungsbild und die Interaktivität der einzelnen Optionen festlegt.

Code 4.6: Optionen und Scrollbar-Styling

```
1 .options-column {
2   position:      relative;
3   width:        100%;
4   overflow-y:    scroll;
5   overflow-x:    hidden;
6   max-height:   240px;
7   min-height:   100%;
8   padding:       5px;
9   flex-grow:    2;
10  flex-shrink:  1;
11 }
12 .option-item {
13   position:      relative;
14   padding:       10px 20px;
15   display:       block;
16   cursor:        pointer;
17   width:        100%;
18   line-height:  1.2;
19   overflow:     hidden;
20   text-overflow: ellipsis;
21 }
```

Die Nutzung von den CSS-Styles im JavaScript stellt sicher, dass die dynamischen Klassen und Stile bei den Elementen bei Erstellung oder Aktualisierung ihre Anwendung finden. Das Beispiel 4.7 aus den Dateien `selectProjector.js` und `columnOptionsProjector.js` verwenden die erwähnten CSS-Klassen:

Code 4.7: CSS-Styles im JavaScript

```
1 // selectProjector.js
2 const projectOptionsView = (selectController) => {
3   const optionsContainer = document.createElement("div");
4   optionsContainer.id     = selectController.getId() + "-options";
5   optionsContainer.classList.add(optionsClassName);
6   optionsContainer.setAttribute("popover", "auto");
7   // more popover logic
8 };
9
10 // columnOptionsProjector.js
```

```

11 const projectOption = (selectedOptionController, option, optionType,
12   cursorPositionController) => {
13   const item = document.createElement("div");
14   item.innerHTML = option.getLabel();
15   item.setAttribute("data-id", elementId(option));
16   item.setAttribute("data-value", option.getValue());
17   item.setAttribute("data-label", elementDataLabel(option));
18   item.classList.add(optionClassName);
19   item.classList.add(optionType + " - " + optionClassName);
20   // more option logic
21 };

```

Dieser JavaScript Code erstellt die HTML-Elemente und ergänzt die CSS-Klassen, um das Styling und die Funktionalität sicherzustellen. Das `selectProjector.js` erstellt und stylt den Popover-Container, während das `columnOptionsProjector.js` die einzelnen Options-Elemente gestaltet. Diese Kombination aus CSS und JS stellt sicher, dass die Dropdown- und Popover-Komponenten korrekt angezeigt und interaktiv sind.

4.1.8 Prototyping und Benutzerfeedback

Der Einsatz von interaktiven Figma-Prototypen ist hilfreich beim Evaluieren der initialen Benutzerfreundlichkeit und intuitiven Bedienung. Mit der Integration der Rückmeldungen von Benutzerinteraktionen in das Design verbessert sich die Usability kontinuierlich. Beispiele für die Prototypen und die verschiedenen Zustände der Dropdown-Komponente sind in der oberen Hälfte des Bildes 4.1 ersichtlich.

Die Gestaltung der Auswahlkomponente umfasst sowohl visuelle als auch funktionale Aspekte. Gezielte CSS-Anpassungen und ein durchdachtes Interaktionsdesign finden in der Realisierung ihren Platz. Das Ziel ist, eine ansprechende und benutzerfreundliche Komponente zu schaffen. Sie fügt sich nahtlos in das Kolibri-Designsystem ein und überzeugt gleichzeitig durch optimierte Les- und Bedienbarkeit.

4.1.9 Implementationsresultat

Die diversen oben beschriebenen Design-Schritte resultieren in den Abbildungen 4.5 bis 4.10. Weitere Bilder befinden sich im Anhang B.



Abbildung 4.5:
Geschlossene
SelectComponent

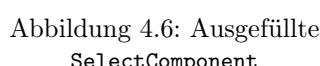


Abbildung 4.6: Ausgefüllte
SelectComponent

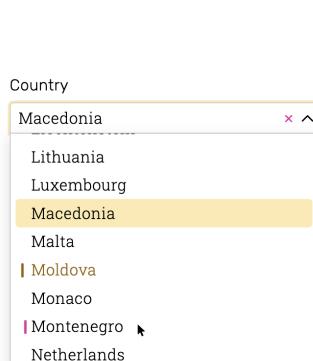


Abbildung 4.7: Offene
SelectComponent – 3 Zustände

Year	
1940's	2000
1950's	2001
1960's	2002
1970's	2003
1980's	2004
1990's	2005
2000's	2006
	2007

Abbildung 4.8: Offene
SelectComponent – 2
Spalten

Im geschlossenen Zustand (Abbildungen 4.5) zeigt der Pfeil auf der rechten Seite nach unten. Das Icon ändert beim Öffnen der neuen Komponente die Richtung und zeigt nach oben (Abbildungen 4.7). Die selektierte Option kann wie in der Grafik 4.6 ein Bild enthalten. Der Fokus auf der `SelectComponent` ist durch einen gelben Rahmen um die Detail-Ansicht visualisiert. Dies ist in Abbildung 4.6 im oberen Bereich ersichtlich. Ist das Feld `required`, erhält der Rahmen einen rosa-farbenen Rahmen. Dabei ist der Fokus aber nicht überdeckt. Die `disabled SelectComponent` zeigt sich wie die anderen Eingabefelder ebenfalls ausgegraut.

Die offene Komponente stellt den Options-Container in jedem Fall unterhalb dar. Die Abbildung 4.7 enthält – nebst normal – die drei aktuell möglichen Zustände der einzelnen Optionen. Der Wert *Macedonia* visualisiert die Selektion, welche sich ebenfalls im oberhalb befindlichen Detail-Container wiederfindet. *Moldova* zeigt die Optionen mit der Cursor Position und somit die Stelle, an welcher sich die Tastatur befindet. Das Highlight enthält den Wert *Montenegro*. Die Cursor Position und das Highlight können auf der gemeinsamen Option liegen. Dabei sind beide Zustände sichtbar.

Die Abbildungen 4.8 bis 4.10 zeigen, dass die `SelectComponent` mehrere Spalten besitzen kann.



Abbildung 4.9: Offene `SelectComponent` mit Bildern

City		
Europe	Canada	L.A.
North America	United States	New York
Asia		Ottawa
		Washington DC

Abbildung 4.10: Offene `SelectComponent` – 3 Spalten

In diesen Beispielen befindet sich die Value-Option³ jeweils in der rechten Spalte. Die anderen Spalten dienen zur Filterung der Werte. Die Grafik 4.9 mit der Selektion *2000's* begrenzt die Jahreszahlen von ursprünglich 70 auf noch 10 Werte. Wenn mehrere Kategorien existieren, bewirkt eine Selektion ganz links eine Reduktion aller Optionen die auf der rechten Seite liegen. In der Abbildung 4.10 bewirkt die Wahl von *North America* die Reduktion der Länder und der Städte.

4.2 Interaktionen

Damit ein gemeinsames Verständnis entsteht, gilt es für die Bedienung der Komponente Regeln festzulegen. Wie in den Grundlagen bereits beschrieben kann sich ein Wert aus dem Optionen-Container in verschiedenen Zuständen befinden. In diesem Absatz spielen Selektion, Highlight und Cursor Position eine Rolle. Zur Auffrischung:

- **Selektion:** Ausgewählter Wert der Spalte
- **Highlight:** Element unterhalb des Mauszeigers
- **Cursor Position:** Position (Element) der Tastatur

³Formular-Wert, wenn einer selektiert ist

Bei der Festlegung der Maus-Interaktion fällt die Entscheidung auf folgendes:

- **mouseover:** visuelles Highlighting des Elements ohne Selektionsänderung
- **click:** Änderung der Cursor Position & Selektionsänderung

Die Tastatur-Steuerung mit den Pfeiltasten hingegen hält sich an diese Bedienungen:

- Änderung der Cursor Position
- keine Selektionsänderung

Als Basis für den ersten Projektor der neuen Komponente ergeben sich aus den oben genannten Regeln folgende Interaktionen (Tabelle 4.1).

Tabelle 4.1: Aktionen bei der ersten Version der neuen Komponente

Kriterium	geschlossen	offen
↑ / ↓	Selektion ändern	Cursor Position ändern
← / →	-	Cursor Position ändern
Buchstaben	Selektion auf Suchergebnis ¹ ändern	Cursor Position auf Suchergebnis ¹ ändern
Leerschlag	Liste öffnen	Selektion ändern
Backspace	Selektion löschen	Selektion löschen
Delete	Selektion löschen	Selektion löschen
Esc	-	Liste schliessen
Enter	-	Selektion ändern
Tab	Input-Feld verlassen	Liste schliessen & Input-Feld verlassen
PageUp / PageDown	Fenster scrollen	Cursor Position auf jeden 10. Wert ändern
Home / End	Selektion auf ersten/letzten Wert ändern	Cursor Position auf ersten/letzten Wert ändern
Scroll	Fenster scrollen	Aussen: Liste bleibt offen Innen: Liste scrollen & Highlight ändern
Hover	-	Highlight ändern
Click	Liste öffnen	in Liste: Selektion ändern in Wertefeld: Liste schliessen

* Änderung der Selektion bewirkt Änderung der Cursor Position auf den selben Wert

¹ Suche: Erster mit dem eingegebenen Symbol passender Wert aus der Liste, wenn Eingabe nicht passend ⇒ nächster nachfolgender Wert; Liste unverändert; nach jedem Symbol ⇒ neue Suche

Das Undo und das Redo auf der Komponente erhält im ersten Projektor keine spezielle Definition. Gewisse Verhaltensweisen finden sich im geschlossenen als auch offenen Zustand der Komponente wieder. Anders als bei den existierenden Komponenten, ist bei der Neuen die Leertaste neu belegt. Ist die Liste bereits offen, selektiert diese Interaktion den sich aktuell unter der Cursor Position befindliche Wert. Andere Projektoren können eigene Interaktionen definieren.

4.3 Prinzipien & Regeln

Diverse Prinzipien garantieren einen stabilen und verständlichen Code. Ein Ansatz ist, alle Objekte so immutable als möglich zu halten. Dadurch lassen sich unerwartete Änderungen

verhindern. Weiter gilt es, die Bestandteile im KISS-Stil umzusetzen. Dazu zählt, dass die einzelnen Objekte und Funktionen möglichst privat zu gestalten sind. Die Bausteine sind kurz und übersichtlich aufzubauen. Zu diesem Zweck soll Separation of Concern zum Einsatz kommen, so dass jede Funktion nur eine Aufgabe zu erfüllen hat. Damit der Code einfach und lesbar bleibt bzw. wird, gilt es, Entscheidungen zu treffen. Zu diesen Entschlüsse zählt das bewusste Weglassen von Funktionalität und somit auch Komplexität.

Beim Implementieren ist darauf zu achten, den Code sauber zu formatieren. Zudem ist es sinnvoll, die Änderungen regelmässig mit dem Code-Analyse-Tool von IntelliJ auf ihre Qualität zu prüfen. Diese Prinzipien und Regeln unterstützen eine ordentliche Entwicklungsumgebung für eine stabile Komponente. Das Kapitel Patterns bietet eine weitere Möglichkeit den Code strukturiert zu halten.

4.4 Patterns

In diesem Projekt finden sich einige Code-Patterns wieder. Die Wichtigsten wie Null-Object, Projector und Decorator sind in den nachfolgenden Unterkapitel genauer erläutert. Eine weitere Rolle spielt unter anderem die Master-Detail-View, aber im Zusammenhang mit der Komponente eher nebensächlich. Zudem ist die Anwendung nicht typisch bzw. genau abgegrenzt. Die Implementation erhält durch die verwendeten Patterns eine Struktur und läuft stabiler.

4.4.1 Null Object Pattern

Ein Pattern, welches im Verlauf der Arbeit eine wichtige Rolle eingenommen hat, ist das Null-Object Pattern^[4]. Null hat den Nachteil, dass alle Funktionsaufrufe darauf zu Fehlern führen. Das Null-Object besteht aus vordefinierten Default-Werten und besitzt für alle Funktionen eine *Do-Nothing*-Implementation. Durch die Verwendung dieses speziellen Objekts entfällt eine ansonsten notwendige Nullwertprüfung. Zudem ist jedes erstellte Null-Object wertegleich.

Eine Null-Option ist notwendig, um eine Selektion zurücksetzen zu können. Die Verwendung des Null-Objects findet sich an mehreren Stellen des Codes wieder. Die Definition der angewendeten Null-Option zeigt der nachfolgende Code.

Code 4.8: Null-Option Definition

```
1  /** @private @returns { OptionType } */
2  const reset = () => {
3      return Option(null, null);
4  };
5
6  /** @public @type { OptionType } */
7  const nullOption = reset();
```

Mit dem Erhalt des Typs `Option` bietet die Konstante die selbe Funktionalität wie die gewünschten Objekte. Der Codeausschnitt 4.8 befindet sich in der Datei `optionsModel.js`. Mehr zur File Aufteilung ist in den nächsten zwei Unterkapiteln zu lesen.

⁴[GeeksforGeeks, 2024]

4.4.2 Projector Pattern

Das Projector Pattern^[5] basiert auf dem verbreiteten Model-View-Controll Pattern. Das Model verwaltet die dargestellten Daten. Zudem enthält die Komponente des Patterns die Geschäftslogik und verarbeitet die Regeln und Anfragen für die Daten. Ein Controller generiert privat gehaltene Modelle. Dabei stellt dieser nur die notwendigen Funktionen zur Verfügung. Diese Funktionen können Getter, Setter und Listener der observierten Modelle und Werte sein. Der Projektor bindet Daten-Modelle über den Controller an die View. Auf der anderen Seite bindet sich die View an die Models. Aus den Bindings und den Daten generiert ein Projektor die passende View. Die View ist passiv und hat keine Kenntnis über die anderen Komponenten.

Dieses Pattern zeigte sich als eines der Wichtigsten für die Erstellung der neuen Komponente. In den folgenden Grafiken sind Models als Zylinder, Controller als schiefes Rechteck und Projectors als Oval dargestellt. Die Raute mit Option ist ein Daten-Typ, der über das gesamte Projekt seine Anwendung findet. Das `starter.js` beinhaltet alle Bestandteile, welche für eine Anwendung notwendig sind. Eine genauere Beschreibung des Puzzles folgt im Unterkapitel **Decorator Pattern**. Die erste Implementation, welche dieses Pattern verwendet, ist auf Abbildung 4.11.

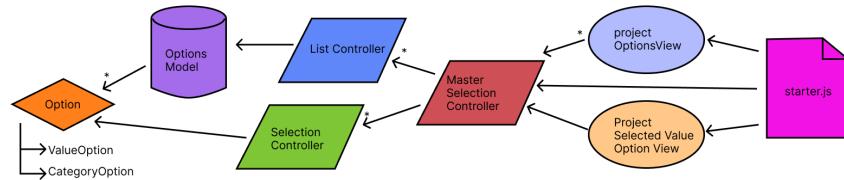


Abbildung 4.11: Diagramm Select Component – erste Version

Diese Version zeigt noch viel Komplexität und duplizierenden Code in den einzelnen Funktionen. Eine genaue Analyse der Komponente zeigt, dass sich das Pattern zwei Mal anwenden lässt. Die neue Aufteilung ergibt die zwei folgenden Abbildungen 4.12 und 4.13. Die Implementation der dargestellten Diagramme resultiert aus einem Refactoring im grösseren Rahmen.

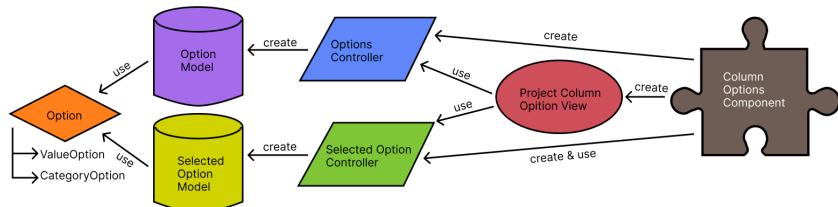


Abbildung 4.12: Diagramm ColumnComponent

Zum einen findet sich das Projector Pattern in einer einzelnen Spalte in der Options-Liste wieder. Pro Kolonne existiert eine Auswahl und eine Menge von Optionen. Diese beiden Be-

⁵ [König, 2024]

standteile besitzen je ein eigenes Model und einen eigenen Controller. Der Projektor generiert eine gemeinsame View und bindet diese an die beiden Controller. Bei einer Anwendung übernimmt die `ColumnComponent` die Verwaltung gewisser Bausteine (Mehr dazu später).

