

# Open-Source Web UI Toolkit

## IP6 Projektarbeit

Dusan Misic

Fabian Häfliger

Brugg – 20 August 2021





---

## Management Summary

Das UI-Toolkit soll die Entwicklung des Frontend vereinfachen und unabhängig eines Frameworks sein, sodass es effizient, sicher und kostengünstig eingesetzt werden kann. Die Gestaltung der Komponenten soll eine hohe Qualität an Usability, User Experience und Interaktion in verschiedensten Domänen wie Business-Controls oder Engineering-Controls aufweisen. Dies bedeutet, dass die Arbeit organisatorische, wissenschaftliche, gestalterische und technische Herausforderung enthält. Das Toolkit soll auch für zukünftige Arbeiten erweiterbar und leicht anpassbar sein. Die Modularität und Einfachheit für den Anwender sollen ebenfalls gewährleistet sein. Durch verschiedene Ansätze sind wir zu einer optimalen Lösung gekommen, die alle oben genannten Aspekte zufriedenstellend lösen.

Der Auftraggeber erhält mit unserer Arbeit eine solide Grundlage für die Weiterentwicklung des Toolkits. Mit den erarbeiteten Komponenten und dem Proof of Concept wurde gezeigt, dass moderne Frontend-Applikationen mithilfe des Projektor-Patterns effizienter gebaut werden können. Unsere Komponenten wurden iterativ und nutzergerecht entwickelt und verbessert. Die Auslieferung des Toolkits kann mittels CDN und Downloadables geschehen. Das Toolkit wird Entwicklern als Bibliothek und Opensource Repository zur Verfügung gestellt, sodass es optimal in ihre Applikation eingebaut werden kann.



# Inhaltsverzeichnis

Kapitel 1 – Einführung.....	2
1.1 Motivation.....	2
1.2 Ziele .....	2
1.3 Problem Statement.....	2
1.4 Scope.....	2
1.5 Schlüsselbegriffe.....	3
2. Gestaltung .....	5
2.1 Zielgruppe.....	5
2.2 Branding.....	5
2.2.3 Namen.....	5
2.2.3 Logo.....	6
2.3 Farbkonzept.....	6
2.3.1 Kriterien.....	7
2.3.2 Farbauswahl .....	7
2.3.3 Zusammenfassung.....	10
2.4 Komponenten.....	18
2.4.1 Analyse Usability Test.....	19
2.4.2 Login Screen .....	27
2.4.3 Register Screen .....	31
2.5 Code.....	34
2.5.1 CSS .....	34
2.5.2 Anpassbarkeit.....	34
3. Verpackung, Verbreitung, Strukturierung und Implementierung.....	35
3.1 Verpackungsmöglichkeiten.....	35
3.1.1 Framework.....	35
3.1.2 Library.....	36
3.1.3 Fazit.....	36
3.2 Verbreitungsmöglichkeiten.....	37
3.2.1 NPM.....	37
3.2.2 CDN .....	38
3.2.3 Downloadables.....	40
3.2.4 Fazit.....	40
3.3 Strukturierung des Codes.....	41

3.3.1 Web Components.....	41
3.3.2 HTML, CSS, JS .....	47
3.3.3 Projektor-Pattern.....	48
3.3.4 Fazit.....	48
3.4 Projektor-Pattern: Aufbau und Implementation.....	49
4. Proof of Concept .....	58
5. Testing.....	61
5.1 Test-Framework.....	61
Suite .....	61
5.2 Testbeispiele der Login Komponente .....	65
5.2.1 Setup .....	65
5.2.2 Tests .....	66
5.2.3 Ausgabe auf dem Browser .....	68
6. Weiteres Vorgehen.....	69
6.1 Mehr Komponenten aus verschiedenen Domänen.....	69
6.2 Veröffentlichbarkeit des Projektor-Pattern als CDN .....	69
6.3 Kompatibilität mit existierenden Frameworks und Libraries .....	69
6.4 Open Source Lizenz.....	69
6.5 ReadMe.md für Release.....	69
7. Reflexion.....	70
7.1 Ergebnisse.....	70
7.1.1 Design-Guide .....	70
7.1.2 Variantenvergleich Auslieferung / Verpackung.....	70
7.1.3 Projektor-Pattern.....	70
7.1.4 Proof of Concept.....	70
7.1.5 Testframework .....	70
7.2 Erkenntnisse.....	71
7.2.1 Erstellen des Design-Guides.....	71
7.2.2 Verteilung / Verpackung.....	71
7.2.3 JavaScript ES6 .....	71
Anhang A – Kontaktdaten .....	72
Anhang B – Ehrlichkeitserklärung.....	73
Glossar .....	74
Akronyme .....	76

Linksammlung: .....	77
Abbildungsverzeichnis.....	78
Quellenverzeichnis.....	81





# Kapitel 1 – Einführung

## 1.1 Motivation

Unser Toolkit soll eine weitreichende Variation an Komponenten in verschiedenen Domänen bereitstellen, sodass eine Benutzeroberfläche einfach und kostengünstig angefertigt werden kann. Viele bestehende Frameworks und Bibliotheken bergen bestimmte Risiken, da sie mit vielen Abhängigkeiten verknüpft sind oder dem Entwickler zu wenig Freiheit anbieten.

## 1.2 Ziele

Das Ziel der Arbeit ist der Aufbau eines hochwertigen, unabhängigen Open-Source Web UI Toolkits, das allen Web-Entwicklern als freies Repository zur Verfügung steht. Die enthaltenen Komponenten sollen höchsten Ansprüchen genügen, was die Gestaltung und die Usability angeht. Ebenfalls wird die innere Strukturierung, die Testbarkeit, die Haltbarkeit, die Konsistenz sowie eine verständliche Dokumentation forciert, damit das Toolkit erweitert werden kann.

Es soll gezeigt werden, dass Komponente einfach eingesetzt werden können und dem Entwickler ein bestimmtes Mass an Freiheit gegeben wird, was den Designaspekt angeht.

## 1.3 Problem Statement

Für die Webentwicklung existieren heute eine weitreichende Anzahl von Libraries und Frameworks bereit, um Webapplikation, Webservices und Webseiten aufzubauen. Viele dieser Frameworks und Libraries sind jedoch nur im Zusammenhang mit anderen Technologien verwendbar oder bringen anderweitige Abhängigkeiten mit sich. Dies kann sicherheitstechnische Risiken nach sich ziehen. Ebenfalls besteht ein Mangel an Flexibilität für den Entwickler. Viele bestehende Toolkits lassen sich nicht individuell anpassen und sind somit für gewisse Projekte unbrauchbar.

## 1.4 Scope

Mit unserer Arbeit soll die Grundlage für das Web-Ui-Toolkit geschaffen werden. Wir sind verantwortlich für gestaltungstechnische Aufgaben wie Farbkonzept, Logo und User Experience. Ebenfalls müssen wir Entscheidungen zur Architektur von zukünftigen Komponenten festlegen sowie diese dokumentieren und festhalten.

Der Umfang unserer Arbeit beinhaltet ein Designkonzept, eine Login Komponente, eine Register Komponente sowie ein Repository mit unserem Code.

## 1.5 Schlüsselbegriffe

In diesem Abschnitt erklären wir Begriffe, welche im Zusammenhang mit unserer Arbeit stehen.

### **CDN**

Ein CDN (Content-Delivery-Network) ist ein verteiltes und über das Internet verbundenes Verteilnetzwerk, über welche Dateien ausgeliefert werden können. Wird in der Softwareentwicklung rege genutzt.

### **CSS**

Cascading Style Sheets ist eine Stylesheet Sprache, welche zur Gestaltung von Webseiten oder elektronischen Dokumenten verwendet wird.