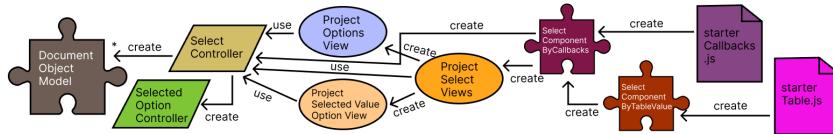


Abbildung 4.13: Diagramm `SelectComponent`

Die Anwendungskomponente einer Column findet sich als Bestandteil des zweiten Projektor Pattern wieder. Ein `SelectController` verwaltet eine bis mehrere `ColumnComponents` als auch ein Element für die Tastaturnavigation. Die sogenannte Cursor Position verwendet im Hintergrund ebenfalls einen `SelectedOptionController`. Dieser ist derselbe wie in der Abbildung 4.12 und findet hier eine Wiederverwendung. Für die Definition der Bindings greifen die einzelnen Projektoren auf den selben Controller zu. Der Master-Detail-Aufbau der neuen Komponente findet sich in der Aufteilung der Projektoren wieder. Der Detail-Baustein kümmert sich um die aktuelle Auswahl und das Eingabefeld. Die Master-Komponente verwaltet alle Spalten mit den Kategorie- und Werte-Optionen, sowie dessen Bindings. Eine weitere Funktion ist die Einbettung der beiden Projektoren in eine gemeinsame View. Auch diese Projector Pattern Anwendung schliesst mit einer Component – der `SelectComponent` – ab. Mehr zu dieser und der `ColumnComponent` steht im nächsten Kapitel.

4.4.3 Decorator Pattern

Ein Decorator^[6] bietet zusätzliches Verhalten ohne das Originale Objekt zu verändern. Dabei lassen sich verschiedene Funktionen kombinieren. Dieses Pattern ermöglicht die Erstellung eines modularen und anpassbaren Codes. In diesem Projekt unterstützt es die Gestaltung und Erweiterung der Auswahlkomponente.

Wie im vorherigen Kapitel erwähnt, besteht die neue Komponente unter anderem aus zwei sogenannten Component-Bausteinen. Die Decorator sind in den Abbildungen 4.12 und 4.13 als Puzzle dargestellt. Diese Bestandteile kombinieren die Funktionalität des Controllers mit der Erstellung der View. Dadurch lässt sich die neue Komponente einfacher anwenden.

Code 4.9: `SelectComponentByTableValues` dekoriert `SelectComponentByCallbacks`

```

1 const SelectComponentByTableValues = (
2   selectAttributes,
3   optionsTable,
4   sortColumnOptionsAlphabetical = false
5 ) => {
6   /* code for mapping between table and callbacks */
7   const component = SelectComponentByCallbacks(selectAttributes,
8     callbacks);
8   return {
9     ...component,
10   };
11 };

```

⁶[Kumar, 2024]

Ein weiterer Einsatzort ist in der Abbildung 4.13 aus dem Vorkapitel zu sehen. Der `SelectComponentByTableValues` in Code 4.9 dekoriert die `SelectComponentByCallbacks`. Damit bietet die neue Komponente zwei verschiedene Möglichkeiten der Anwendung. Das nächste Kapitel geht genauer auf den Master-View-Bereich des `SelectProjectors` – Abbildung 4.13 – ein.

4.5 Dropdown-Container

Für die Darstellung aller Optionen (zu gegebener Zeit) stehen verschiedene Varianten zur Auswahl. Eine Möglichkeit ist, den Container als HTML-Dialog zu gestalten. Die vorhandenen Funktionen sind jedoch nicht für diese Komponente geeignet. Für den gewünschten Zweck erfordert das Dialog-Element noch einiges an benutzerdefinierter Anpassung.

Eine weitere Variante ist, ein normales `div` als Options-Container zu verwenden. Dies erfordert ebenfalls einen enormen Implementationsaufwand. Eine Anwendung dieses Ansatzes findet sich in der ersten Version der Komponente. Hierbei eröffnet sich das Problem von der Inkonsistenz zwischen UI und Controller. Zudem ist es möglich, dass unerwünscht mehrere Dropdown-Container gleichzeitig offen sind.

Als dritte Möglichkeit bietet sich die Popover-API an. Seit 2024 unterstützen alle gängigen Browser diese Schnittstelle. Durch das Refactoring der Variante mit dem normalen Div-Container resultiert eine Version mit der Anwendung dieser API. Im Gegensatz zu den beiden oben erwähnten Container-Implementationen reduziert sich durch diese Schnittstelle der Zusatzaufwand. Der Grundaufbau des Popover-Container ist im folgenden Code 4.10 dargestellt.

Code 4.10: Popover-Container Beispiel

```
1 <div popover="auto"
2   id="select-component-0-options"
3   class="options-component"
4 > <!-- content --> </div>
```

Bei diesem Codeausschnitt ist wichtig, dass das Attribut `popover` den Wert `auto` erhält. Dies bewirkt, dass die Popover sich automatisch schliessen, wenn ein Klick ausserhalb des Container passiert. Das Öffnen und Schliessen des Dropdown-Elements lässt sich über das `popovertarget`-Attribut auf der Bedienkomponente steuern. Dieses Target enthält die `id` des Div-Containers mit dem Attribut `popover`. Als Alternative dazu besteht die Möglichkeit, das Popover über JavaScript zu steuern. Hierbei besteht die Möglichkeit auf den Status und das Event des Toggelns zuzugreifen. Diese Funktionalitäten erlauben den Controller konsistent zum UI zu halten.

Obwohl nur die Unterstützung der aktuellen Browser verlangt ist, soll die Komponente ältere Browser-Versionen nicht komplett ignorieren. Erst seit April 2024 unterstützen alle Browser die Popover-API. Ein Test auf einem veralteten Firefox zeigt, dass die Komponente wegen der Popover-Funktionalität einen Fehler in die Console wirft. Dadurch visualisiert der Browser die Komponente nicht. Der folgende Code 4.11 fängt die Fehlermeldung ab.

Code 4.11: Workaround für ältere Browser

```
1 try {
2   /* code block with
3    * 'popoverContainer.showPopover();'
4    * or 'popoverContainer.hidePopover();'
```

```

5   */
6 } catch (e) {
7   /* inform user about unsupported popover
8    * the first time an alert popup appears */
9 }

```

Ein Alert-Popup informiert den Nutzer über Einschränkungen. Einige zusätzliche CSS-Definitionen garantieren eine halbwegs ansehnliche Darstellung. Alle Browser, welche das nested CSS⁷ nicht unterstützen, zeigen die neue Komponente teilweise unsauber an.

4.6 Performance

Mit der Kenntnis des Rendering Prozesses einer Webseite lässt sich eine gute Performance umsetzen. Dieser Ablauf ist im Kapitel **Hintergrund** unter **Rendering Prozess** genau beschrieben. Die folgende Abbildung 4.14 zeigt die Kernelemente des Prozesses nochmals im Überblick.

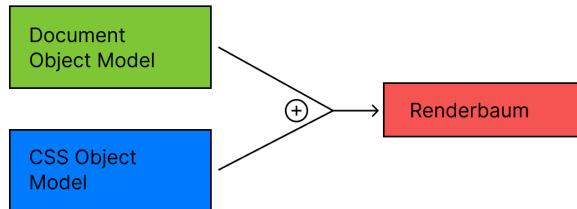


Abbildung 4.14: Rendering Prozess

Zur Auffrischung die wichtigsten Punkte nochmals: Der Browser kann die Webseite maximal 60 Mal pro Sekunde neu zeichnen. Änderungen am DOM (Abbildung 4.14) lösen das Rendern aus.

Deswegen müssen viele kleine Änderungen ausserhalb des Browser-DOMs⁸ – am besten in einem sogenannten Shadow-DOM – geschehen. Ein Shadow-DOM ist ein Teilbaum, welcher nicht am Browser-DOM angehängt ist. Für einen reibungslosen Ablauf ist es sinnvoll, die Änderungen nach dem Abhängen des Elternknotens zu vollziehen. Nach den Änderungen ist der Teilbaum wieder an den gewünschten Ort einzuhängen.

Code 4.12: Performance Optimierung (aus `columnOptionsComponent.js`)

```

1 const addAllOptions = (options) => {
2   const placeHolder = createHolder();
3   columnView.replaceWith(placeHolder);
4   if (options.length > 50) {
5     // same work as else but async
6   } else {
7     options.forEach((option) => {
8       optionsController.addOption(option);
9     });
10    updateScrollbar(columnView);
11    placeHolder.replaceWith(columnView);
12  }
13};

```

⁷Verschachtelter CSS-Code mit Verwendung der &-Funktionalität

⁸Im Renderbaum verwendet und im Browser angezeigter DOM

Code 4.12 ist eine Stelle, die diese Technik verwendet. Das Anzeigen eines Platzhalters mit einem Lade-Indikator an der Ursprungsstelle bewirkt, dass der Nutzer ein Feedback erhält. Sobald der SpaltenContainer abgekoppelt ist, lädt die Funktion die Optionen in den Shadow-DOM. Nach Abschluss ersetzt sich der Lade-Indikator der originalen Stelle im Renderbaum durch den Container mit den neuen Elementen.

Weiter ist darauf achten, dass CSS-Klassen¹⁰ an Stelle von Inline-Styles¹¹ ihre Verwendung finden. Die Selektoren sollten hierarchisch möglichst flach und nicht verschachtelt sein. Wenn es die Situation erlaubt, ist es besser, nicht mit `innerHTML` zu arbeiten. In diesem Projekt ist es für die Anzeige der Label jedoch nötig `innerHTML` zu nutzen. Dies liegt daran, dass ein Label auch ein Bild enthalten kann. Generell verwendet die neue Auswahlkomponente keine Inline-Styles. Die einzige Ausnahme betrifft das verborgene Eingabefeld. Durch den Code 4.13 sind die Properties vor dem einfachen Überschreiben¹¹ geschützt.

Code 4.13: Inline-Style für Input-Feld

```

1  inputElement.setAttribute(
2    "style", "all:           unset      !important; " +
3    "z-index:          -1         !important; " +
4    "position:         absolute   !important; " +
5    "inset:            5px        !important; " +
6    "color:            transparent !important; " +
7    "pointer-events:  none       !important; "
8  );

```

Die Regeln in Code 4.13 sorgen dafür, dass das Input-Feld transparent als auch resetet ist. Zudem befindet es sich im Hintergrund und besitzt dieselben Grösse wie der Container mit dem ausgewählten Wert.

4.6.1 Performance Vergleich

Durch die Anpassungen der Performance Optimierung, verschnellert sich die Ladezeit bei grossen Datenmenge enorm. Die Testseite enthält vier existierende und vier neue Auswahlkomponenten mit den selben Inhalten wie je eines der Existierenden. Je eine der Selects enthält eine grosse Datenmenge von über 4'000 Werten. Die folgenden zwei Bilder 4.15 und 4.16 zeigen die Messung während des Seitenaufbaus.

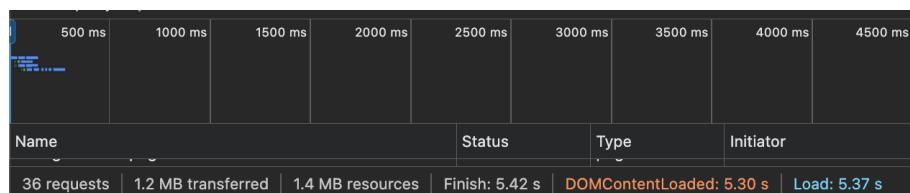


Abbildung 4.15: Performance Test vor Anpassungen

Auf der Grafik 4.15 ist zu sehen, dass der Seitenaufbau der früheren Version mit über 5 Sekunden (rechts unten in blau) sehr lange dauerte. Die Implementation führte sehr viele Aktionen auf dem Renderbaum aus. Die lange Wahrzezeit bestätigten mehrere Feedbacks der Nutzer, später mehr dazu.

⁹[Ofoegbu, 2023]

¹⁰Offizieller Begriff: Element attached style ⇒ Styles direkt im HTML-Element mit Attribut `style` definiert

¹¹`!important` besser nicht verwenden

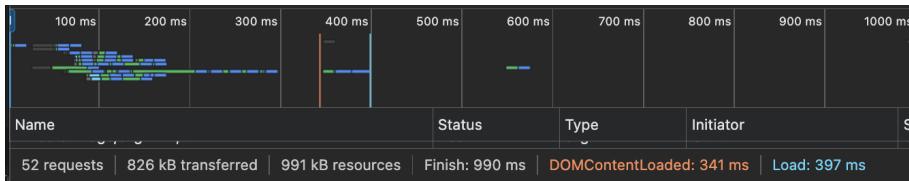


Abbildung 4.16: Performance Test nach Anpassungen

Ein Refactoring der früheren Implementation führt zur Auslagerung der aufwendigen Arbeiten auf den Shadow-DOM. Dadurch verkürzt sich der Ladevorgang um vier bis fünf Sekunden auf knapp 0.4 Sekunden (rechts unten in blau). Der direkte Vergleich zwischen Abbildung 4.15 und 4.16 zeigt, dass die Seite um Faktor 13 schneller geladen ist. Feedbacks zu den durchgeführten User-Tests mit Programmierern als auch Endnutzern sowie manuelle Tests sind im nachfolgenden Kapitel aufgeführt.

4.7 Testing

Verschiedene Arten von Tests garantieren die Korrektheit der einzelnen Funktionen und Komponenten. In manuellen Tests findet die Kontrolle der Konsistenz im UI und der Interaktion statt. Die automatisierten Code-Tests stellen sicher, dass das Aufrufen der gebotenen Funktionalität die gewünschten Änderungen ausführt. Komponenten, welche für die Erstellung des UI zuständig sind, prüfen die Existenz von Elementen als auch das Triggern der Events. Die User Tests mit unterschiedlichen Personen gewährleisten eine gute Nutzerzufriedenheit als auch die intuitive Anwendung durch Programmierer. Die Auswahl der Nutzer fällt auf eine grosse Vielfalt, damit die Resultate unterschiedliche Sichtweisen einbeziehen.

4.7.1 Manuelle Tests

Manuell durchgeführte Tests – in den vier aktuellen Browsern Edge, Chrome, Firefox und Safari – beweisen die Konsistenz der neuen Komponente. Als Beispiel dient die `SelectComponent` mit drei Spalten aus Abbildung 4.17.

City		
Europe	Canada	Berlin
North America	France	Bern
Asia	Germany	Brugg
	Japan	Hamburg
	Swiss	L.A.
	United States	Marseille
		New York
		Ottawa

Abbildung 4.17: Manuelle Tests – Komponente

Diese Komponente unterzieht sich einer Prüfung auf Mac als auch Windows. In den Einstellungen sind in einem ersten Durchgang der Light-Mode und in einem Zweiten das Dark-Theme gespeichert. Die genaue Kombination zwischen Browsern und Betriebssystemen ist wie folgt:

- **Mac bzw. OSX:** Chrome (127), Firefox (128), Safari (17.5)
- **Windows (10/11):** Chrome (127), Firefox (128), Edge (127)

Die SelectComponent aus Abbildung 4.17 zeigt sich in den aufgezählten Kombinationen grundlegend konsistent. Es existiert nur eine kleine Abweichung. Die Scrollbars auf Windows besitzen auf der rechten Seite einen grösseren Abstand als auf Mac. Dabei spielt der Browser keine Rolle. Bei den Interaktionen verhält sich die Komponente ebenfalls konsistent über alle Browser. Die einzige Abweichung findet sich in der systemabhängigen Interaktion, um eine Aktion auszuführen. Auf Windows-PCs mit Nummer-Pad funktionieren die PageUp-, PageDown-, Home- und End-Keys¹² auf der Tastatur. Auf OSX verlangen die oben genannten Tasten und Delete eine Tastenkombination mit der fn-Taste.

Zusätzlich zu den im Vorfeld vereinbarten Systemen und Browsern finden noch folgende weitere Prüfungen statt:

- **Linux (24 LTS):** Firefox (125), Chromium (127)
- **iOS (Simulator):** Safari (17.2)
- **Android (Pixel Simulator):** Chrome (124)

Diese Tests beweisen, dass die Komponente über die geforderten Umgebungen hinaus funktioniert.

Die mobilen Geräte öffnen keine virtuelle Tastatur. Deswegen sind nur Touch-Interaktionen möglich. Zu diesen Bedienungsmöglichkeiten gehören das Scrollen und das Klicken. Auf iOS und auf Android stellen die Browser das UI meist konsistent – wie auf dem Desktop Safari – dar. Das Highlight und die Cursor Position haben keinen weiteren Nutzen. Die beiden Zustände bewegen sich mit der Selektion mit. Das Highlight verschwindet, wenn das Touch ausserhalb einer Option erfolgt. Die einzige Inkonsistenz auf iOS zeigt sich beim Klick ausserhalb der SelectComponent. Anders als Safari auf Desktop schliesst diese Interaktion den Container nicht. Wnn der Klick auf der X einen bereits selektierten Wert löscht, bleibt der Container ebenfalls offen. Auf dem Pixel Chrome reagiert der Klick auf die Detail-Ansicht abweichend. Ist das Popover bereits geöffnet, schliesst die Interaktion dieses und öffnet es gleich wieder. Trifft die Touch-Geste das X bei einer bereits getroffenen Selektion, passiert dieser Fehler nicht.

Folgende veraltete Browser finden sich in oberflächlichen Tests wieder:

- **Mac:** Firefox Version 123 => vor Popover API
- **iOS:** Safari Version 16.7 => vor Popover API & partitital nesting CSS (&) Support
- **Windows:** Internet Explorer Version 11 => vor Popover API & nesting CSS (&)

Die breite Abdeckung der manuellen Tests garantiert und beweist bei den anfangs genannten Browsern eine konsistente Darstellung und einheitliche Interaktionen. Zudem sind Nutzer mit leicht veralteten Browsern nur leicht beeinträchtigt. Eine gewisse Bedienbarkeit ist möglich.

¹²Gewisse Tastaturen besitzen die genannten Tasten doppelt – einmal im Num-Pad und einmal in der obersten Zeile



4.7.2 Automatisierte Tests

Tests für Komponente der `SelectComponent` beweisen die Stabilität des Codes. Generell befinden sich die einzelnen Komponenten nach Separation of Concern je in einer eingenen Test-Einheit. Dies garantiert das schnelle Auffinden von Bugs und unerwünschtem Verhalten direkt in einer Komponente. Auf der Abbildung 4.18 ist zu sehen, dass eine breite Palette an Kontrollen stattfindet und sauber durchläuft.

All Choice Input Tests Report

7	tests in	projector/simpleForm/optionsModel	ok
28	tests in	projector/selectComponent/optionsModel	ok
9	tests in	projector/selectComponent/optionsController	ok
6	tests in	projector/selectComponent/columnOptionsProjector	ok
11	tests in	projector/selectComponent/columnOptionsComponent	ok
26	tests in	projector/selectComponent/selectController	ok
11	tests in	projector/selectComponent/selectProjector	ok
38	tests in	projector/selectComponent/selectComponent	ok
1	tests in	projector/selectComponent/selectProjector label (async)	ok
3	tests in	projector/selectComponent/selectProjector popover (async)	ok

Check console for possible errors.

Abbildung 4.18: Automatisierte Tests

Die oberen vier Zeilen auf Abbildung 4.18 zeigen Tests zu den Bausteinen aus Abbildung 4.12. Diese decken den Level der Options bis zur Column ab. Die ersten zwei Test-Suites kümmern sich genauer um die Korrektheit der Models und Controller. Dabei besteht das erste File aus den Tests der Typen `ValueOption` und `CategoryOption`. Dazu gesellen sich Prüfungen der Komponenten `OptionsModel` und `SelectedOptionModel`. Der `OptionsController` und der `SelectedOptionController` finden sich in der Test-Datei wieder. Der `projectColumnOptionsView`-Test deckt die Kontrolle des Bindings und die Existenz der View-Elemente ab. Die Korrektheit des Zusammenspiels zwischen Projekt und Controller findet sich im `ColumnOptionsComponent`-Testing wieder.

Die restlichen Test-Bibliotheken decken die Korrektheit der auf Abbildung 4.13 sichtbaren Bausteine ab. Diese drei Dateien prüfen das Zusammenspiel der Columns als auch der allgemeinen Komponenten-Funktionalitäten. In einzelnen Units prüft die `SelectController`-Test-Suite verschiedene Varianten der Anwendung. In dem `projectSelectViews`-Test befindet sich – wie bereits bei der Column – die Prüfungen der Korrektheit der Bindings und View-Elemente. Die letzte Datei kümmert sich um die Kontrolle verschiedener angewandter `SelectComponentByCallbacks` und `SelectComponentByTableValues`. Die breite Abdeckung der Einzelteile garantiert die korrekte Funktion der neuen Komponente. Die Tests im nachfolgenden Kapitel sind für eine gute Anwendbarkeit im Code als auch angenehme Benutzung der Auswahlkomponente im Browser notwendig.

4.7.3 User Tests

Programmierer

Im Rahmen der Evaluierung der neuen Dropdown-Komponente sind Programmierer dazu eingeladen, spezifische Aufgaben zu erfüllen. Diese Aufgaben umfassen die Implementierung der Dropdown-Komponente in eine bestehende Webanwendung. Für den Erhalt eines möglichst realistisches Feedbacks sind die Teilnehmer gebeten, nach der Implementierung eine Google-Umfrage auszufüllen.

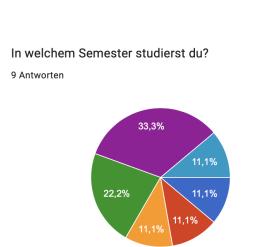


Abbildung 4.19: Verteilung Semester

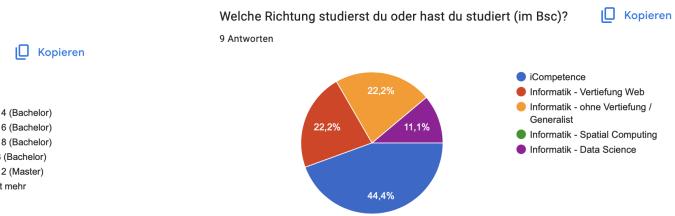


Abbildung 4.20: Verteilung Studienrichtung

Insgesamt haben 9 Studenten an dem User-Test teilgenommen und die Aufgaben versucht zu lösen. Die Grafiken 4.19 und 4.20 zeigen, dass die Hintergründe der Probanden breit gefächert ist. Zwei der Testpersonen konnten die Aufgabe nicht korrekt lösen.

Die Aufgaben beinhalten verschiedene Szenarien. Die Entwickler sollen die neue Komponente unter anderem für das Mittagessen, eine Heimatregion oder ein Geburtsjahr einsetzen. Der bereitgestellte Code (Anhang D) dient als Basis für die Implementierung. Dabei fällt der Fokus darauf, dass die Programmierer die Anbindung an die vorhandene Infrastruktur korrekt umsetzen.

Nach der systematischen Erfassung der Umfrage-Rückmeldungen erfolgt das Analysieren dieser. Programmierer bewerten die Komponente als benutzerfreundlich und ästhetisch ansprechend.

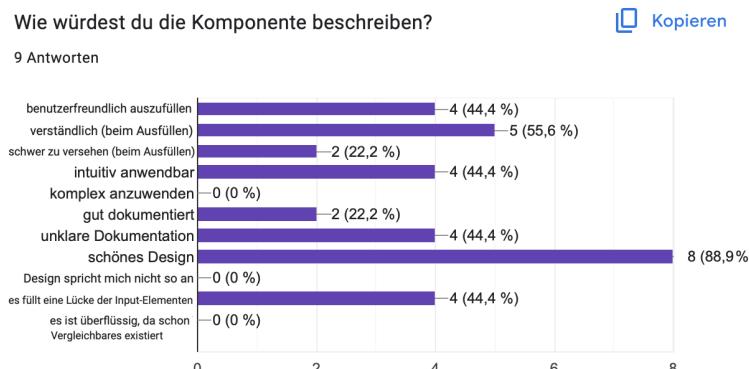


Abbildung 4.21: Meinung der Testpersonen

Die Umfrageergebnisse zeigen eine überwiegend positive Bewertung der Komponente (Abbildung 4.21). Sie sind in den Abbildungen D.1 bis D.7 im Anhang D illustriert. Einige Teil-

nehmer äussern jedoch Kritikpunkte hinsichtlich der Dokumentation. Insbesondere die Beschreibung der Rückgabewerte und die Integration von Callbacks wiesen vor der Anpassung Schwierigkeiten auf. Weitere Verbesserungsvorschläge umfassen eine bessere Spezifizierung der Typen in der JSDoc und die Möglichkeit, die Komponente nach der Auswahl automatisch zu schliessen. Es ist festzuhalten, dass Personen ohne umfassende JavaScript-Kenntnisse Schwierigkeiten haben, die Komponente intuitiv zu verwenden. Die Testperson, welche keine JavaScript-Kenntnisse vorweisen kann, konnte die Aufgaben nicht korrekt lösen. Für einige Testnutzer ist die Verwendung von Callbacks als Service-Funktionen eine Stolperfalle.

Die Ergebnisse der Tests liefern wertvolle Erkenntnisse, welche zur Verbesserung der Dokumentation beitragen. Die Überarbeitung der Dokumentation als auch der Rückgabe der Komponente behebt die eine grosse Kritik an der Komponente. Die Erstellung des zweiten Konstruktors mit einer Tabelle von Werten ermöglicht das Umgehen des Stolpersteins der Callbacks.

Formular-Ausfüller

4.8 Fazit

Das Design der Komponente, inspiriert durch das Kolibri-Designsystem, stellt sicher, dass die visuelle Konsistenz und Benutzerfreundlichkeit erhalten bleiben. Die spezifischen Designoptionen wie Farbpalette und Layout sind sorgfältig ausgewählt und implementiert. Dadurch ist eine optimale Lesbarkeit und Bedienbarkeit gewährleistet. Diese Punkte sind durch die durchgeführten Tests bewiesen.

Der Einsatz von Figma-Prototypen ist entscheidend. Daraus erfolgen eine realitätsnahe Vorschau und Benutzerfeedback. Aus diesen Gründen verbessert sich die Benutzerfreundlichkeit kontinuierlich. Die Kombination von durchdachten CSS-Anpassungen stellt sicher, dass die Komponente intuitiv bedienbar ist. Eine klare Interaktions-Definition garantiert Maus- als auch für Tastaturbenutzern eine gute Zugänglichkeit.

Die Integration von Prinzipien und Patterns führen zu einem stabilen und wartbaren Code. Mit KISS und Separation of Concern bleibt der Code einfach und gut lesbar. Die Umsetzung von dem Null Object und dem Projector Pattern garantieren die Wiederverwendbarkeit und halten sich an das Separation of Concern Prinzip. Ein wichtiger Baustein der `SelectComponent` ist der Container mit allen Werten. Die Wahl der Popover-API vereinfacht die Implementation enorm, da weniger zusätzliche Funktionalität notwendig ist.

Stetig durchgeführte, manuelle Tests ermöglichen eine Überwachung und einen Vergleich der Performance. Optimierungen im Code führen zu einer schnelleren Performance und einer angenehmen Benutzererfahrung. Automatische Tests gewährleisten eine hohe Codequalität und die Korrektheit der Funktionalitäten. Die durchgeführten User Tests bestätigten die hohe Usability der Komponente. Ein agiles Entwicklungsverfahren ermöglicht während der Entstehung der neuen, effizienten Auswahlkomponente eine stetige Optimierung. Die `SelectComponent` bildet eine solide Grundlage für zukünftige Erweiterungen und Anpassungen.

Kapitel 5

Diskussion

Die in Kapitel 3 beschriebenen Komponenten Select und Datalist kommen in vielen Webseiten zu Anwendung. Das Auswählen eines Wertes aus einer vorgegebenen Menge ist mit diesen Elementen ineffizient und unästhetisch. Die in Projekt 5 erstellte Länderauswahl löst das Problem aber nur für diesen einen spezifischen Anwendungsfall. Die neue Komponente `SelectComponent` baut auf der Länderauswahl auf, ist jedoch generalisiert und lässt sich mit unterschiedlichsten Werten anwenden. Dabei spielt es keine Rolle, welcher Konstruktor bzw. welche Art der Datenübergabe zur Anwendung kommt.

Die Datalist und das Select zeigen im UI als auch der Interaktion einige Inkonsistenzen auf. Die Darstellung lässt sich mit den wenigen Styling-Möglichkeiten der HTML-Elemente teilweise beheben. Richtig unangenehm gestaltet sich die Anwendung dieser Auswahlkomponenten bei einer grossen aber doch begrenzten Menge von Optionen. Eine saubere Integration in eine konsistent designete Webseite ist jedoch nicht möglich. Der Container mit den Werten lässt sich bei beiden Elementen nicht umgestalten und zerstört das Bild des abgestimmten Designs. Die Lösungen von Frameworks und Libraries blasen eine ansonsten schlanke Codebasis unnötig auf. Zudem bieten diese Komponenten häufig zu viele Funktionen an, welche die Anwendung verkomplizieren.

An diesem Punkt bietet die `SelectComponent` des Kolibri ohne externe Abhängigkeiten eine konsistente und anpassbare Auswahlkomponente an. Das Konsistenzproblem ist durch ein klar gestaltetes Design gelöst. Die Implementation ist auf den gängisten Browsern Edge (127), Chrome (127), Firefox (128) und Safari (17.5) auf Desktop getestet. User-Tests mit Endnutzern zeigen, dass die Komponente dessen Bedürfnisse abdecken und die Auswahl vereinfacht. Der modulare Aufbau ermöglicht eine hohe Wiederverwendbarkeit. Die Subkomponente `ColumnOptionsComponent` kann für ein Einsatzgebiet ausserhalb der Auswahlkomponente zur Anwendung kommen. Eine mögliche Verwendung kann bei einer Tabellen-Ansicht sein. Ein weiterer Vorteil der einzelnen Komponenten ist, dass andere Projektoren zur Visualisierung zum Einsatz kommen können.

Diese Arbeit ist zeitlich und personell begrenzt. Deswegen bietet die Komponente nur einen Projektor für die Auswahlkomponente. Die spät durchgeführten User Tests mit Programmierern führen dazu, dass nach der Implementation der Feedbacks keine zusätzliche Kontrolle mehr durchführbar ist. Dies ist als erstes Future Feature im nachfolgenden Kapitel genauer beschrieben.

5.1 Future Features

Die Zukunft der zwei entstandenen `SelectComponents` zeigt sich sehr vielfältig. Durch die modulare Struktur vereinfacht sich die Erweiterung, indem einzelne Komponente wie Projektoren austauschbar sind. Nachfolgend sind mögliche Verbesserungen und Features aufgelistet.

5.1.1 Weitergehende User-Tests und Nutzerbefragungen

Die aus den Feedback der Programmierer entstandene `SelectComponent` sollte sich weiteren Tests mit Entwicklern stellen. Dies garantiert eine gut verständliche Dokumentation und effiziente Einbindung in den Code. Allenfalls lassen sich weitere Bugs aufdecken. Die weiteren Tests bieten die Möglichkeit einen Vergleich zu den bereits durchgeführten Tests zu ziehen. Spezifische Fragen können die Vereinfachung der Anwendung beweisen oder widerlegen.

Die Anzahl Personen der Testgruppe massiv zu erhöhen, führt zu mehr Feedback. Gewisse Interaktionen der Nutzer können möglicherweise auf bisher unentdeckte Probleme hinweisen. Diese lassen sich in einem weiteren Schritt beheben.

Eine Umfrage bei den Endanwendern weist auf bisher unbekannte Wünsche hin. Durch eine grosse Diversität beim Hintergrundwissen der Befragten entstehen neue Designansätze. Aus den Feedbacks lassen sich neue Projektoren – ob UI oder Interaktion - designen und umsetzen.

5.1.2 Weitere UI-Projektoren

Die während des Design-Prozesses entstandenen Prototypen bieten sich als Grundlage für neue UI-Projektoren¹ an. Aus diesen Designskizzen lassen sich in Figma ausgearbeitete Prototypen erstellen. Diese finden sich in der späteren Implementation zu einzelnen Projektoren wieder.

Weitere Projektoren könnten den gleichen Aufbau besitzen, aber mit alternativen Design-Implementationen ausgestattet sein. Dabei zeigen sich die Unterschiede beispielsweise in der Visualisierung des Highlights, der Cursor Position oder der Selektion. Das Design kann jedoch wie beim Darkmode die komplette Komponente betreffen.

Statt einer Spalten-Darstellung visualisiert sich eine Idee als Zeilen-Darstellung (Abbildung 5.1). Die Kategorien zeigen initial nur die ausgewählte Option in einer Zeile an. Die `ValueOption` stellt die Werte in einer normalen Spalte dar. Beim Eintreten² in eine Kategorie ändert sich die Darstellung der aktiven Zeile in ein Rad (Mitte Abbildung 5.1). Sobald das Highlight bzw. die Cursor Position die Kategorie-Zeile verlässt, minimiert sich die Visualisierung zurück auf den selektierten Wert. Nach dem Erstellen eines Figma-Prototypen findet sich in dieser Idee ein neuer UI-Projektor wieder.

Die entstandenen als auch zukünftige UI-Projektoren lassen sich mit unterschiedlichen Interaktionen verbinden. Dabei ist es hilfreich, verschiedene Interaktions-Projektoren³ zu bieten.