### **Darkmode**

Viele heutige Betriebssysteme, Apps und Webseiten bieten einen Darkmode an, welcher Hintergründe dunkel und Text hell darstellt. Damit kann der Akku eines Gerätes geschont werden sowie die Gesundheit der Augen.

### **Framework**

Ein Gerüst für die Softwareentwicklung. Enthält Module, APIs und/oder Vorlagen, mit welcher die Softwareentwicklung vereinfacht werden kann.

### **Input Feld**

Input Feld werden in Formularen verwendet, damit der Benutzer Daten eingeben kann. Es gibt verschiedene Typen von Inputfeldern wie E-Mail, Text oder Passwort.

### **Interface Design**

Mit Interface Design wird das Design von Schnittstellen zwischen Menschen und Maschine bezeichnet.

### **Library**

Eine Sammlung von Unterprogrammen, welche Funktionen anbietet. Eine Library ist kein eigenständiges Programm.

### **Lifecycle Methods**

Als Lifecycle Methods werden bestimmte Funktionen oder Ausführungen genannt, die zu bestimmten Lebensabschnitte einer Komponente oder Software durchgeführt werden. Solche Lebensabschnitte können Ereignisse, wie das Erstellen, Verändern oder Löschen einer Komponente sein.

### **Lightmode**

Das Gegenteil von einem Darkmode. Der Hintergrund ist hell und der Text ist dunkel eingestellt.

### **Modularität**

In der Softwareentwicklung bezeichnet Modularität die systematische Aufteilung des Programmes in einzelne Teilblöcke.

### **Node.js**

Eine JavaScript Laufzeitumgebung, welche es ermöglicht, JavaScript ausserhalb eines Webbrowsers auszuführen.

### **NPM**

NPM ist ein Paketmanager für Node.js, welche verwendet wird, um Softwarepakete herunterzuladen und Abhängigkeiten zu verwalten.

**Open-Source**

Software, bei welcher der Quellcode öffentlich zugänglich und von Drittpersonen verändert und genutzt werden kann.

**Paketmanager**

Ein Paketmanager ermöglicht die Verwendung, Verwaltung und Organisation von Software. Er übernimmt Aufgaben wie das Installieren, Deinstallieren und Aktualisieren von Programmpaketen.

**Repository**

Verzeichnis zur Speicherung von digitalen Objekten.

**Usability**

Usability bedeutet auf Deutsch übersetzt Benutzbarkeit, Benutzerfreundlichkeit. Damit wird beschrieben, ob ein Nutzer sein Ziel effektiv, effizient und zufriedenstellend erreichen kann.

**Weissraum**

Als Weissraum wird im Interface Design der Bereich bezeichnet, welcher sich horizontal oder vertikal um ein Element befindet und ungenutzt ist.

## 2. Gestaltung

Dieses Kapitel behandelt den gestalterischen Teil unserer Arbeit. Es wird aufgezeigt, anhand welcher Kriterien das Farbkonzept erarbeitet wurde und wie die einzelnen Komponenten aussehen sollen. Ebenfalls wurde ein Logo sowie einen Namen für das Toolkit gesucht. Darauffolgend erklärt wird, wie diese Designvorgaben im Code respektive genauer gesagt, in CSS, umgesetzt wurden. Zuletzt wird aufgezeigt, dass Entwickler das Toolkit in Zukunft individuell anpassen können und somit ihre eigenen Designvorgaben mit den im Toolkit enthaltenen Komponenten realisieren können.

### 2.1 Zielgruppe

Das langfristige Ziel des Auftraggebers ist es, dass mit dem UI-Toolkit Anzeigen aus verschiedenen Bereichen, wie Business, Wissenschaft oder Ingenieurwesen erzeugt werden können. Wie im Kapitel Einleitung erklärt wurde, gehören zu dieser Arbeit organisatorische, wissenschaftliche, gestalterische und technische Herausforderungen. Durch diese Arbeit sollte eine solide Grundlage für ein erfolgreiches Toolkit gelegt werden. Aus diesem Grund musste geklärt werden, an welche Zielgruppe sich das Toolkit genau wendet. Klar ist, dass schlussendlich Webentwickler unser Toolkit verwenden sollen, jedoch sind die Nutzer der jeweiligen Komponenten bei der Komponentengestaltung im Fokus.

Da neben den auf spezifische Domänen ausgerichteten Anzeigen auch allgemeine Komponenten zur Verfügung stellen sollen, wurde der Fokus in dieser Arbeit auf diese Komponenten gelegt. Zu allgemeinen Komponenten gehören sowohl Login- als auch Registrierungskomponente.

Aus diesem Grund wurde für gestalterische Fragen als Zielgruppe gewöhnliche Verwender des Internets gewählt. Ein gewöhnlicher Benutzer des Internets hat keinen Hintergrund in Webentwicklung oder Webdesign. Er benötigt Feedback der Applikation, welche ihm das Nutzererlebnis erleichtert. Unsere Zielgruppe erwartet sichere und vertraute Anzeigen, welche seine Anforderungen erfüllen.

### 2.2 Branding

Das Toolkit benötigt neben eines Farbschemas auch einen Namen sowie ein Logo, damit es entsprechend vermarktet werden kann und einen entsprechenden Wiedererkennungswert hat.

#### 2.2.3 Namen

Um einen passenden Namen zu finden, wurde überlegt, was das gewünschte Ziel des Toolkits ist und wie diese mit Adjektiven beschrieben werden können. Dabei konnten die folgenden Adjektive mit dem Toolkit assoziiert werden.

- Flexibel
- Einfach
- Kostengünstig
- Leicht

Bei der Suche nach einem passenden Namen wurde der Fokus vor allem auf die Tierwelt gelegt, da ursprünglich das Toolkit vom Auftraggeber "Butterfly" genannt wurde. Für den Namen des Toolkits wurde schlussendlich Kolibri gewählt. Kolibris sind Vögel, welche im Volksmund durch Ihre hohe Kadenz an Flügelschlägen pro Minute bekannt sind. Ein Kolibri verfügt über enorm bewegliche Flügel, was ihn zusätzlich rückwärts und seitwärts fliegen lässt. Sie sind sehr kleine und leichte Vögel, der kleinste Kolibri wird nur 6 cm gross. Der Kolibri kann sehr schnell fliegen. Durch seine Kompaktheit kann er hohe Geschwindigkeiten erreichen.

Der Name Kolibri passt zum Toolkit, da es eine vielseitige Anzahl aus Komponenten aus verschiedenen Domänen anbieten soll. Es soll flexibel sein und einfach. Der Entwickler soll es mühelos verwenden können. Da das Toolkit auch keinerlei Abhängigkeiten mit sich bringen soll, ist es dementsprechend sehr leicht. Diese Attribute repräsentiert der Kolibri. Er ist beweglich und flexibel, da er in jede Richtung fliegen kann. [1]



### 2.2.3 Logo

Basierend auf dem Namen wurde ein Logo für das Toolkit entworfen. Nach anfänglichen Skizzen ist das in der Abbildung 1 ersichtliche Logo entworfen worden. Kolibris erscheinen in der Natur in vielen verschiedenen Farben. Wieso das Logo in diesen Blau- und Violetttönen gehalten wurde, wird im nächsten Abschnitt dargelegt.