5.1.3 Weitere Interaktions-Projektoren

Neue UI-Projektoren lassen sich nicht ausnahmslos mit den bestehenden Tastatur-Interaktionen kombinieren. Zudem ist nicht in jeder Situation erwünscht, dass die Cursor Position

¹ Projektoren, die nur die View generieren (ohne Tastatur-Interaktion)

² Wechsel der Cursor Position in die Kategorie oder ein Klick auf die Kategorie

³ Projektoren, die nur die Tastatur-Interaktion definieren

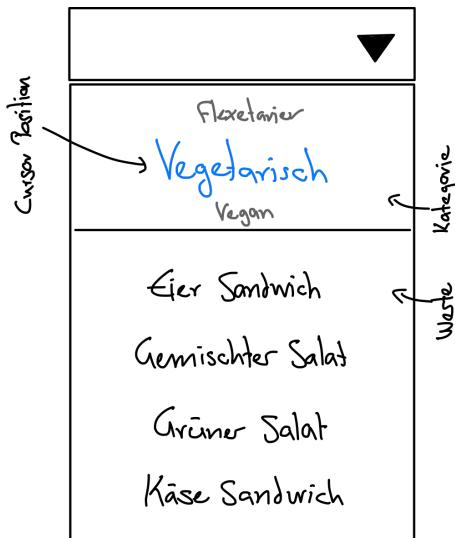


Abbildung 5.1: Mögliches UI mit Zeilen-Aufbau

die Selektion beeinflusst. Weitere Projektoren können neue, eventuell UI-Projektor spezifische Interaktionen bieten.

Ein Interaktions-Projektor kann das Auswahlkomponente bezogene Undo und Redo enthalten. Diese Funktion bietet an, eine geänderte Auswahl rückgängig zu machen oder zu wiederholen.

5.1.4 Spezifische Auswahlkomponenten und Erweiterungen

Zukünftig besteht die Möglichkeit Auswahlkomponenten für spezielle Anwendungsfälle zu erstellen. Ein altbekanntes Beispiel ist die Auswahl eines Datums mit Tag, Monat und Jahr. Dabei kann weitere Logik zur Kontrolle der möglichen Werte zum Einsatz kommen.

Das Erweitern der Multiselect-Funktionalität führt die Anpassung mehrerer Subkomponenten mit sich. Es betrifft die Implementation von der `SelectComponent` bis hin zu der `columnOptionsComponent`. Eine mögliche neue Single-Select Komponente bietet die Funktionalität mehrere Kategorien auswählen zu können. Hierbei ist zu unterscheiden, ob die ausgewählten Kategorien als AND- oder OR-Verknüpfung zu verstehen sind.

5.1.5 Performance verbessern und mögliche Datenmenge erhöhen

Die `SelectComponent` garantiert bis 5'000 Werte eine annehmbare⁴ Ladezeit. Diese Performance lässt sich weiter optimieren. Zudem führen diese Verbesserungen dazu, dass sich eine noch höhere Anzahl Optionen anwenden lässt. Eine Kombination der Auswahlkomponente mit der Lazy Table des Kolibri ermöglicht eine optimierte Performance. Denn die Lazy Table kann eine Datenmenge von über 100'000 Werte verwalten.

⁴Ladezeit < 500ms bis die Webseite fertig geladen ist

5.1.6 Accessibility verbessern

Die entstandene Auswahlkomponente ist nur für nicht beeinträchtigte Menschen optimiert. Zukünftig ist die Komponente noch für Screen-Reader und somit für Blinde zu ergänzen. Die getroffenen Styles benötigen ein Testing mit Sehbeeinträchtigten, um die Accessibility garantieren zu können. Dazu zählt zum Beispiel die bekannte Rot-Grün-Schwäche.

5.1.7 Abschliessende Bemerkungen

All diese Erweiterungen und Verbesserungen zielen darauf ab, die Auswahlkomponente für alle Endnutzer zugänglich zu machen und eine angenehme Benutzung zu ermöglichen. Zudem soll die `SelectComponent` einfach in jeden bestehenden JavaScript Code einzubinden sein. Anschliessend an diese Arbeit steht die Integration in das Toolkit Kolibri an.

Kapitel 6

Glossar

Begriff	Beschreibung
Aktuelle Browser	Edge: Version 127 (Windows) Chrome: Version 127 (Windows / Mac) Firefox: Version 128 (Windows / Mac) Safari: Version 17.5 (Mac)
ARIA-Rolle	Accessible Rich Internet Applications Role. Sie definiert die Bezeichnung oder Funktion eines HTML-Elements. Sie dient als Markierung für Struktur und erweitert die Bedienbarkeit für u.a. Screenreader.
Ausgrauen	Das Element erscheint in Grautönen mit eher schwachem Kontrast.
Client-Side	Direkt im Browser. Aktionen sehen den Server nie und bleiben im Browser. Der Code ist direkt auf der HTML-Seite eingebunden. In Zusammenhang mit Formularen erfährt der Server nichts von der Client-Side Validierung.
Do-Nothing-Implementation	Implementation, welche als Stellvertreter z.B. beim Null-Object Pattern dient. Eine Funktion mit einer solchen Implementation führt nichts aus und hat keine Seiteneffekte.
Hovern	Mit der Maus über ein Element fahren.
Immutable	Unveränderbar. Objekte sind nicht überschreibbar.
Kategorie	Gruppierung der Optionen. Das Selektieren einer Kategorie in einer SelectComponent reduziert die Anzahl der Optionen. Die Komponente zeigt nur noch Optionen an, welche der Kategorie / Gruppe zugewiesen sind.
KISS	Keep it Simple and Stupid. Ein Prinzip der Programmierung, in welcher alles möglich einfach zu halten ist. Einfache und kurze Codestücke minimieren das Risiko von Fehlern und Unverständlichkeit.
Multiselect	Auswahlkomponente (HTML select Element), bei welchem mehrere Werte selektierbar sind. Bei der Selektion einer zweiten Option hebt sich die vorherige Auswahl nicht auf. Auf Desktop-Browsern muss jedoch die passende Taste bei der Auswahl gedrückt sein, damit sich die Selektion erweitert.
Regex	Regular Expression. Beschreibt eine Zeichenkette. Syntaktische Regeln helfen bei der Verarbeitung von Texten.
Separation of Concern	Ein Prinzip, welches in der Programmierung beim Aufteilen eines Programmes zur Anwendung kommt. Jede Komponente besitzt nur eine Aufgabe. Dadurch entsteht ein modularer Aufbau.
Single-Select	Auswahlkomponente (HTML select Element), bei welchem nur ein Wert selektierbar ist. Bei der Selektion einer zweiten Option hebt sich die vorherige Auswahl auf.
Spiegelstrich	Ein Strich, welcher sich auf der linken Seite eines Elements bzw. einer Option befindet. Der Strich ist vertikal und dient zum Hervorheben des Elements.
Triggern	Auslösen eines Events.
UI/UX-Design	Design der Benutzeroberfläche und der Benutzererfahrung.
Workaround	Alternativer Code, wenn der Ursprüngliche nicht funktioniert. Eine Möglichkeit ein Problem zu umgehen.
* Tabellen-Fussnote	Bemerkung gilt für die ganze Tabelle.

Quelleverzeichnis

- Dev, F. (2020). *Webpage Rendering: How It Works + Tips on Optimization* [<https://www.quantum-real.com/blog/webpage-rendering-how-it-works-tips-on-optimization>, Gesehen 19/07/2024].
- contributors, M. (2023). *option: The HTML Option element* [<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/option>, Gesehen 09/04/2024].
- Ofoegbu, J. (2023). *Best Practices for Efficient DOM Manipulation in JavaScript* [<https://code.tutsplus.com/tutorials/for-efficient-dom-manipulation-in-javascript--cms-106848a>, Gesehen 03/07/2024].
- contributors, M. (2024a). *datalist: The HTML Data List element* [<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/datalist>, Gesehen 09/04/2024].
- contributors, M. (2024b). *input: The Input (Form Input) element* [<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input>, Gesehen 09/04/2024].
- contributors, M. (2024c). *select: The HTML Select element* [<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/select>, Gesehen 09/04/2024].
- GeeksforGeeks. (2024). *Null Object Design Pattern* [<https://www.geeksforgeeks.org/null-object-design-pattern/>, Gesehen 09/06/2024].
- König, P. D. (2024). *Projector Pattern* [<https://dierk.github.io/Home/projectorPattern/ProjectorPattern.html>, Gesehen 09/06/2024].
- Kumar, S. (2024). *Decorator Design Pattern* [https://www.geeksforgeeks.org/decorator-pattern/?ref=header_search, Gesehen 09/06/2024].
- Marti, R., & Burki, L. (2024). *Auswahlkomponente für das WebUI Toolkit Kolibri* [<https://kolibri-selection-component.gitbook.io/kolibri-auswahlkomponente/>, Gesehen 19/07/2024].
- Ola & Markus. (2024a). *Chromium-based Browsers* [<https://alternativeto.net/category/browsers/chromium-based/>, Gesehen 21/04/2024].
- Ola & Markus. (2024b). *Firefox-based Browsers* [<https://alternativeto.net/category/browse rs/chromium-based/>, Gesehen 21/04/2024].
- Wikipedia. (2024a). *Blink (browser engine)* [[https://en.wikipedia.org/wiki/Blink_\(browser_engine\)](https://en.wikipedia.org/wiki/Blink_(browser_engine)), Gesehen 28/04/2024].
- Wikipedia. (2024b). *WebKit* [<https://en.wikipedia.org/wiki/WebKit>, Gesehen 30/06/2024].

Abbildungsverzeichnis

2.1	Master-Detail Aufteilung der Auswahlkomponente	5
2.2	Geschlossene Komponente	5
2.3	Offene Komponente	5
2.4	Rendering Prozess	7
3.1	Geschlossenes Select OSX Safari	15
3.2	Geschlossenes Select OSX Firefox	15
3.3	Geschlossenes Select Windows Chrome	15
3.4	Offenes Select Windows Chrome	15
3.5	Offenes Select OSX Chrome	15
3.6	Geschlossene Datalist OSX Safari	16
3.7	Geschlossene Datalist OSX Firefox	16
3.8	Offene Datalist Windows Chrome	16
3.9	Offene Datalist OSX Chrome	16
3.10	Geschlossene Länderauswahl	25
3.11	Offene Länderauswahl	25
4.1	Figma Prototyp – Dropdown Komponente 1	30
4.2	Figma Prototyp – Dropdown Komponente 2	30
4.3	Figma Prototyp – Dropdown Komponente 3	30
4.4	Kolibri Design System – Farbpalette	31
4.5	Geschlossene SelectComponent	35
4.6	Ausgefüllte SelectComponent	35
4.7	Offene SelectComponent – 3 Zustände	35
4.8	Offene SelectComponent – 2 Spalten	35
4.9	Offene SelectComponent mit Bilder	36
4.10	Offene SelectComponent – 3 Spalten	36
4.11	Diagramm Select Component – erste Version	39
4.12	Diagramm ColumnComponent	39
4.13	Diagramm SelectComponent	40
4.14	Rendering Prozess	42
4.15	Performance Test vor Anpassungen	43
4.16	Performance Test nach Anpassungen	44
4.17	Manuelle Tests – Komponente	44
4.18	Automatisierte Tests	46
4.19	Verteilung Semester	47

4.20 Verteilung Studienrichtung	47
4.21 Meinung der Testpersonen	47
5.1 Mögliches UI mit Zeilen-Aufbau	51
B.1 Geschlossene SelectComponent	64
B.2 Fokussierte SelectComponent	64
B.3 Required SelectComponent	64
B.4 Disabled SelectComponent	64
B.5 Ausgefüllte SelectComponent	64
B.6 Ausgefüllte SelectComponent mit Bild	64
B.7 Offene SelectComponent - 1 Spalte	65
B.8 Offene SelectComponent mit Highlight	65
B.9 Offene SelectComponent mit Cursor Position	65
B.10 Offene SelectComponent mit Selektion	65
B.11 Offene SelectComponent mit Highlight & Cursor Position	65
B.12 Offene SelectComponent mit Highlight & Cursor Position & Selektion	65
B.13 Offene SelectComponent - 2 Spalten	66
B.14 Offene SelectComponent mit Kategorie-Selektion	66
B.15 Offene SelectComponent mit Bilder	66
B.16 Offene SelectComponent - 3 Spalten	66
B.17 Offene SelectComponent mit Kategorie-Selektion	66
B.18 Offene SelectComponent mit Kategorie- & Subkategorie-Selektion	66
C.1 Geschlossenes Select auf OSX Chrome	67
C.2 Geschlossenes Select auf Windows Chrome	67
C.3 Geschlossenes Select auf OSX Firefox	67
C.4 Geschlossenes Select auf Windows Firefox	67
C.5 Geschlossenes Select auf OSX Safari	67
C.6 Geschlossenes Select auf Windows Edge	67
C.7 Offenes Select auf OSX Chrome	68
C.8 Offenes Select auf OSX Firefox	68
C.9 Offenes Select auf OSX Safari	68
C.10 Offenes Select auf Windows Chrome	68
C.11 Offenes Select auf Windows Firefox	68
C.12 Offenes Select auf Windows Edge	68
C.13 Geschlossenes Select auf iOS Safari	68
C.14 Geschlossenes Select auf Android Firefox	68
C.15 Offenes Select auf iOS Safari	69
C.16 Offenes Select auf Android Firefox	69
C.17 Disabled Select auf OSX Chrome	69
C.18 Disabled Select auf OSX Firefox	69
C.19 Disabled Select auf OSX Safari	69
C.20 Disabled Select auf Windows Chrome	70
C.21 Disabled Select auf Windows Firefox	70
C.22 Disabled Select auf Windows Edge	70
C.23 Multiselect auf OSX Chrome	70
C.24 Multiselect auf Windows Chrome	70

C.25 Multiselect auf OSX Firefox	70
C.26 Multiselect auf Windows Firefox	70
C.27 Multiselect auf OSX Safari	71
C.28 Multiselect auf Windows Edge	71
C.29 Select mit Optgroups auf OSX Chrome	71
C.30 Select mit Optgroups auf OSX Firefox	71
C.31 Select mit Optgroups auf OSX Safari	71
C.32 Select mit Optgroups auf Windows Chrome	71
C.33 Select mit Optgroups auf Windows Firefox	71
C.34 Select mit Optgroups auf Windows Edge	71
C.35 Geschlossene Datalist auf OSX Chrome	72
C.36 Geschlossene Datalist auf Windows Chrome	72
C.37 Geschlossene Datalist auf OSX Firefox	72
C.38 Geschlossene Datalist auf Windows Firefox	72
C.39 Geschlossene Datalist auf OSX Safari	72
C.40 Geschlossene Datalist auf Windows Edge	72
C.41 Offene Datalist auf OSX Chrome	72
C.42 Offene Datalist auf Windows Chrome	72
C.43 Offene Datalist auf OSX Firefox	73
C.44 Offene Datalist auf Windows Firefox	73
C.45 Offene Datalist auf OSX Safari	73
C.46 Offene Datalist auf Windows Edge	73
C.47 Geschlossene Datalist auf iOS Safari	73
C.48 Geschlossene Datalist auf Android Firefox	73
C.49 Offene Datalist auf iOS Safari	74
C.50 Offene Datalist auf Android DuckduckGo	74
C.51 Disabled Datalist auf OSX Chrome	74
C.52 Disabled Datalist auf Windows Chrome	74
C.53 Disabled Datalist auf OSX Firefox	75
C.54 Disabled Datalist auf Windows Firefox	75
C.55 Disabled Datalist auf OSX Safari	75
C.56 Disabled Datalist auf Windows Edge	75
C.57 Labeled Datalist auf OSX Firefox	75
C.58 Labeled Datalist auf Windows Firefox	75
C.59 Labeled Datalist auf OSX Chrome	76
C.60 Labeled Datalist auf Windows Chrome	76
C.61 Labeled Datalist auf OSX Safari	76
C.62 Labeled Datalist auf Windows Edge	76
C.63 Range-Datalist auf OSX Chrome	76
C.64 Range-Datalist auf OSX Firefox	76
C.65 Range-Datalist auf OSX Safari	76
C.66 Range-Datalist auf Windows Chrome	76
C.67 Range-Datalist auf Windows Firefox	76
C.68 Range-Datalist auf Windows Edge	76
C.69 Color-Datalist auf OSX Chrome	77
C.70 Color-Datalist auf OSX Safari	77
C.71 Color-Datalist auf iOS Safari	77

C.72 Color-Datalist auf Windows Chrome	77
C.73 Color-Datalist auf Windows Firefox	77
C.74 Color-Datalist auf Windows Edge	77
C.75 Date-Datalist auf OSX Chrome	77
C.76 Date-Datalist auf Windows Chrome	77
C.77 Date-Datalist auf Windows Edge	77
C.78 Time-Datalist auf OSX Chrome	77
C.79 Time-Datalist auf Windows Chrome	77
C.80 Time-Datalist auf Windows Edge	77
D.1 User Test Programmierer Frage 1	86
D.2 User Test Programmierer Frage 2	86
D.3 User Test Programmierer Frage 3	86
D.4 User Test Programmierer Frage 4	86
D.5 User Test Programmierer Frage 5	87
D.6 User Test Programmierer Frage 6	87
D.7 User Test Programmierer Frage 7	87

Codeverzeichnis

3.1	Option Beispiel	10
3.2	Disabled Select Beispiel	10
3.3	Optgroup Beispiel	11
3.4	Datalist Beispiel	12
3.5	Länderauswahl Beispiel	25
4.1	CSS Imports	31
4.2	CSS für das Popover-Element	31
4.3	Aktuelle (<code>cursor-position</code>) und ausgewählte (<code>selected</code>) Option	32
4.4	Steuerung der Highlighting-Klassen	33
4.5	Event-Handling für die Auswahl einer Option	33
4.6	Optionen und Scrollbar-Styling	34
4.7	CSS-Styles im JavaScript	34
4.8	Null-Option Definition	38
4.9	<code>SelectComponentByTableValues</code> dekoriert <code>SelectComponentByCallbacks</code>	40
4.10	Popover-Container Beispiel	41
4.11	Workaround für ältere Browser	41
4.12	Performance Optimierung (aus <code>columnOptionsComponent.js</code>)	42
4.13	Inline-Style für Input-Feld	43
D.1	<code>userTesting.js</code>	78
D.2	<code>userTesting.html</code>	80
D.3	<code>starter.js</code>	81
D.4	<code>dataService.js</code>	83

Tabellenverzeichnis

3.1	Vergleich der Auswahlmöglichkeiten	9
3.2	Vergleich Interaktion Datalist & Select in Edge (Windows)	17
3.3	Vergleich Interaktion Datalist & Select in Chrome (Mac / Windows)	18
3.4	Vergleich Interaktion Datalist & Select in Firefox (Mac / Windows)	19
3.5	Vergleich Interaktion Datalist & Select in Safari (Mac)	20
3.6	Vergleich Interaktion Datalist & Select in Firefox (Android, mobile)	21
3.7	Vergleich Interaktion Datalist & Select in DuckDuckGo (Android, mobile) . .	21
3.8	Vergleich Interaktion Datalist & Select in Safari (iOS, mobile)	22
3.9	Aktionen bei der Länderauswahl Komponente	26
4.1	Aktionen bei der ersten Version der neuen Komponente	37

Anhang A

Aufgabenstellung

24FS_IMVS17: Generalisierte Auswahlkomponente für das Web UI Toolkit "Kolibri"

Betreuer: [Dierk König](#)
[Fabian Affolter](#)

	Priorität 1	Priorität 2
Arbeitsumfang:	P6 (360h pro Student)	---
Teamgrösse:	2er Team	---

Sprachen: Deutsch oder Englisch
Studiengang: Informatik

Ausgangslage

Die FHNW betreibt das Web UI Toolkit Kolibri, das die Errungenschaften aus den Web Aktivitäten der Hochschule für Technik der Wirtschaft zur Verfügung stellt. In dem Toolkit sind unter anderem standardisierte Komponenten für Benutzerinteraktionen verfügbar. Es sind aber noch nicht alle Interaktionsmöglichkeiten abgedeckt. Diese gilt es zu vervollständigen.

Ziel der Arbeit

Ziel der Arbeit ist die Gestaltung und Umsetzung einer Komponente, die eine Auswahl aus mehreren, vorgegebenen Auswahlmöglichkeiten ansprechend und effizient zu ermöglichen.

Das Vorgängerprojekt hat eine spezialisierte Lösung erarbeitet, die in diesem Projekt generalisiert werden soll.

Dabei ist auf Einhaltung der Qualitätsvorgaben für Kolibri Komponenten zu achten: Synchronisation von Gestaltung und Umsetzung, Einbindung in das Kolibri Design System, Visuelles und Interaktionsdesign, Validierung des Designs durch Benutzertests, automatisierte Umsetzungstests, Anbindung an die Kolibri Modelle und Projektoren, Usability Tests für die Developer Experience, Beweis der Nützlichkeit in einer Demo Applikation.



Kolibri Logo

Problemstellung

Die Standard-Auswahlkomponenten im Web (select und datalist) sind von der gestalterischen Seite her wenig ansprechend, schwer in ein vereinheitlichtes Datenmodell zu integrieren, inconsistent im Interaktionsdesign und lassen in der Benutzungseffizienz zu wünschen übrig. Diese Mängel gilt es auf eine Art zu beheben, die es dem Entwickler erlaubt, eine hochqualitative Auswahlmöglichkeit mit minimalem Aufwand in seine Applikation einzubinden.

Technologien/Fachliche Schwerpunkte/Referenzen

HTML, CSS, JavaScript Kolibri

Bemerkung

Dieses Projekt ist für Ramona Marti und Lea Burki reserviert.

Anhang B

Neue Komponente – Bilder

Country



Abbildung B.1: Geschlossene
SelectComponent

Country



Abbildung B.2: Fokussierte
SelectComponent

Country*



Abbildung B.3: Required
SelectComponent

Country

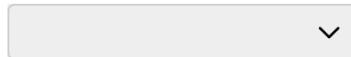


Abbildung B.4: Disabled
SelectComponent

Country



Abbildung B.5: Ausgefüllte
SelectComponent

CHF



Abbildung B.6: Ausgefüllte
SelectComponent mit Bild

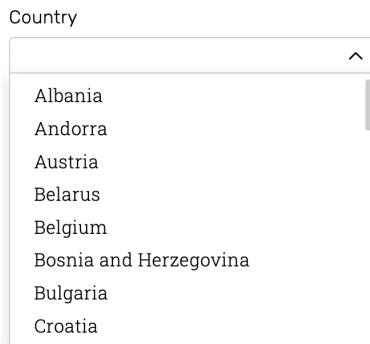


Abbildung B.7: Offene SelectComponent -
1 Spalte

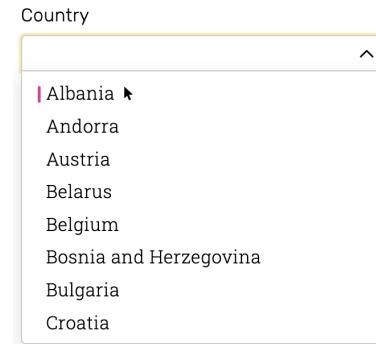


Abbildung B.8: Offene SelectComponent
mit Highlight

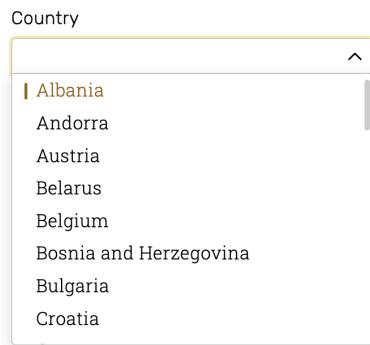


Abbildung B.9: Offene SelectComponent
mit Cursor Position



Abbildung B.10: Offene SelectComponent
mit Selektion

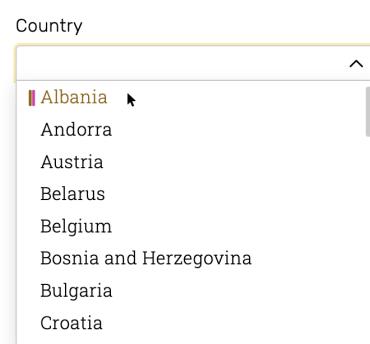


Abbildung B.11: Offene SelectComponent
mit Highlight & Cursor Position

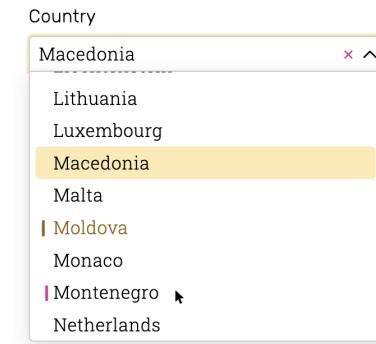


Abbildung B.12: Offene SelectComponent
mit Highlight & Cursor Position &
Selektion

Year	
1940's	1940
1950's	1941
1960's	1942
1970's	1943
1980's	1944
1990's	1945
2000's	1946
	1947

Abbildung B.13: Offene SelectComponent
- 2 Spalten

Year	
1940's	2000
1950's	2001
1960's	2002
1970's	2003
1980's	2004
1990's	2005
2000's	2006
	2007

Abbildung B.14: Offene SelectComponent
mit Kategorie-Selektion

City	
Europe	I CH
America	EU
	US

Abbildung B.15: Offene SelectComponent
mit Bildern

City		
Europe	Canada	Berlin
North America	France	Bern
Asia	Germany	Brugg
	Japan	Hamburg
	Swiss	L.A.
	United States	Marseille
		New York
		Ottawa

Abbildung B.16: Offene SelectComponent
- 3 Spalten

City		
Europe	Canada	L.A.
North America	United States	New York
Asia		Ottawa
		Washington DC

Abbildung B.17: Offene SelectComponent
mit Kategorie-Selektion

City		
Europe	Canada	L.A.
North America	United States	New York
Asia		Washington DC