Abbildung 1: Kolibri Logo

## 2.3 Farbkonzept

Farben spielen eine wichtige Rolle für die User Experience. Sie können Inhalte für den Nutzer angenehmer machen und geben ihnen mehr Persönlichkeit. Mit einer passenden Farbwahl kann die Marke des Produktes enorm gestärkt werden und ein hoher Wiedererkennungswert erreicht werden. [2]

Da das Ziel ist, im Toolkit einen Light- sowie einen Darkmode anzubieten, wurden zwei verschiedene Farbschemas aufgebaut. Die Anforderungen an einzelne Farben sind für einen Darkmode anders als bei einem Lightmode.

### 2.3.1 Kriterien

Um ein passendes Farbkonzept zu erstellen, wurden die folgenden Kriterien definiert. Alle Entscheidungen hinsichtlich des Farbkonzeptes wurden aufgrund dieser Kriterien validiert.

#### **Grundlegendes Ziel der Farbgebung**

Welche Stimmung soll dem Nutzer beim Verwenden unserer Komponenten vermittelt werden? Welche Emotionen sollen damit beim Nutzer ausgelöst werden?

#### **Sehchwächen (Farbenfehlsichtigkeit)**

Es gibt viele spezifische Arten von Sehchwächen. Darunter gehören Schwachsichtigkeit, Fehlsichtigkeit oder auch ein Glaukom, auch grüner Star genannt. Der Schwerpunkt wurde in dieser Arbeit auf Farbfehlsichtigkeiten gelegt. Diese sind eine Kategorie der Fehlsichtigkeit.

Menschen, welche an Farbfehlsichtigkeit leiden, haben eine eingeschränkte Farbwahrnehmung. Statistisch gesehen leiden Männer mehr an Farbfehlsichtigkeit als Frauen. Als Bekannte unter den Farbfehlsichtigkeiten gelten Protanopie (Rotblindheit), Deutanopie (Grünblindheit), Tritanopie (Blaubindheit) und Achromatopsie (totale Farbenblindheit). [3][4]

#### **Usability**

Im Interface Design können dem Nutzer durch farbliche Kodierung Hinweise auf Status oder die Funktionalität eines Elements gegeben werden. Beispielsweise kann ein Button, welcher inaktiv ist, in einem grauen Farbton dargestellt werden. Dem Nutzer wird damit signalisiert, dass er diesen nicht verwenden kann.

#### **User Experience**

Mit einer schönen und sauberen Farbwahl können dem User die gewünschten Gefühle und Emotionen vermittelt werden. Das Nutzererlebnis kann damit merklich verbessert werden. Farben spielen eine massgebliche Rolle dabei, wie ein Nutzer eine Oberfläche wahrnimmt.

#### **Normengerechte Farbgestaltung**

Kulturelle Farbkonventionen sind bei der Farbwahl ebenfalls zu beachten. Zum Beispiel wird in unserem Kulturraum die Farbe Rot häufig für Gefahr verwendet. In asiatischem Kulturraum wird Rot jedoch an Glück zugewiesen.

### 2.3.2 Farbauswahl

Im folgenden Abschnitt wird aufgezeigt, wie das Farbkonzept aufgebaut wurde. Die selektierten Farben werden danach anhand der im vorherigem Abschnitt 2.3.1 definierten Kriterien validiert.

#### *2.3.2.1 Primärfarben*

Als Primärfarbe wird die dominierende Farbe in einem Farbschema bezeichnet. Da als Nutzergruppe für die zu erstellenden Komponenten gewöhnliche Benutzer gewählt wurden, wurde ein Moodboard erstellt. Das Moodboard sollte die Nutzergruppe mit den Adjektiven, welche unser Toolkit repräsentieren, verbinden. Bei der darauffolgenden Selektierung wurden diese beiden blauen Farbtöne aus dem Moodboard ausgewählt.

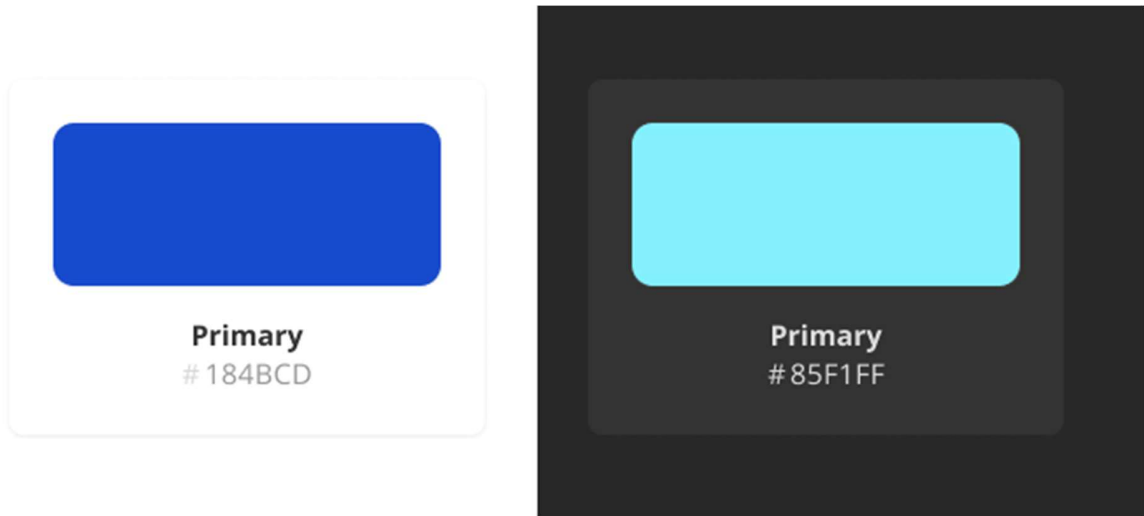


Abbildung 2: Primärfarbe Light- und Darkmode

### 2.3.2.2 Graustufen

Ein gutes Farbkonzept verfügt über mehrere Graustufen, welches mehrere Rollen im Interface Design erfüllt. Grau wird verwendet, um Hierarchien festzulegen und das sekundäre Merkmale der Oberfläche in den Hintergrund treten. Mit Grau kann Raum im Interface Design geschaffen werden, mit welchem Inhalte atmen können. [5][6]

Die folgenden Farbtöne wurden für den Lightmode ausgewählt.

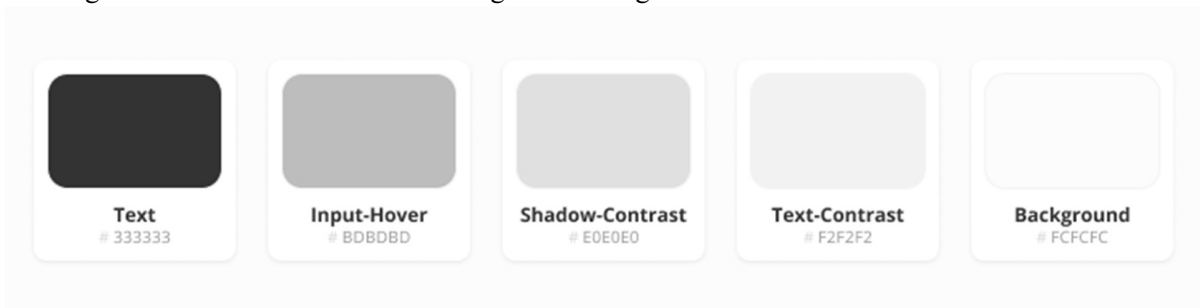


Abbildung 3: Graustufen im Lightmode

Die Grautöne im Darkmode sind wie folgt.