Abbildung B.18: Offene SelectComponent
mit Kategorie- & Subkategorie-Selektion

Anhang C

Existierende Komponenten – Bilder

Select



Abbildung C.1: Geschlossenes Select auf OSX Chrome



Abbildung C.2: Geschlossenes Select auf Windows Chrome



Abbildung C.3: Geschlossenes Select auf OSX Firefox



Abbildung C.4: Geschlossenes Select auf Windows Firefox



Abbildung C.5: Geschlossenes Select auf OSX Safari

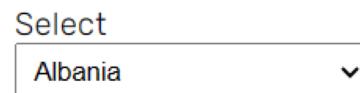


Abbildung C.6: Geschlossenes Select auf Windows Edge

ANHANG C. EXISTIERENDE KOMPONENTEN – BILDER

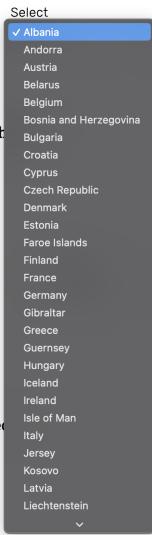


Abbildung C.7: Offenes Select auf OSX Chrome



Abbildung C.8: Offenes Select auf OSX Firefox

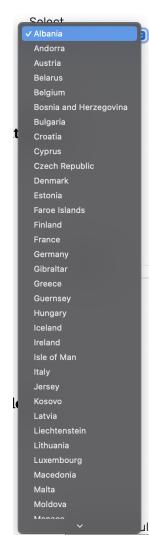


Abbildung C.9: Offenes Select auf OSX Safari

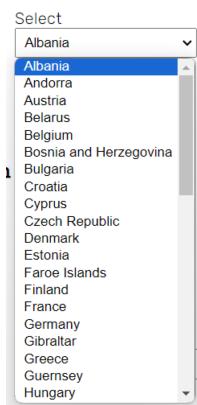


Abbildung C.10: Offenes Select auf Windows Chrome



Abbildung C.11: Offenes Select auf Windows Firefox

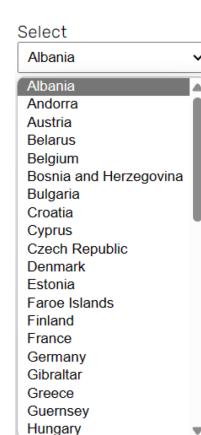


Abbildung C.12: Offenes Select auf Windows Edge



Abbildung C.13: Geschlossenes Select auf iOS Safari



Abbildung C.14: Geschlossenes Select auf Android Firefox

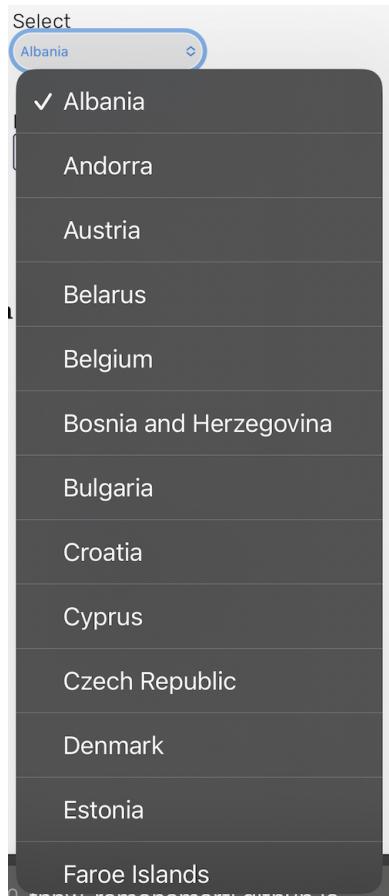


Abbildung C.15: Offenes Select auf iOS Safari

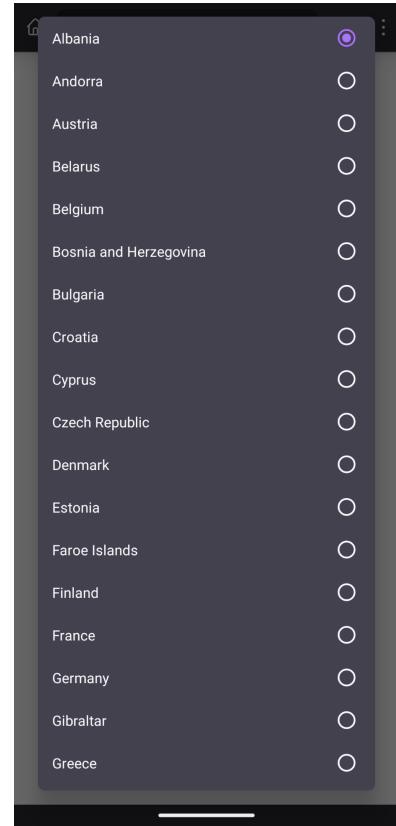


Abbildung C.16: Offenes Select auf Android Firefox

Abbildung C.17: Disabled Select auf OSX Chrome

Abbildung C.18: Disabled Select auf OSX Firefox

Abbildung C.19: Disabled Select auf OSX Safari

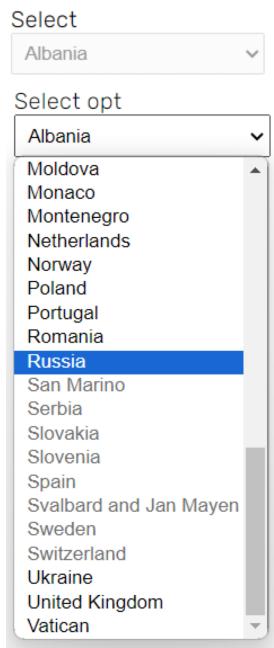


Abbildung C.20:
Disabled Select auf
Windows Chrome



Abbildung C.21:
Disabled Select auf
Windows Firefox

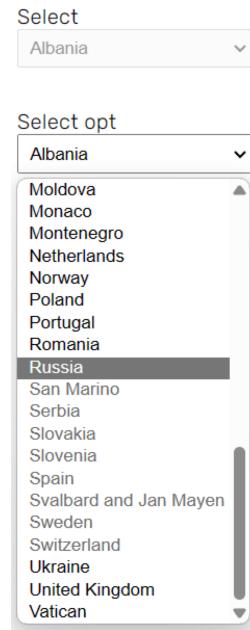


Abbildung C.22: Disabled
Select auf Windows Edge



Abbildung C.23: Multiselect auf OSX
Chrome



Abbildung C.24: Multiselect auf
Windows Chrome

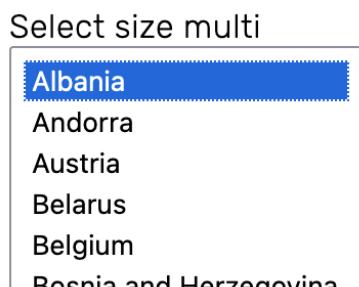


Abbildung C.25: Multiselect auf OSX
Firefox



Abbildung C.26: Multiselect auf
Windows Firefox

Select size multi



Abbildung C.27: Multiselect auf OSX
Safari

Select size multi

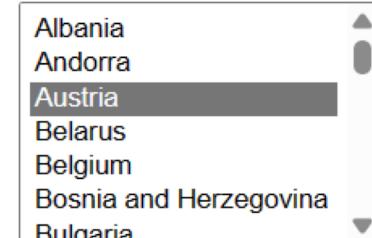


Abbildung C.28: Multiselect auf
Windows Edge

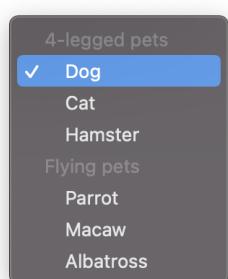


Abbildung C.29: Select
mit Optgroups auf OSX
Chrome

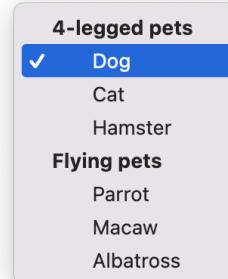


Abbildung C.30: Select
mit Optgroups auf OSX
Firefox

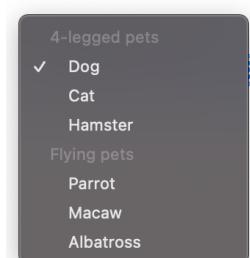


Abbildung C.31: Select
mit Optgroups auf OSX
Safari

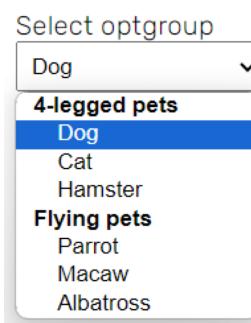


Abbildung C.32: Select
mit Optgroups auf
Windows Chrome



Abbildung C.33: Select
mit Optgroups auf
Windows Firefox

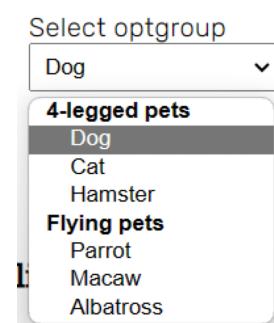


Abbildung C.34: Select
mit Optgroups auf
Windows Edge

Datalist

Datalist



Abbildung C.35: Geschlossene Datalist
auf OSX Chrome

Datalist



Abbildung C.36: Geschlossene Datalist
auf Windows Chrome

Datalist



Abbildung C.37: Geschlossene Datalist
auf OSX Firefox

Datalist



Abbildung C.38: Geschlossene Datalist
auf Windows Firefox

Datalist



Abbildung C.39: Geschlossene Datalist
auf OSX Safari

Datalist



Abbildung C.40: Geschlossene Datalist
auf Windows Edge

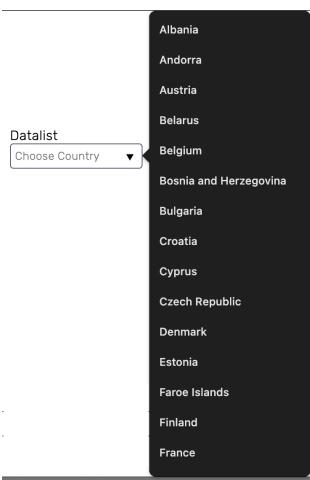


Abbildung C.41: Offene Datalist auf
OSX Chrome

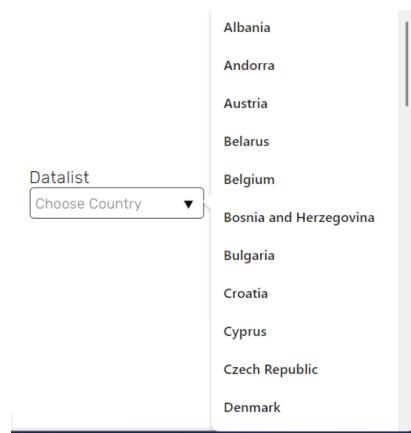


Abbildung C.42: Offene Datalist auf
Windows Chrome

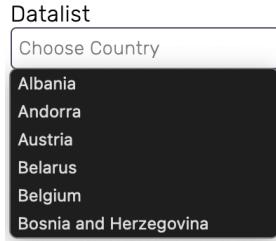


Abbildung C.43: Offene Datalist auf OSX Firefox

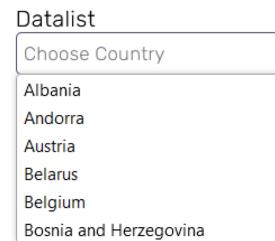


Abbildung C.44: Offene Datalist auf Windows Firefox

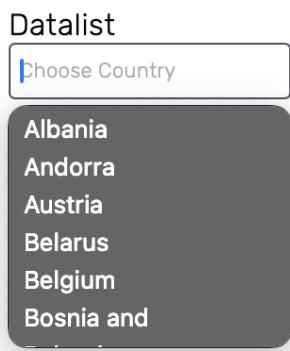


Abbildung C.45: Offene Datalist auf OSX Safari

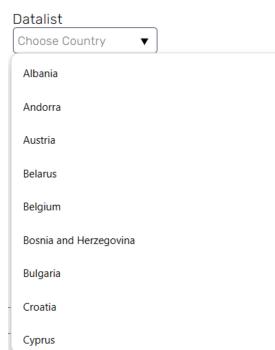


Abbildung C.46: Offene Datalist auf Windows Edge



Abbildung C.47: Geschlossene Datalist auf iOS Safari



Abbildung C.48: Geschlossene Datalist auf Android Firefox

ANHANG C. EXISTIERENDE KOMPONENTEN – BILDER

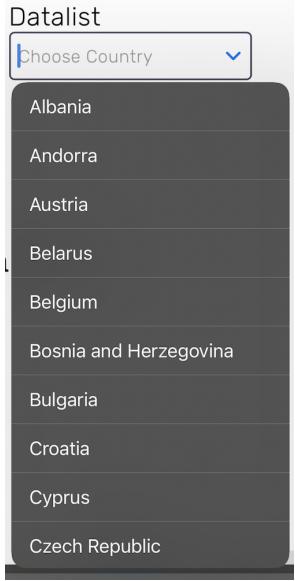


Abbildung C.49: Offene Datalist auf iOS Safari

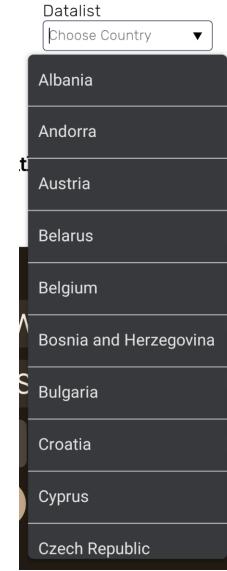


Abbildung C.50: Offene Datalist auf Android DuckduckGo

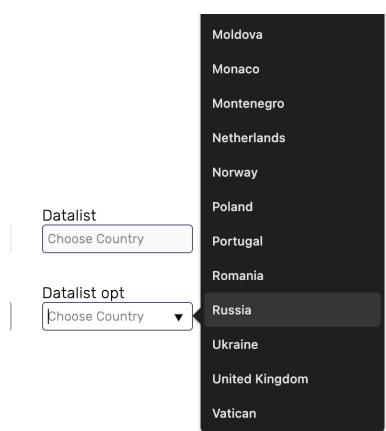


Abbildung C.51: Disabled Datalist auf OSX Chrome

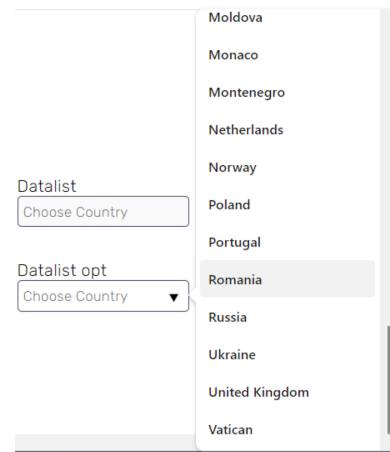


Abbildung C.52: Disabled Datalist auf Windows Chrome

Datalist

Datalist opt

- Portugal
- Romania
- Russia
- Ukraine
- United Kingdom
- Vatican

Abbildung C.53: Disabled Datalist auf OSX Firefox

Datalist

Datalist opt

- Portugal
- Romania
- Russia
- Ukraine
- United Kingdom
- Vatican

Abbildung C.54: Disabled Datalist auf Windows Firefox

Datalist

Datalist opt

- Portugal
- Romania
- Russia
- Ukraine
- United Kingdom
- Vatican

Abbildung C.55: Disabled Datalist auf OSX Safari

Datalist

Datalist opt

- Netherlands
- Norway
- Poland
- Portugal
- Romania
- Russia
- Ukraine
- United Kingdom
- Vatican

Abbildung C.56: Disabled Datalist auf Windows Edge

Datalist

- NorwayLabel
- SwedenLabel
- SwitzerlandLabel

Abbildung C.57: Labeled Datalist auf OSX Firefox

Datalist

- NorwayLabel
- SwedenLabel
- SwitzerlandLabel

Abbildung C.58: Labeled Datalist auf Windows Firefox

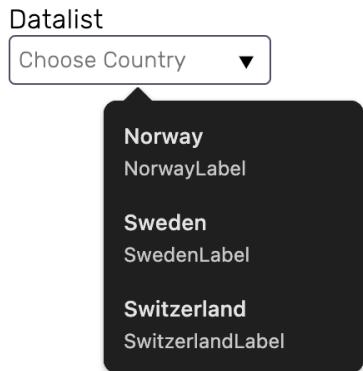


Abbildung C.59: Labeled Datalist auf OSX Chrome

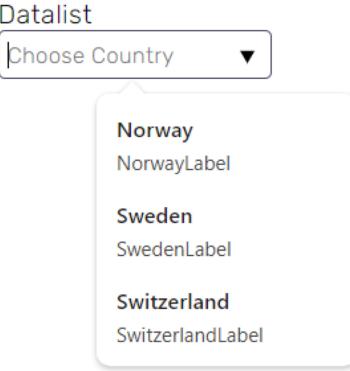


Abbildung C.60: Labeled Datalist auf Windows Chrome

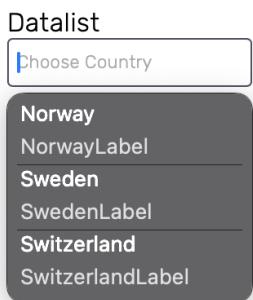


Abbildung C.61: Labeled Datalist auf OSX Safari

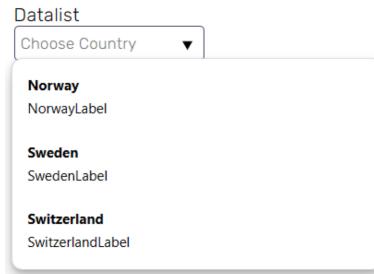


Abbildung C.62: Labeled Datalist auf Windows Edge

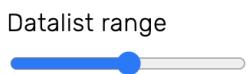


Abbildung C.63:
Range-Datalist auf OSX
Chrome



Abbildung C.64:
Range-Datalist auf OSX
Firefox

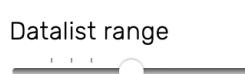


Abbildung C.65:
Range-Datalist auf OSX
Safari



Abbildung C.66:
Range-Datalist auf
Windows Chrome



Abbildung C.67:
Range-Datalist auf
Windows Firefox



Abbildung C.68:
Range-Datalist auf
Windows Edge

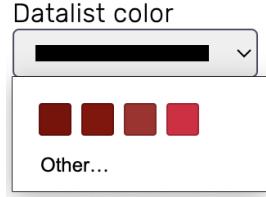


Abbildung C.69:
Color-Datalist auf OSX
Chrome



Abbildung C.70:
Color-Datalist auf OSX
Safari



Abbildung C.71:
Color-Datalist auf iOS
Safari

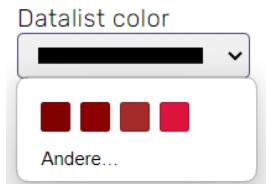


Abbildung C.72:
Color-Datalist auf
Windows Chrome



Abbildung C.73:
Color-Datalist auf
Windows Firefox

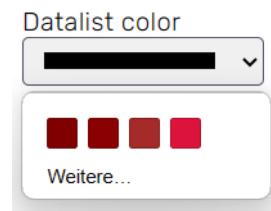


Abbildung C.74:
Color-Datalist auf
Windows Edge



Abbildung C.75:
Date-Datalist auf OSX
Chrome



Abbildung C.76:
Date-Datalist auf
Windows Chrome



Abbildung C.77:
Date-Datalist auf
Windows Edge

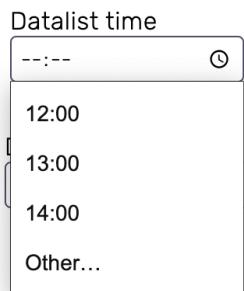


Abbildung C.78:
Time-Datalist auf OSX
Chrome

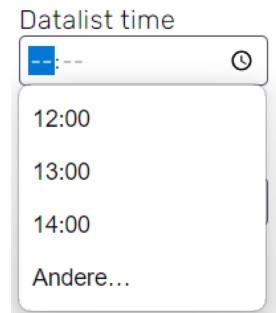


Abbildung C.79:
Time-Datalist auf
Windows Chrome

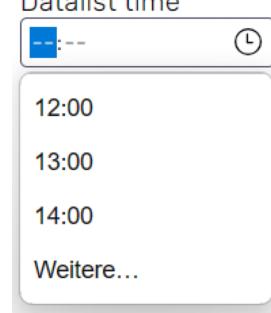


Abbildung C.80:
Time-Datalist auf
Windows Edge

Anhang D

User Test für Programmierer

Code

Code D.1: userTesting.js

```
1 import { SelectComponent, pageCss } from "https://
  fhnw-ramonamarti.github.io/Kolibri/src/examples/select-new/
  selectComponent.js";
2 import * as Service                      from "./dataService.js";
3
4 // load styles for new component
5 const style = document.createElement("style");
6 style.textContent = pageCss;
7 document.querySelector("head").append(style);
8
9 /**
10  * helper to replace elements to form
11  * @param {*} id - id of the html element to place the projected elements
12  * in
13  * @param {*} labelElement - new label to relape in the form
14  * @param {*} inputElement - new input element to relape in the form
15  * @example
16      addSelectViewToUi("task1", lunchLabelElement, lunchInputElement);
17 */
18 const addSelectViewToUi = (id, labelElement, inputElement) => {
19     const inputContainer = document.getElementById(id);
20     const labelContainer = document.getElementById(id + "-label");
21     labelContainer.replaceWith(labelElement);
22     inputContainer.replaceWith(inputElement);
23 }
24 // NOTE: You can read about our project in README.md file.
25 // Please follow the TODO's for the five tasks.
26 // The tasks are structured in the UI as follows:
27 //     1. Demo container with the usage of an existing way to create a
28 //         selection input.
29 //     2. Task container with TODO's placeholder is positioned in the
30 //         form
31 //             where the new componenent should be fit in.
32 // ----- TASK 1 -----
```

```

33 /*
34  TODO: Create a selection input using our new component.
35  The label should be 'Lunch' and the name 'lunch'.
36  The resulting component should provide 1 column with the data.
37  The data is provided by the function 'Service.getLunchTypes()' 
38  and can be used to fulfill the task.
39  To add the created view elements to the form you can use
40  the function 'addSelectViewToUi' from the top.
41  The id of the container to fill the view elements in is 'task1-lunch'.
42 */
43
44 // TODO: SOLUTION TASK 1 HERE
45
46
47
48 // ----- TASK 2 -----
49 // ----- TASK 2.1 -----
50 /*
51  TODO: Create a selection input using our new component.
52  The label should be 'Home region' and the name 'home-region'.
53  The resulting component should provide 2 column with the value data
54  and categories.
55  The value data is provided by the function ' 
56   Service.getRegionsByCountryChDeAt()', 
57  the categories are provided by the function ' 
58   Service.getCountriesChDeAt()' 
59  and they can be used to fulfill the task.
60  To add the created view elements to the form you can use
61  the function 'addSelectViewToUi' from the top.
62  The id of the container to fill the data view elements is ' 
63   task2-home-region'.
64 */
65
66 // ----- TASK 2.2 -----
67 /*
68  TODO: Create a selection input using our new component.
69  The label should be 'Birth region' and the name 'birth-region'.
70  The resulting component should provide 2 column with the value data
71  and categories.
72  The value data is provided by the function ' 
73   Service.getRegionsByCountry()', 
74  the categories are provided by the function 'Service.getCountries()' 
75  and they can be used to fulfill the task.
76  To add the created view elements to the form you can use
77  the function 'addSelectViewToUi' from the top.
78  The id of the container to fill the data view elements is ' 
79   task2-birth-region'.
80 */
81
82
83 // ----- TASK 2.3 -----
84 /*

```

```

85     TODO: Create a selection input using our new component.
86     The label should be 'Birth year' and the name 'birth-year'.
87     The resulting component should provide 2 column with the value data
88     and categories.
89     The value data is provided by the function 'Service.getYearsByDecade()'
90     ,
91     the categories are provided by the function 'Service.getDecades()'
92     and they can be used to fulfill the task.
93     To add the created view elements to the form you can use
94     the function 'addSelectViewToUi' from the top.
95     The id of the container to fill the data view elements is '
96     task2-birth-year'.
97
98
99
100
101 // -----
102 // Please fill out the question form to give us feedback about how the
103 // tasks worked.
104 // Please send your solution as a zip back to Ramona Marti on MS Teams.