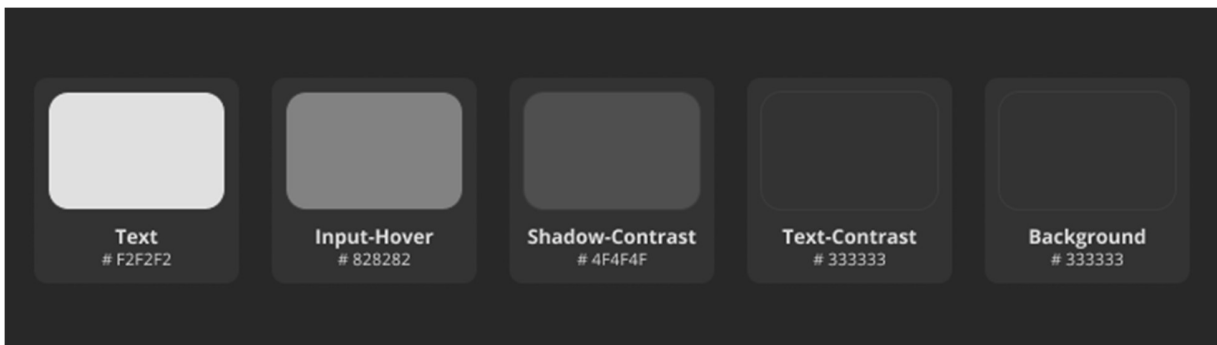


Abbildung 4: Graustufen im Darkmode



Die verschiedenen Grautöne werden für die folgenden Aufgaben verwendet.

### **Text**

Dieser Grauton wird für Text verwendet.

### **Input-Hover**

Ein Grauton, welcher verwendet wird, um bei Inputfelder den Zustand "Hover " darzustellen.

### **Shadow-Contrast**

Ein Grauton für Schatten und Kontrast

### **Text-Contrast**

Ein Grauton, um bei Elementen mit dunklem oder hellem Hintergrund Text darzustellen.

### **Background**

Generelle Hintergrundfarbe.

### *2.3.2.3 Hilfsfarben*

Um das Farbkonzept zu komplettieren, wurden zusätzliche Farben wie "Success", "Warning" und "Error" ausgewählt. Diese Farben sind wichtig für ein Farbschema, da sie gebraucht werden, um den Nutzer Feedback zu geben oder kulturellen Farbkonventionen zu folgen.

Die Farben für den Lightmode sind die folgenden.

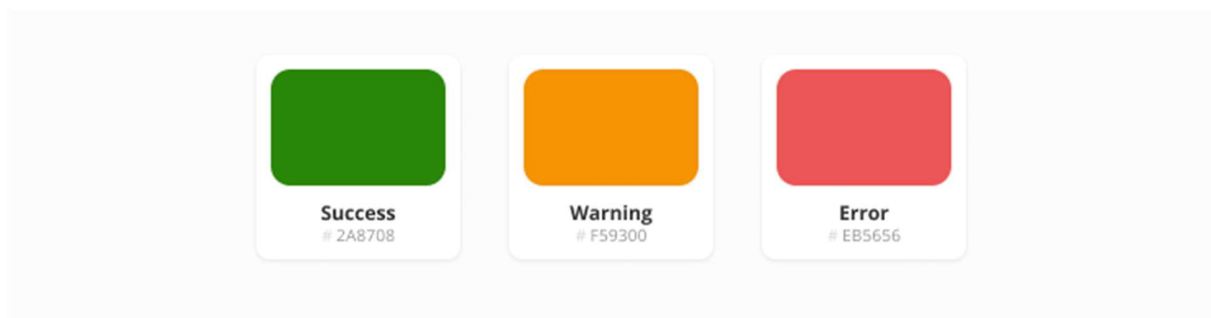


Abbildung 5: Hilfsfarben im Lightmode

Für den Darkmode wurden die folgenden Farben gewählt.

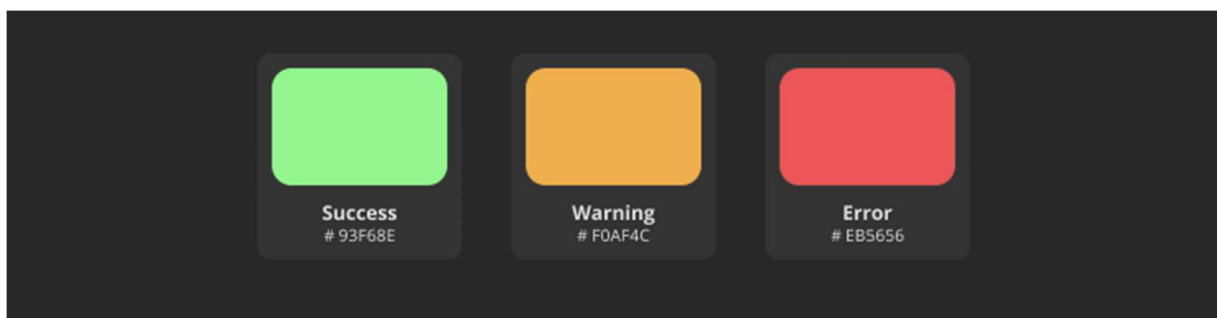


Abbildung 6: Hilfsfarben im Darkmode

### 2.3.3 Zusammenfassung

Grundsätzlich handelt es sich beim Farbschema um ein monochromatisches Farbschema. Es wurde eine Primärfarbe, welche in der praktischen Umsetzung mit unterschiedlicher Deckkraft verwendet werden kann, damit einzelne Elemente mehr oder weniger Meinung gegeben werden kann. Mit den ergänzenden Grautönen und Hilfsfarben konnte das Farbschema komplettiert werden.

Die ausgewählten Farben wurden aufgrund der oben erklärten Kriterien validiert. Auch wurde das Feedback von unserem Kunden sowie eines Usability Tests hinzugezogen. Die Resultate des Usability Tests werden in einem nachfolgenden Kapitel analysiert. Ein Link zur Auswertung mit allen Antworten befindet sich in der Linksammlung.

#### 2.3.3.1 Grundlegendes Ziel der Farbgebung

Grundsätzlich gilt blau sowohl bei Männern als auch bei Frauen als die beliebteste Farbe. Der Farbe Blau wird eine beruhigende und ausgleichende Wirkung zugeschrieben. Bildlich verbinden Menschen Blau mit Wasser, Schnee, Eis und dem Ozean. Dadurch wird die Farbe auch als kalt wahrgenommen. Jedoch wird Blau aus diesem Grunde als frisch, ruhig und tief empfunden. [7]

Gemäss einer 2003 als Teil einer Bachelorarbeit durchgeführten Umfrage wird die Farbe Blau im Vergleich mit anderen Farben mit den folgenden Attributen assoziiert. [8]

- 34% Vertrauen
- 28% Sicherheit
- 42% Zuverlässigkeit / Verlässlichkeit

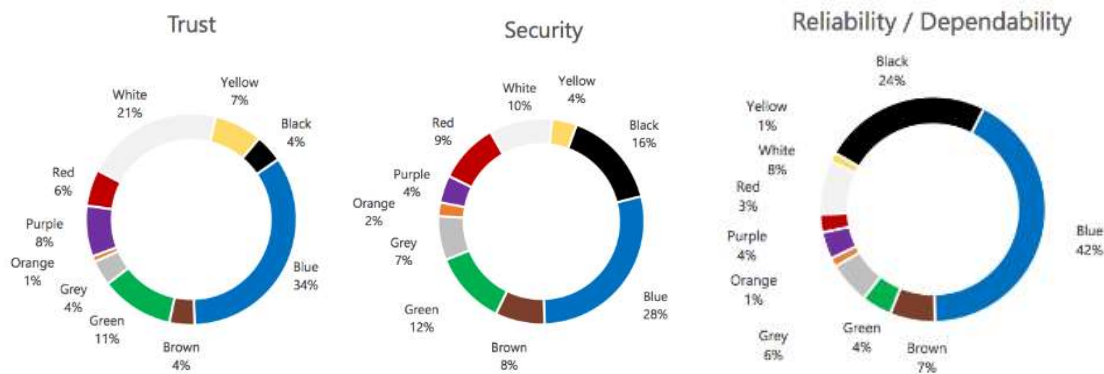


Abbildung 7: Umfrageergebnisse aus Quelle [8]

Häufig haben Banken oder Universitäten ein blaues Logo. Im Finanzbereich ist es von Bedeutung, dass die Firma als sicher und vertrauenswürdig angesehen wird. Auf den Abbildungen 8 und 9 wird dies veranschaulicht.