```

Code D.2: userTesting.html

```

1  <!DOCTYPE html>
2  <html lang="en">
3      <head>
4          <meta charset="UTF-8" />
5          <meta name="viewport" content="width=device-width,
6              initial-scale=1.0" />
7          <title>Auswahlkomponente User Test</title>
8          <link
9              rel="shortcut icon"
10             type="image/png"
11             href="https://fhnw-ramonamarti.github.io/Kolibri/img/logo/
12                 logo-60x54.png"
13             />
14          <link
15              rel="stylesheet"
16              href="https://fhnw-ramonamarti.github.io/Kolibri/css/
17                  kolibri-base.css"
18              />
19          <style>
20              .newComponent {
21                  color: red;
22                  font-weight: bold;
23              }
24              label {
25                  display: flex;
26                  align-items: center;
27              }
28              select,
29              input {
30                  border: #ccc solid 1px;
31                  font-size: 1rem;
32                  height: 2rem;
33                  border-radius: 4px;

```

```

31         }
32         .task .select-input-component {
33             width: 320px;
34         }
35     </style>
36 </head>
37 <body>
38     <div class="demo" id="demo1"></div>
39     <div class="task" id="task1"></div>
40     <div class="demo" id="demo2"></div>
41     <div class="task" id="task2"></div>
42     <div class="demo" id="demo3"></div>
43     <div class="task" id="task3"></div>
44     <script type="module" src="starter.js"></script>
45     <script type="module" src="userTesting.js"></script>
46 </body>
47 </html>

```

Code D.3: starter.js

```

1 import { projectForm, FORM_CSS } from "https://fhnw-ramonamarti.github.io
   /Kolibri/src/kolibri/projector/simpleForm/simpleFormProjector.js"
2 import { SimpleFormController } from "https://fhnw-ramonamarti.github.io
   /Kolibri/src/kolibri/projector/simpleForm/simpleFormController.js"
3 import { TEXT, CHOICE, COMBOBOX } from "https://fhnw-ramonamarti.github.io
   /Kolibri/src/kolibri/util/dom.js";
4
5 import * as Service from "./dataService.js";
6
7 // load style
8 const style = document.createElement("style");
9 style.textContent = FORM_CSS;
10 document.querySelector("head").append(style);
11
12 // prepare filling container
13 const fillContainer = (id, title, buildForm) => {
14     const formHolder = document.getElementById(id);
15     if (null != formHolder) {
16         const [fieldset] = buildForm();
17         formHolder.innerHTML = `<h3>${title}</h3>`;
18         formHolder.append(fieldset);
19     }
20 };
21
22 // prepare containers to do the tasks in
23 const replaceField = (id, name) => {
24     const inputElement = document.getElementById(id).querySelector(`[name=${name}]`);
25     const inputId = inputElement.id;
26     const fieldset = inputElement.closest("fieldset");
27     const labelElement = fieldset.querySelector(`[for="${inputId}"]`);
28     const spanElement = fieldset.querySelector(`span[data-id="${inputId}"]`);
29
30     const containerLabel = document.createElement("span");
31     containerLabel.id = id + "_" + name + "-label";
32     const container = document.createElement("span");
33     container.id = id + "_" + name;
34     container.textContent = "TODO";

```

```
35     container.classList.add("newComponent");
36     spanElement.replaceWith(container);
37     labelElement.replaceWith(containerLabel);
38
39     return [containerLabel, container];
40 };
41
42 // ----- DEMO 1 -----
43 const demo1 = () => {
44     /** @type { Array<OptionType> } */
45     const types = Service.getLunchTypes().map((types) => ({ value: types
        }));
46
47     const formStructure = [
48         { value: "", label: "Name", name: "name", type: TEXT },
49         { value: "", label: "Lunch", name: "lunch", type: COMBOBOX, list:
            types },
50     ];
51     const controller = SimpleFormController(formStructure);
52     return projectForm(controller);
53 };
54 fillContainer("demo1", "Demo 1 - Short datalist", demo1);
55
56 // ----- TASK 1 -----
57 const task1 = () => {
58
59     const formStructure = [
60         { value: "", label: "Name", name: "name", type: TEXT },
61         { value: "", label: "Lunch", name: "lunch", type: "hidden" },
62     ];
63     const controller = SimpleFormController(formStructure);
64     return projectForm(controller);
65 };
66 fillContainer("task1", "Task 1 - 1 column layout", task1);
67 replaceField("task1", "lunch");
68
69 // ----- DEMO -----
70 const demo2 = () => {
71     /** @type { Array<OptionType> } */
72     const years = Service.getYearsByDecade().map((year) => ({ value: year
        }));
73
74     /** @type { Array<OptionType> } */
75     const filterRegionsChDeAt = (...countryCode) =>
76         Service.getRegionsByCountryChDeAt(...countryCode).map((region) =>
            ({ value: region }));
77
78     /** @type { Array<OptionType> } */
79     const filterRegions = (...countryCode) =>
80         Service.getRegionsByCountry(...countryCode).map((region) => ({
            value: region }));
81     const formStructure = [
82         { value: "", label: "Name", name: "name", type: TEXT },
83         {
84             value: "Aargau",
85             label: "Home region",
86             name: "home-region",
87             type: CHOICE,
88             list: filterRegionsChDeAt(),
```

```

89         },
90         {
91             value: "",
92             label: "Birth region",
93             name: "birth-region",
94             type: CHOICE,
95             list: filterRegions(),
96         },
97         { value: "", label: "Birth year", name: "birth-year", type: CHOICE
98             , list: years },
99     ];
100     const controller = SimpleFormController(formStructure);
101     return projectForm(controller);
102 };
103 fillContainer("demo2", "Demo 2 - Long selects", demo2);
104 // ----- TASK 2 -----
105 const task2 = () => {
106
107     const formStructure = [
108         { value: "", label: "Name", name: "name", type:
109             TEXT },
110         { value: "", label: "Home region", name: "home-region", type:
111             "hidden" },
112         { value: "", label: "Birth region", name: "birth-region", type:
113             "hidden" },
114         { value: "", label: "Birth year", name: "birth-year", type:
115             "hidden" },
116     ];
117     const controller = SimpleFormController(formStructure);
118     return projectForm(controller);
119 };
116 fillContainer("task2", "Task 2 - 2 column layout", task2);
117 replaceField("task2", "home-region");
118 replaceField("task2", "birth-region");
119 replaceField("task2", "birth-year");

```

Code D.4: `dataService.js`

```

1 export {
2     getAllContinents,
3     getCountries,
4     getCountriesChDeAt,
5     getRegionsByCountry,
6     getRegionsByCountryChDeAt,
7     getYearsByDecade,
8     getDecades,
9     getLunchTypes,
10 };
11
12 /**
13  * @type { (String) => String } */
14 const getAllContinents = () => [
15     ...new Set(...allCountriesWithContinent.map((country) =>
16         country.continent)),
17 ];
18 /**
19  * @type { (String) => String } */
20 const getCountries = (continent) =>
21     allCountriesWithContinent

```

```

20     .filter((e) => e.continent === continent || !continent)
21     .map((country) => country.country);
22
23 /** @type { (String) => String } */
24 const getRegionsByCountry = (country) =>
25   allRegionsWithCountry
26     .filter((e) => e.country === country || !country)
27     .map((region) => region.region).sort();
28
29 /** @type { (String) => String } */
30 const getCountriesChDeAt = () => ["Switzerland", "Germany", "Austria"];
31
32 /** @type { (String) => String } */
33 const getRegionsByCountryChDeAt = (country) =>
34   allRegionsWithCountry
35     .filter(
36       (e) => e.country === country || (!country && ["CH", "DE", "AT"]
37         .includes(e.code))
38     )
39     .map((region) => region.region).sort();
40
41 /** @type { (String) => String } */
42 const getYearsByDecade = (decade) => {
43   const decadeStart = decade?.slice(0, 3);
44   const data = [...Array(90).keys()].map((e) => e + 1930 + "");
45   return data.filter((e) => !decade || e.startsWith(decadeStart));
46 };
47
48 /** @type { (String) => String } */
49 const getDecades = () => [...Array(9).keys()].map((e) => e * 10 + 1930 +
50   "s");
51
52 /** @type { (String) => String } */
53 const getLunchTypes = () => ["all", "vegetarian", "vegan", "flexitarian",
54   "gluten-free", "lactose-free"];
55
56 /* data tables for 'allCountriesWithContinent' and 'allRegionsWithCountry'
57 */
58 /* Link: https://github.com/fhnw-ramonamarti/fhnw-ramonamarti.github.io/blob/main/ip6/userTest/dataService\_old.js */

```

Resultate

Siehst du irgendwo Verbesserungspotenzial?

- Detail: `numberOfColumns` ist etwas "verbose"
- Doku: In der Code Doku von `SelectComponent` dürfte der Return Value besser beschrieben werden (welches Array Element ist was). Das wird erst in den Anwendungsbeispielen klar.

Keyboard navigation. Using keyboard to narrow down possible categories/values (maybe fuzzy searching), otherwise it takes me longer to find something than compared to a standard dropdown, even if the list there is bigger.

ich finde es etwas unintuitiv beim `SelectComponent` neben den offensichtlich verständlichen `selectAttributes` (`label`, `name`, `numberOfColumns`) noch ein Callback mitzugeben. ich hätte mir mehr Beschreibung gewünscht wie ich die Komponente verwenden muss. (ich musste eher danach suchen.)

Bessere Dokumentation war verwirrend (Code).

Automatisches Schliessen der Komponente nach der Auswahl würde ich praktisch finden.

Es würde helfen, wenn die Types im JSDOC spezifiziert wären und wenn die Library eingebunden wäre. Dann wäre die Dokumentation leichter auffindbar.

War mir am Anfang nicht klar dass was SelectComponent() zurückgibt und ob das Input-element sowie dass label-element selbst erstellt werden soll. Danach war es intuitiv anzuwenden.

Bemerkung

ich persönlich hatte zu beginn schwierigkeiten zu verstehen wie das framework funktioniert. die beschreibung für die tasks (ab task2) sind zum teil etwas unklar

=> The value data is provided by the function ‘Service.getRegionsByCountry()’, the categories are provided by the function ‘Service.getCountries()’

=> RegionsByCountry muss man noch das Country mitgeben. Hätte in der beschreibung drinstehen können.

Nitpicking: while I see why the component is imported from a URL for the user tests, it makes it hard to use on a spotty network.

Isch s selectAttributes ‘numberOfColumns‘ wörklich nötig? Vo mir uus gseh, chönt mer ‘numberOfColumns‘ vo ‘serviceCallbacks.length‘ ableite...?

Ehr hend nah recht es performance Problem wenn d liste lang isch... Dur s implementiere vo Task 2.2 brucht d websiite ca. 5 sekunde zum lade (rein Javascript execution). Das liht ned ahm Javascript, sondern am HTML rendering sowiit ih gseh ha. Ha suscht nah e Performance Trace gmacht wo mer in Chrome chan drii lade ahhghänkt.

Isch betz speziell, dass ‘serviceCallbacks‘ array in vercherter reihefolg muen drii geh werde wies denne ahhzeigt wird...

Wenn mer 2 Kollone het, aber rechts ke uuswahl, de chan mer au nüt uuswähle... Ja isch z erwarte aber isch au ih de Demo vorhande

För mich isch ned immer klar gsii, wenns nah meh optione unde oder obedraa het.

ANHANG D. USER TEST FÜR PROGRAMMIERER

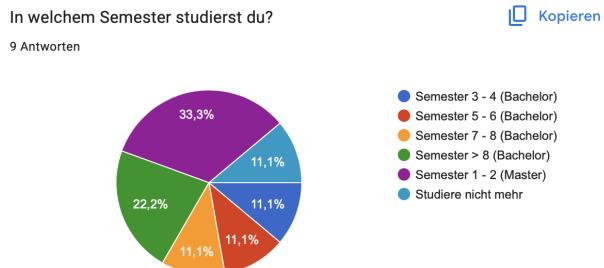


Abbildung D.1: User Test Programmierer Frage 1

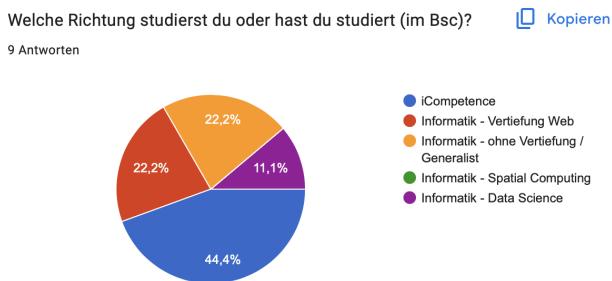


Abbildung D.2: User Test Programmierer Frage 2

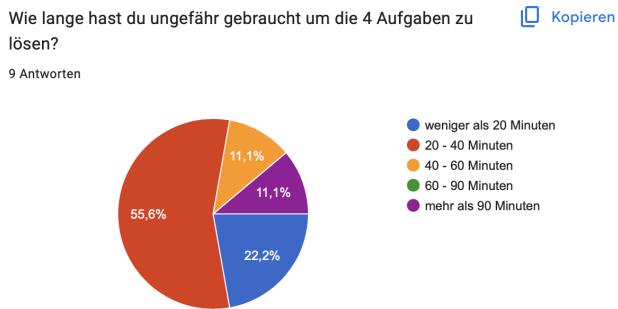


Abbildung D.3: User Test Programmierer Frage 3

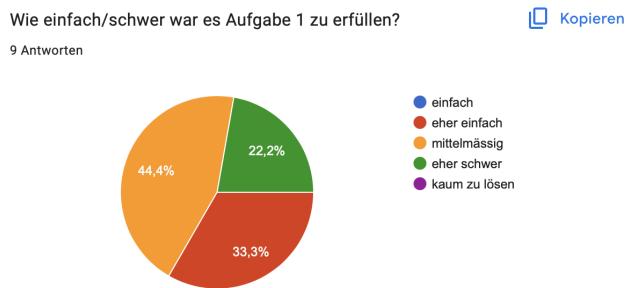


Abbildung D.4: User Test Programmierer Frage 4

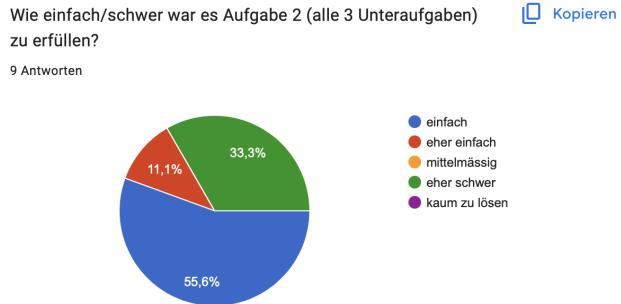


Abbildung D.5: User Test Programmierer Frage 5

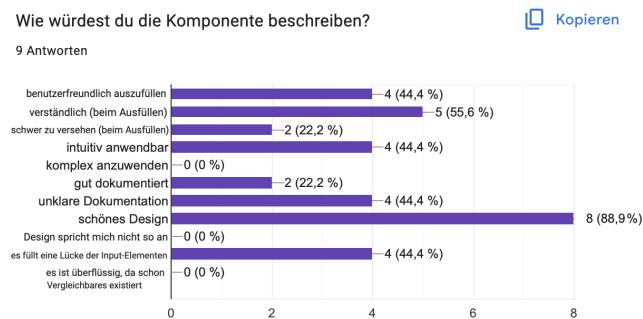


Abbildung D.6: User Test Programmierer Frage 6

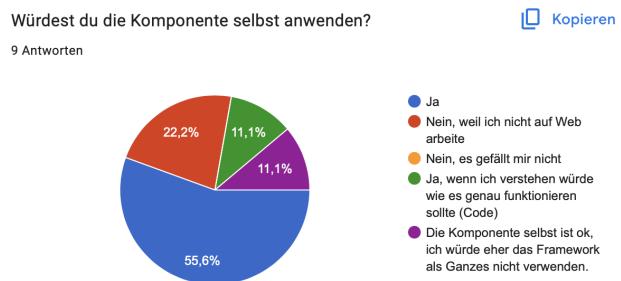


Abbildung D.7: User Test Programmierer Frage 7

Anhang E

Personas

Love Berg

#travellover

#technophile

#efficiency

Hintergrund

Love ist eine dynamischer Marketing-Experte, der in der digitalen Welt zu Hause ist. Er nutzt täglich verschiedene Online-Tools, um ihre Projekte zu verwalten und mit internationalen Kunden zu kommunizieren. Love liebt es, neue Kulturen zu entdecken und hat ein Faible für Sprachen.

Schmerzpunkte

- Frustration über umständliche oder langsame Anwendungen.
- Verwirrende Benutzeroberflächen beeinträchtigen seine Arbeitseffizienz.