Abbildung 8: Logos von bekannten Finanzinstituten [9]



Abbildung 9: Credit Suisse Logo [10]

Damit trifft die Farbe genau die Emotionen, welche das Toolkit vermitteln möchte. Als Nutzer will man vertrauenswürdige und sichere Anzeigen. In der von uns angezielten Domänen ist es von grundlegender Bedeutung, dass dem Nutzer ein sicheres und verlässliches Nutzererlebnis ermöglicht wird.

### 2.3.3.2 Sehschwächen

Unsere Farben wurden mit einem Plug-In im Designtool Figma auf die üblichsten Sehschwächen validiert. Mit dem Plug-In kann man eine Farbe auswählen und es werden für acht verschiedene Farbfeldsichtigkeiten die Farbtöne zurückgegeben. Anhand dieser Auswertungen kann das Farbschema aus der Perspektive einer Person mit Sehschwäche angesehen werden. [11]

In den folgenden Abbildungen sieht man, wie sich die Primärfarbe sowie Hilfsfarben unter den einzelnen Farbfeldsichtigkeiten verändern. Je nach Farbfeldsichtigkeit ist es äussert schwierig, ausser leichten Unterschieden überhaupt festzustellen, ob es sich nun um eine Primärfarbe oder eine Hilfsfarbe handelt. Wie schon im Abschnitt 2.3.1 erläutert wurde, wurde der Schwerpunkt auf die häufigsten Farbfeldsichtigkeiten Protanomalie, Deuteranomalie und Tritanomalie gelegt. Bei allen drei ist der farbliche Unterschied genug gross, um eine Differenz festzustellen. Vor allem wichtig war es, dass zwischen "Error" und "Warning" eine genug grosse farbliche Differenz besteht.



Abbildung 10: Vergleich Farbfelsichtigkeit der Primärfarbe im Lightmode



Abbildung 11: Vergleich Farbfelsichtigkeit der Hilfsfarbe «Success» im Lightmode



Abbildung 12: Vergleich Farbfelsichtigkeit der Hilfsfarbe "Warning" im Lightmode



Abbildung 13: Vergleich der Farbfelsichtigkeit der Hilfsfarbe "Error" im Lightmode

Natürlich ist der Darkmode ebenfalls auf Farbfelsichtigkeiten geprüft worden. Bei der Farbauswahl für den Darkmode war es schwerer, die passenden Farben zu wählen. Vor allem mit grünen und orangen Farbtönen sind schwierig zu wählen und abzustimmen.

Schlussendlich wurde eine gute Lösung gefunden, wobei einzig die "Success" Farbe bei Tritanomalie ähnlich erscheint wie die Primärfarbe. Dies kann beim Nutzer zu Verwirrung führen, jedoch wurde dies nicht als gravierender Fehler betrachtet. Es wäre ein grösseres Problem, wenn die Primärfarbe ähnlich wie die "Error" Farbe wäre. Relativiert wird dies ebenfalls auch dadurch, dass nicht nur auf farblicher Ebene dem Nutzer ein Feedback gegeben wird. Die Komponente geben bei Fehlern ebenfalls ein textliches Feedback zurück. [12]



Abbildung 14: Vergleich Farbfeldsichtigkeit der Primärfarbe im Darkmode



Abbildung 15: Vergleich Farbfeldsichtigkeit der Hilfsfarbe "Success" im Darkmode



Abbildung 16: Vergleich Farbfeldsichtigkeit der Hilfsfarbe "Warning" im Darkmode

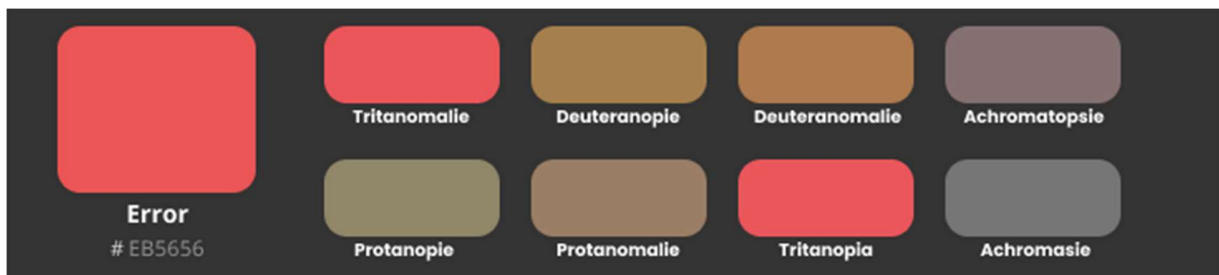


Abbildung 17: Vergleich Farbfeldsichtigkeit der Hilfsfarbe "Error" im Darkmode

## Kontrast

Um zu testen, ob beide Primärfarben genug Kontrast auf schwarzen und weissen Hintergrund haben, wurde ebenfalls ein hilfreiches Plug-In im Design-Tool Figma verwendet. Mit dem Stark-Plugin kann der Kontrast einer Farbe zum jeweiligen Hintergrund geprüft werden. Zusätzlich wird angezeigt, ob normaler oder grosser Text lesbar ist. [13]

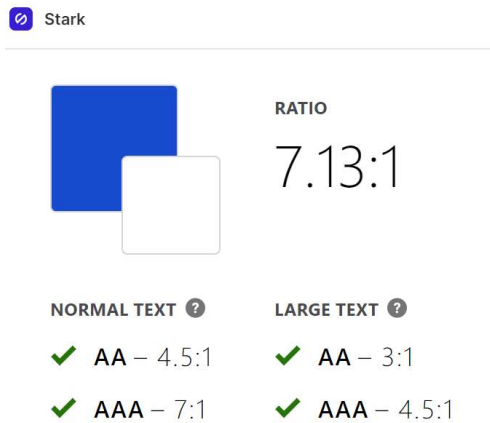


Abbildung 18: Stark Auswertung Primärfarbe Lightmode

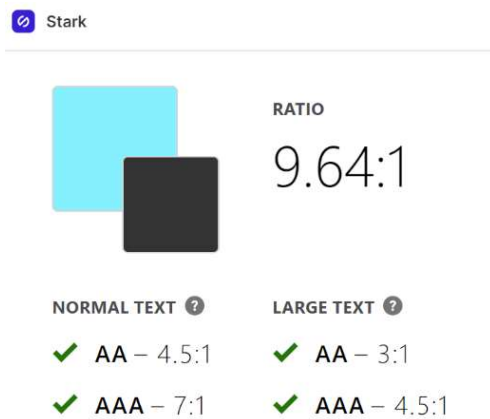


Abbildung 19: Stark Auswertung Primärfarbe Darkmode

Wie in der Abbildungen 18 und 19 ersichtlich ist, haben beide Primärfarben einen genügend hohen Kontrast zu den gewählten Hintergrundfarben. Dies sieht man an allen grünen Haken.