Ziele und Bedürfnisse

- Effizienz beim Arbeiten
Schneller Zugriff auf internationale Daten: Love benötigt effiziente Tools, um globale Marktanalysen durchzuführen.
- Technologieaffinität
Benutzerfreundliche Anwendungen: Love bevorzugt intuitive und einfach zu bedienende Schnittstellen.
- Zeitsparnis
Als Vielbeschäftigter sucht Love nach Lösungen, die ihm Arbeitsergebnisse optimieren und Zeit sparen.



**"Effizient arbeiten mit
modernen
Applikationen"**

Alter
Beruf
Wohnort

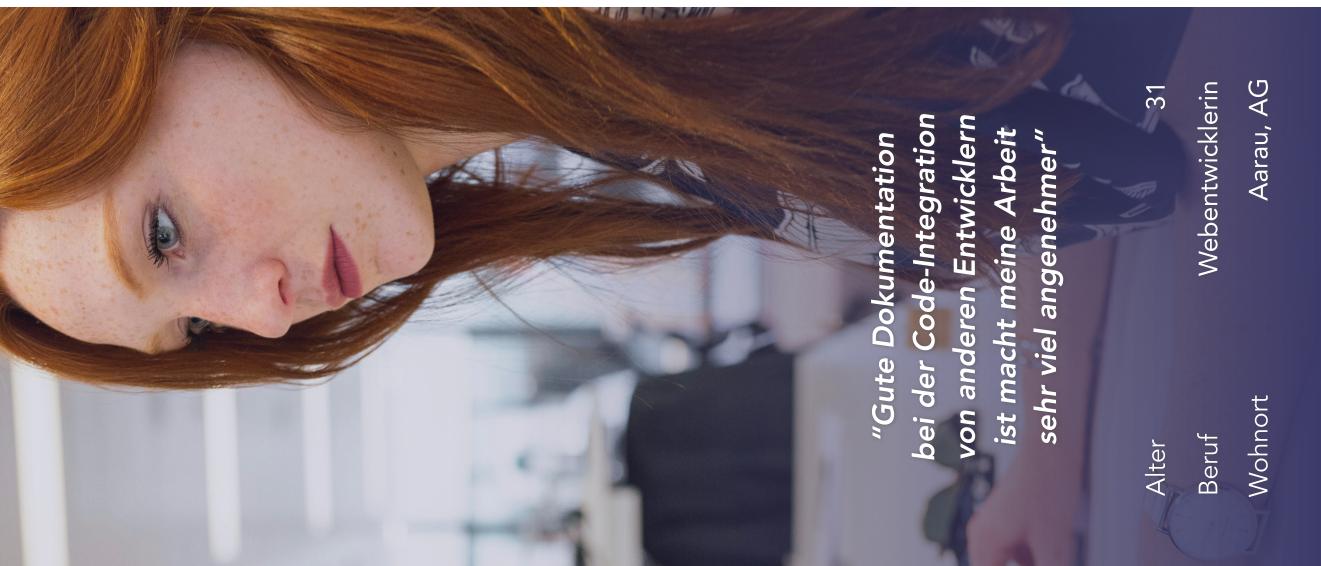
28
Marketig Manager
Zürich, ZH

Estrid Miller

#coffeelover

#csswizard

#moderndesign



USER PERSONA

Hintergrund

Estrid ist eine leidenschaftliche Webentwicklerin mit einem tiefen Verständnis für Frontend-Technologien. Sie arbeitet gerne an Projekten, die sowohl technische Herausforderungen als auch Design-Aspekte beinhalten. Estrid genießt es, in ihrer Freizeit an eigenen Projekten zu tüfteln und sich mit der neuesten Technik auseinanderzusetzen.

Schmerzpunkte

- Fehlende Dokumentation und Unterstützung bei der Integration neuer Komponenten.
- Unflexible oder schwer anpassbare UI-Komponenten limitieren seine Kreativität.

Ziele und Bedürfnisse

- Effiziente Entwicklungs-Tools
Estrid sucht nach Lösungen, die ihre Entwicklungsprozesse vereinfachen und beschleunigen.
- Optimierung der User Experience
Sie legt großen Wert auf die Gestaltung benutzerfreundlicher Interfaces.

Präferenzen

- Estrid bevorzugt modulare und leicht anpassbare Komponenten, die eine schnelle Entwicklung ermöglichen.

Sie schätzt Werkzeuge, die eine flexible und effiziente Entwicklung ermöglichen, besonders solche, die gut dokumentiert und mit umfangreichen Anpassungsmöglichkeiten versehen sind. Sie schätzt Werkzeuge, die eine flexible und effiziente Entwicklung ermöglichen, besonders solche, die gut dokumentiert und mit umfangreichen Anpassungsmöglichkeiten versehen sind. Schnittstellen, die eine schnelle Integration ermöglichen und mit aktuellen Programmierstandards kompatibel sind, zieht sie vor. Sie bevorzugt Lösungen, die sowohl die Entwicklungszeit verkürzen als auch die Benutzererfahrung verbessern, wie z.B. Komponenten mit responsivem Design und Accessibility-Features. Die Möglichkeit, Codebeispiele und Best Practices zu teilen, ist für sie von großem Wert.

**"Gute Dokumentation
bei der Code-Integration
von anderen Entwicklern
ist macht meine Arbeit
sehr viel angenehmer"**

Alter
Beruf
Wohnort

31
Webentwicklerin
Aarau, AG

Anhang F

Links

- **Artefakte-Seite:** <https://fhnw-ramonamarti.github.io/ip6/index.html>
- **Repository:** Projekt-Ordner
 - **Komponente:** <https://github.com/fhnw-ramonamarti/Kolibri/tree/interactionLetters/docs/src/kolibri/projector/selectComponent>
 - **Beispiele:** <https://github.com/fhnw-ramonamarti/Kolibri/tree/interactionLetters/docs/src/examples/selectComponent>

Anhang G

API

Component specific Types

Type	Description
SimpleOption	{ label: String?, value: String }
OptionData	{ String SimpleOption }
OptionsTable	Array<Array<OptionData>>
Callback	(...String) => Array<OptionData>

The `SelectAttributes` object contains the following properties:

Property	Type	Description
label	String	Content of the Label for the input.
name	String	Name of the input that is sent with the form.
isRequired	Boolean	Defines if the selection of an option is required to send the form.
isDisabled	Boolean	Defines if the value can be changed and if the interaction work.
isCursorPositionWithSelection	Boolean	Defines if the cursor position is linked with the selection. If true: the keyboard interaction directly changes the selection.

Components

SelectComponentByCallbacks

Creates an input component with the functionality: selecting an option in a given list.

Returns: SelectComponent

Function	Type	Description
getSelectController()	SelectController	Provides the whole functionality of the select component.
getComponentView()	HTMLDivElement	Provides the whole view with the select element and the label.
getLabelElement()	HTMLLabelElement	Provides the label element from the whole view.

Parameters

Name	Type	Description
selectAttributes	SelectAttributes	Defines properties to customize the select component.
serviceCallbacks	Array<Callback>	Lists the Callback functions that provide the data for the options. The number of callbacks in the array define the number of columns in the component. The most right or biggest indexed callback in the array contains the value providing function. The order is defined as in the view: from most general category to specific value. As parameter the callbacks take the string of the parent category of the current column.

Example

```
1 const getYearsByDecade = (...decades) => {
2   const decadeStarts = decades.map(decade => decade.slice(0, 3));
3   const data = [...Array(70).keys()].map((e) => e + 1940 + "s");
4   return data.filter((e) => decadeStarts.length === 0 ||
5     decadeStarts.includes(e.slice(0, 3)));
6 };
7 const decades = [...Array(7).keys()].map((e) => e * 10 + 1940 + "s");
8 const selectAttribute = { name: "year", label: "Year" };
9 const columnServiceCb = [ () => decades, getYearsByDecade ];
10 const selectComponent = SelectComponentByCallbacks(
11   selectAttribute,
12   columnServiceCb
13 );
14 const componentYear = document.getElementById("componentContainer");
15 componentYear.append(selectComponent.getComponentView());
```



Year

Year	
1940's	1940
1950's	1941
1960's	1942
1970's	1943
1980's	1944
1990's	1945
2000's	1946
	1947

Beispiel `SelectComponentByCallbacks`

SelectComponentByTableValues

Creates an input component with the functionality: selecting an option in a given list.

Returns: SelectComponent

Function	Type	Description
getSelectController()	SelectController	Provides the whole functionality of the select component.
getComponentView()	HTMLDivElement	Provides the whole view with the select element and the label.
getLabelElement()	HTMLLabelElement	Provides the label element from the whole view.

Parameters

Name	Type	Description
selectAttributes	SelectAttributes	Defines properties to customize the select component.
optionsTable	OptionsTable	Lists the options with all the categories denormalized. Each Entry in the outer array contains one OptionData per column. If a value has multiple categories in the same column, it needs multiple entries. If a value has no category in a column 'null' can be placed in that place. Every entry shuold have the same length and defines the options from most general category to value. The shortest entry array defines the number of columns. The category columns will use the distinct OptionData as options.
sortColumnOptionsAlphabetical	Boolean	Defines if the options of each column are sorted alphabetically. The sorting will be applied to every column or none of them.

Example

```
1 const tableOptions = [
2   [ "1940's", 1940 ],
3   [ "1940's", 1941 ],
4   [ "1940's", 1942 ],
5   [ "1940's", 1943 ],
6   /* ... */
7   [ "1950's", 1958 ],
8   [ "1950's", 1959 ],
9   [ "1960's", 1960 ],
10  [ "1960's", 1961 ],
11  /* ... */
12  [ "2000's", 2007 ],
13  [ "2000's", 2008 ],
14  [ "2000's", 2009 ],
15];
```

```
16 const selectAttribute = { name : "year", label: "Year" };
17 const selectComponent = SelectComponentByTableValues(
18   selectAttribute,
19   tableOptions
20 );
21
22 const componentYear = document.getElementById("componentContainer");
23 componentYear.append(selectComponent.getComponentView());
```

Year



Beispiel SelectComponentByTableValues

ColumnOptionsComponent

Creates a column view of a given list of options.

Returns: ColumnOptionsComponent

Function	Type	Description
getOptions()	Array<Option>	Lists all options contained in the column. The order is the one from adding the options.
addOptions(Array<Option>)	void	Adds all passed options to the end of the column. If an option is already contained it is ignored while adding. Options count as same if the label and the value are same.
delOptions(Array<Option>)	void	Removes all passed options from the column. If an option is not contained the code just continues. Options count as same if the label and the value are same.
clearOptions()	void	Removes all options from the column.
getSelectedOption()	Option	Read the selected option.
setSelectedOption(Option)	void	Changes the selected option to the passed option. Only one option can be selected in the column component.
clearSelectedOption()	void	Resets the selection to nothing selected.
onOptionSelected(Consumer<Option>)	void	Adds a listener to the change of the option selection observable.
isSelectedOptionDisabled()	Boolean	Checks if the selected option is disabled to be changed.
setSelectedOptionDisabled(bool)	void	Changes the possibility to change the selection. Sets the disabled to the passed value. If true the selection cannot be changed.
onSelectedOptionDisabledChanged(Consumer<Option>)	void	Adds a listener to the change of the disable observable.
getColumnView()	HTMLDivElement	Provides the whole column component. It is not linked to an input field.

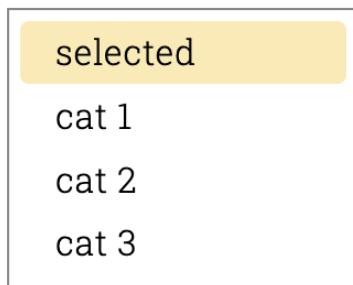
Parameters

Name	Type	Description
cursorPositionController	SelectedOptionController	Controller for the current cursor position. The cursor position is used as indicator at which option the keyboard is currently at.
columnNumber	Number?	Optional position of index of the column. It is used to identify the column in a select component.



Example

```
1 const cursorPos = SelectedOptionController();
2 const component = ColumnOptionsComponent(cursorPos);
3 document.getElementById("componentContainer").append(
    component.getColumnView());
4
5 const selectedOption = CategoryOption("selected");
6 const options = [
7     selectedOption,
8     CategoryOption("cat 1"),
9     CategoryOption("cat 2"),
10    CategoryOption("cat 3")
11];
12
13 component.addOptions(options);
14 component.setSelectedOption(selectedOption);
15
16 document.querySelector("head style").textContent += pageCss;
```



Beispiel ColumnOptionsComponent

Controller

SelectController

Provides the functionality to manage a select input component.

Returns: SelectComponent

Function	Type	Description
getId()	String	Provides a unique id for each component.
getNumberOfColumns()	Number	Provides the fixed number of columns to show in the ui.
getInputController()	SimpleInputController	Provides the functionality of the text input hidden behind the component. This input will send the data in the form.
isCursorPositionWithSelection()	Boolean	Checks if the keyboard interaction also changes the selection. If false the cursor position for the keyboard acts like the highlight for the mouse.
isRequired()	Boolean	Checks if an option has to be chosen in a form.
setRequired(Boolean)	void	Changes the requirement of the select input to the passed value.
onRequiredChanged(Consumer<Option>)	void	Adds a listener to the change of the requirement.
isDisabled()	Boolean	Checks if the option of the select input can be changed.
setDisabled(Boolean)	void	
onDisabledChanged(Consumer<Option>)	void	Adds a listener to the change of possibility to fill the input.
isOptionsVisible()	Boolean	Checks if the options container is visible. It contains all the category and value options. In a default select this would be the not visible part of the component in the initial state.
setOptionsVisibility(Boolean)	void	Change the visibility of the options container to the passed value.
onOptionsVisibilityChange(Consumer<Option>)	void	Adds a listener to the change of the visibility of the options container. It contains all the category and value options. In a default select this would be the not visible part of the component in the initial state.
isSelectedOptionVisible()	Boolean	Checks if the selected option container is visible. It contains only the selected option and maybe some action buttons. In a default select this would be the always visible part of the component.
setSelectedOptionVisibility(Boolean)	void	Change the visibility of the selected option container to the passed value.
onSelectedOptionVisibilityChange(Consumer<Option>)	void	Adds a listener to the change of the visibility of the selected option container. It contains only the selected option and maybe some action buttons. In a default select this would be the always visible part of the component.

Function	Type	Description
getCursorPosition()	Option	Provides the option the keyboard cursor is currently at.
setCursorPosition(Option)	void	Changes the cursor position of the keyboard to the passed option. The cursor position can only contain one option per component.
clearCursorPosition()	void	Reset the keyboard cursor to no option.
onCursorPositionChanged(Consumer<Option>)	void	Adds a listener to the change of the cursor position.
getSelectedValueOption()	Option	Provides the selected value option of the select input. This is the option that is sent in a form.
setSelectedValueOption(Option)	void	Changes the selected value option to the passed option. There can only be one option selected.
clearSelectedValueOption()	void	Reset the selected value option to no option.
getSelectedOptionOfColumns(Number)	Option	Provides the selected option of the passed column. The column 0 contains the same value as in 'getSelectedValueOption()' is provided. The other columns contain the selected category of the passed column.
getColumnOptionsComponent(Number)	Column Options-Component	Provides the column options component of the passed column with the view and functionality. The column 0 provides the value option column component. The other columns provide the category option columns of the passed column.

Parameters

Name	Type	Description
selectAttributes	SelectAttributes	Defines properties to customize the select component.
numberOfColumns	Number	Defines the number of columns in the options container.

Example

```

1 const decades = [
2   ValueOption("1990's"),
3   ValueOption("2000's"),
4   ValueOption("2010's"),
5   ValueOption("2020's")
6 ];
7 const currentYear = CategoryOption("2024");
8 const years = [
9   CategoryOption("1990"),
10  CategoryOption("1991"),
11  /* ... */
12  CategoryOption("2009"),
13  CategoryOption("2010"),
14  /* ... */
15  CategoryOption("2023"),
16  currentYear
17];
18 const selectAttributes = {
19   name : "year",
20   label: "Year",

```



```
21    isRequired: true
22 };
23 const selectController = SelectController(selectAttributes, 2);
24
25 selectController.getColumnOptionsComponent(1).onOptionSelected(
26   newDecade => {
27     selectController.getColumnOptionsComponent(0).clearOptions();
28     selectController.getColumnOptionsComponent(0).addOptions(
29       years.filter(year => year.getLabel().startsWith(
30         newDecade.getLabel().slice(0, 2)))
31     );
32 });
33 selectController.getColumnOptionsComponent(0).onOptionSelected(
34   newDecade => alert("You chose the year: " + newDecade.getLabel()));
35
36 selectController.getColumnOptionsComponent(1).addOptions(descades);
37 selectController.getColumnOptionsComponent(0).addOptions(years);
38
39 selectController.getColumnOptionsComponent(0)
40   .setSelectedOption(currentYear);
```