Eine Graustufe wurde für Text definiert. Damit kann das Auge des Nutzers entlastet werden, da nicht der ganze Text in weiss dargestellt wird. Um den Kontrast festzustellen, wurde auch diese Farbe mit dem Stark Plug-In geprüft. Wie in den Abbildungen 20 und 21 ersichtlich, können beide Farben ohne Probleme gelesen werden. Der Kontrast zum Hintergrund ist genug gross, damit diese lesbar ist.

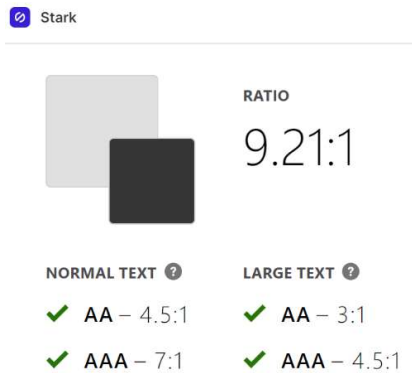


Abbildung 20: Auswertung Stark- Plug-In Darkmode

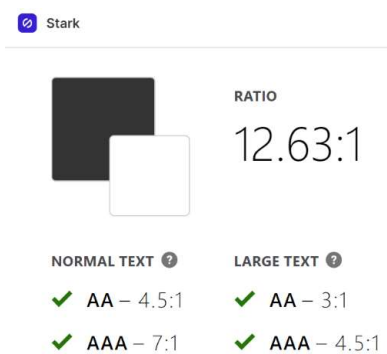


Abbildung 21: Auswertung Stark- Plug-In Lightmode

Somit ist das Farbschema auch für Menschen, welche unter bestimmten Farbfehlsichtigkeiten leiden, geeignet. Mithilfe des Color Blind Plug-In sowie des Stark Plug-In konnten die Farbtöne auf die Verträglichkeit mit Farbfehlsichtigkeiten sowie zu Ihrem Kontrast zum Hintergrund geprüft werden.

### 2.3.3.3 Usability

Durch Farbabstufung können verschiedene Zustände bei unseren Komponenten erreicht werden. Als Beispiel wird ein Primärbutton genommen. Um ihn als inaktiv darzustellen, wird die Deckkraft des Farbtons verringert. Der Nutzer nimmt dadurch wahr, dass dieser Button inaktiv ist und somit nicht klickbar. Falls der Nutzer mit seiner Maus über den Button schwebt, wird der Button leicht heller. Im gedrückten Zustand wird ein innerer Schatten dem Button hinzugefügt. Der Nutzer sieht somit, dass seine Aktion eine Reaktion nach sich zieht. Dies ist in Abbildung 22 ersichtlich.

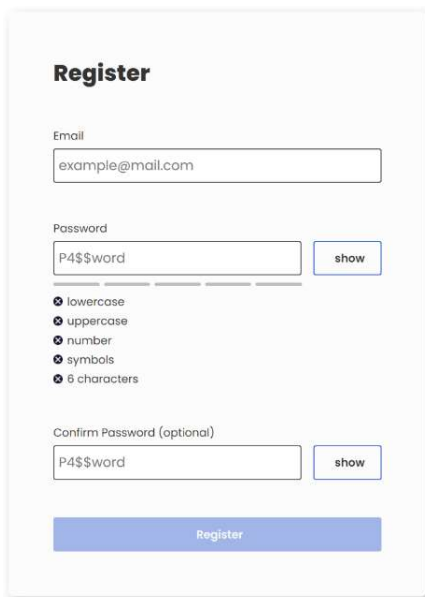


Abbildung 22: Zustände Primärbuttons

Eine verbesserte Usability ist damit durch das Farbschema gegeben. Entwickler, die das Toolkit verwenden, können Elemente hervorheben, verschiedene Zustände anzeigen oder die Aufmerksamkeit des Nutzers subtil auf bestimmte Elemente leiten.

### 2.3.3.4 User Experience

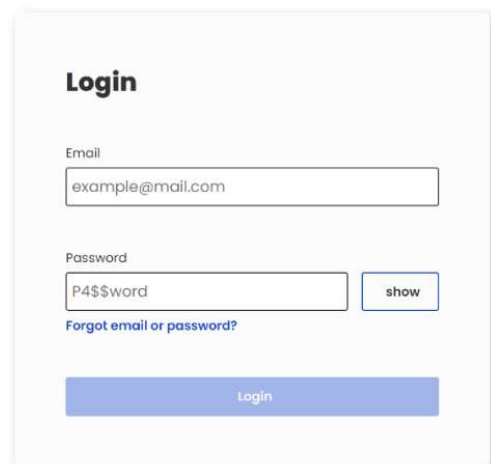
Durch unsere Farbwahl wird versucht, dem Nutzer Verlässlichkeit, Einfachheit und Sicherheit zu vermitteln. Dies ist uns unserer Ansicht nach gelungen. Die folgenden Abbildungen zeigen einige Beispiele von unseren Komponenten.



The screenshot shows a 'Register' form with the following elements: a title 'Register', an 'Email' input field containing 'example@mail.com', a 'Password' input field containing 'P4\$\$word' with a 'show' button, a list of password requirements (lowercase, uppercase, number, symbols, 6 characters) each with a checked radio button, a 'Confirm Password (optional)' input field containing 'P4\$\$word' with a 'show' button, and a blue 'Register' button at the bottom.

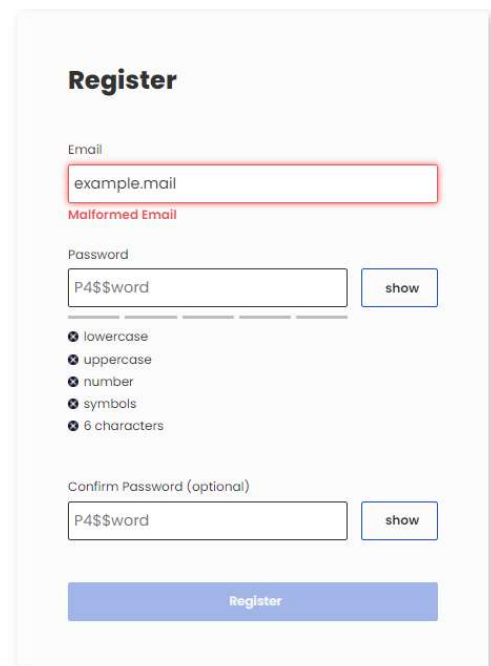
Abbildung 24: Register Komponente

Die Komponenten sind organisiert und sauber. Das User Interface ist konsistent und vermittelt dem Nutzer, dass diese Komponente sich entsprechend verhält und auf die Inputs des Users reagiert. In Abbildung 25 ist ersichtlich, wie eine Eingabe des Nutzers validiert wird und die Komponente entsprechend darauf reagiert. Die Farbwahl zeigt dem Nutzer an, dass seine Eingabe inkorrekt ist und entsprechend korrigiert werden muss.



The screenshot shows a 'Login' form with the following elements: a title 'Login', an 'Email' input field containing 'example@mail.com', a 'Password' input field containing 'P4\$\$word' with a 'show' button, a blue link 'Forgot email or password?', and a blue 'Login' button at the bottom.

Abbildung 23: Login Komponente



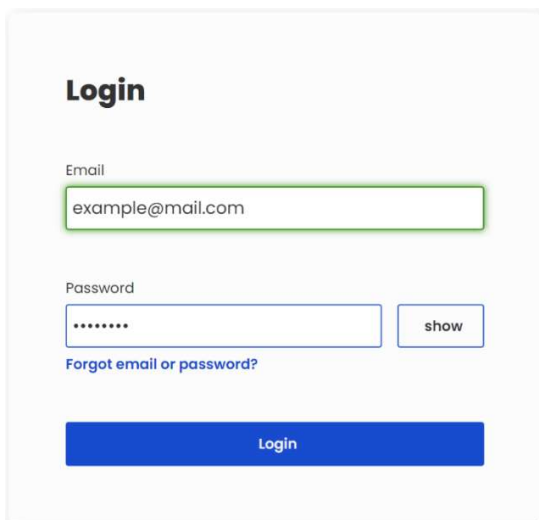
The screenshot shows the 'Register' form with a validation error. The 'Email' input field contains 'example.mail' and is highlighted with a red border. Below the input field, the text 'Malformed Email' is displayed in red. The rest of the form, including the password fields and requirements, is identical to the previous screenshot.

Abbildung 25: Register Komponente falsche E-Mail Eingabe



### 2.3.3.5 Normengerechte Farbgestaltung

Im westlichen Kulturraum werden die Farben Grün, Orange und Rot mit Erfolg, Warnung und Fehler/Stopp verbunden. Damit das Toolkit diese Normen erfüllen kann, wurden unsere Hilfsfarben beim Design der Komponenten für das Zeigen dieser Assoziationen verwendet. Wie in den Abbildungen 26 und 27 sichtbar ist, werden die Farben durch das gesamte Design hindurch verwendet, um den Nutzer Feedback zu geben. Wenn ein Nutzer einen E-Mail-Input falsch erfasst, gibt die Komponente dem Nutzer ein visuelles sowie textliches Feedback. Das visuelle und textliche Feedback ist anhand der kulturellen Normen kodiert und erfüllt diese somit. [14]



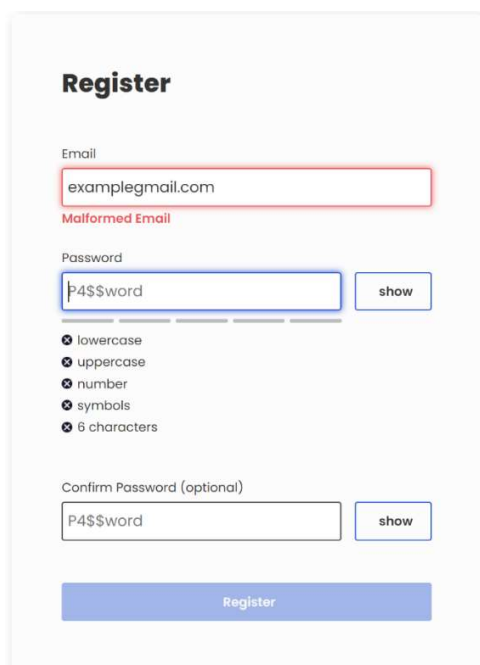
**Login**

Email  
example@mail.com

Password  
.....

[Forgot email or password?](#)

Abbildung 26: Beispielanwendung der Hilfsfarbe "Success"



**Register**

Email  
examplegmail.com  
Malformed Email

Password  
P4\$\$word

- lowercase
- uppercase
- number
- symbols
- 6 characters

Confirm Password (optional)  
P4\$\$word

Abbildung 27: Beispielanwendung der Hilfsfarbe "Error"

## 2.4 Komponenten

Nachdem festgelegt wurde, welche Komponenten zu Beginn realisiert werden, wurde mit dem Prototyping begonnen. Dabei sind Inputfelder skizziert worden und diese mit dem Kunden besprochen.

Mithilfe der erarbeiteten Farbschemas und Prototypen wurden die ersten Komponenten im Code realisiert. Diese sind in den Abbildungen 29 und 30 ersichtlich.

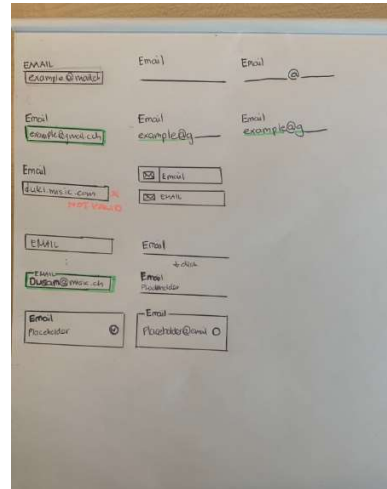


Abbildung 28: Inputfelder Skizzen

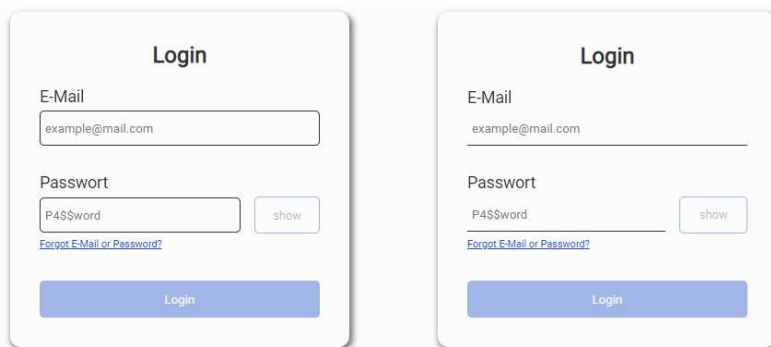


Abbildung 29: erster Login Prototyp

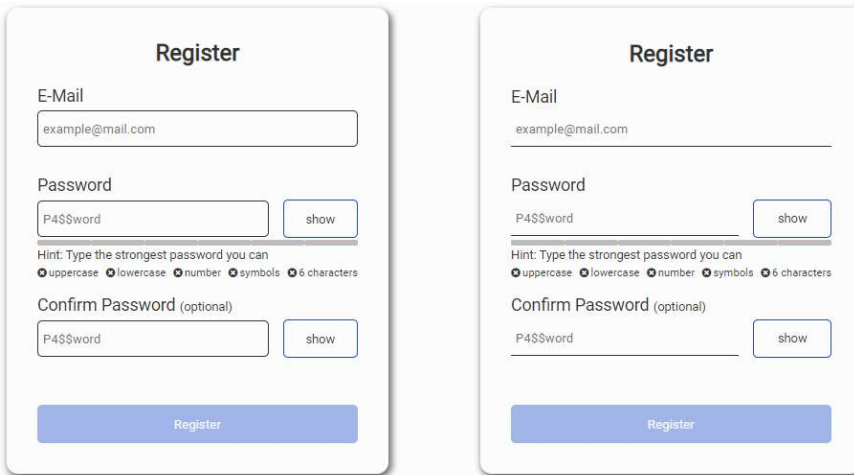


Abbildung 30: erster Register Prototyp

Um die Komponenten breiter zu testen, wurde ein Usability Test erstellt. Das Feedback des Usability Tests wurde analysiert und ebenfalls für die Weiterentwicklung der einzelnen Komponenten verwendet.

## 2.4.1 Analyse Usability Test

Der Usability Test besteht aus einem allgemeinen Teil, in welchem der Teilnehmende persönliche Fragen beantworten muss. Diese allgemeinen Fragen haben wir den Teilnehmenden gestellt, damit wir Ihr Feedback besser einordnen konnten. Für die Registrierung und Login Komponente existieren je zwei einzelne Teile, welche spezifische Fragen zu den Komponenten gestellt werden und der Teilnehmende verschiedene Tasks erfüllen muss. Zum Abschluss werden allfälligen Entwicklern gefragt, ob Sie das Toolkit verwenden würden.

Die Analyse beinhaltet eine Übersicht aller Teilnehmender des Usability Tests. Als Abschluss wird erklärt, was aufgrund der Antworten in den Komponenten verändert wurde.

### 2.4.1.1 Allgemeines

Der Allgemeine Teil besteht aus fünf Fragen zu Alter, IT-Affinität, Passwort-Manager, Login Versuche und Anzahl Stunden im Internet pro Tag.

#### Frage 1: In welcher Altersklasse befinden Sie sich?

Die Mehrheit der Befragten war zwischen 20-29 Jahren alt, da viele Mitstudierende von uns die Umfrage ausgefüllt haben. Die restlichen Befragten waren sehr gleichmässig aufgeteilt in den restlichen Altersklassen verteilt.

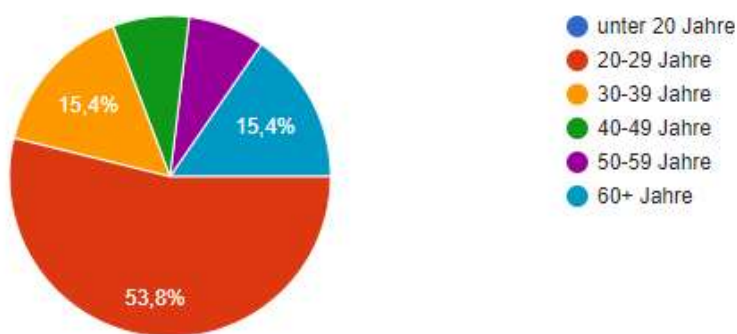


Abbildung 31: Usability Test Altersklassen

#### Frage 2: Wie hoch ist Ihre IT-Affinität (1 tief - 7 hoch)?

Auch von der IT-Affinität her war die Befragungsgruppe sehr ausgeglichen. Es wurden sowohl IT-Experten als auch völlig unerfahrene Personen befragt.

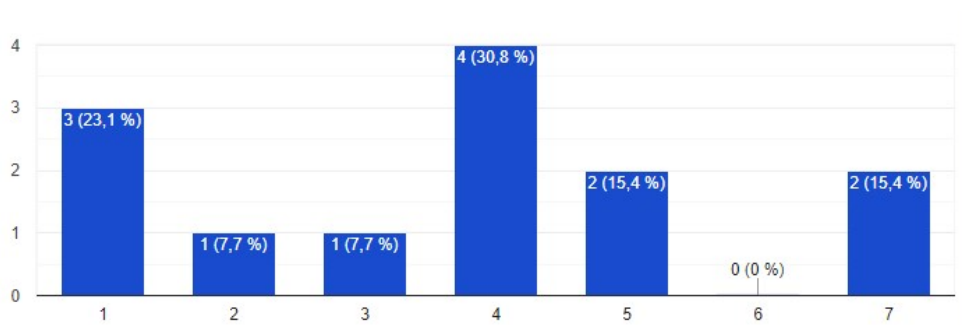


Abbildung 32: Usability Test IT-Affinität

### Frage 3: Verwenden Sie einen Passwort Manager auf Ihrem Browser?

Der Grossteil der Befragten hat entweder keine Ahnung was ein Passwort Manager ist oder verwendet keinen. Nur weniger als ein Viertel verwendet einen Passwort Manager

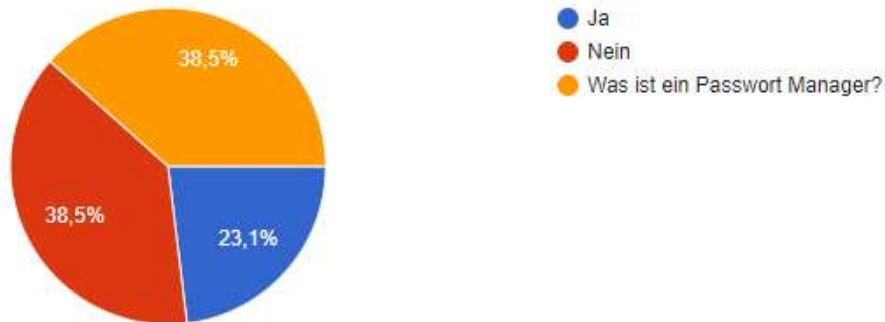


Abbildung 33: Usability Test Passwort Manager

### Frage 4: Wie oft scheitern Sie dabei sich korrekt einzuloggen?

Rund drei Viertel der Befragten benötigen einen oder zwei Versuche, um sich korrekt einzuloggen.

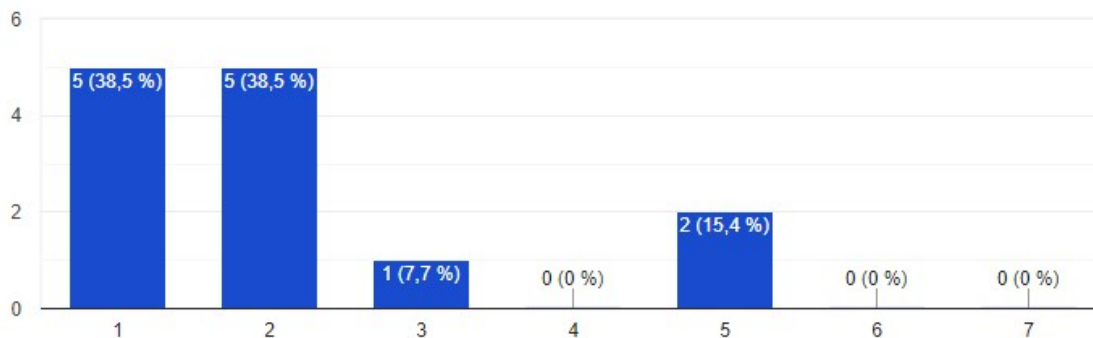


Abbildung 34: Usability Test Einloggen

### Frage 5: Wie viele Stunden verbringen sie circa pro Tag im World Wide Web?

Auch hier sind die Antworten ausgeglichen. Unsere Befragtengruppe deckt die volle Bandbreite an Internetnutzer ab.

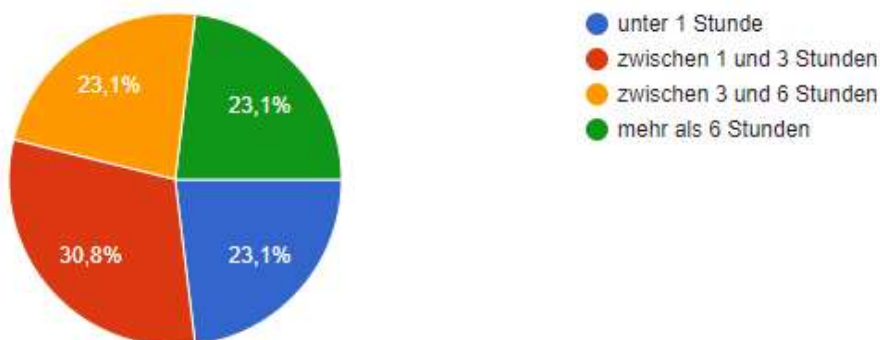


Abbildung 35: Usability Test Internetstunden

### 2.4.1.2 Registrierung

Der Teil zur Registrierungs-Komponente besteht aus sechs verschiedenen Tasks und einzelnen Zusatzfragen.

#### Task 1: Normaler Registrierungsvorgang

Task 1 bestand aus einem normalen Registrierungsvorgang. Der Teilnehmende musste uns danach mitteilen, wie er die Komponente einschätzt. Ebenfalls wurde gefragt, was Ihm Schwierigkeiten bereitet hatte und wo die Komponente verbessert werden könnte.

Auf einer Skala von 1 bis 7 wurde unsere Komponente wie folgt bewertet:

1. unverständlich	7. Verständlich	6.08
1. unangenehm	7. Angenehm	6.23
1. herkömmlich	7. modern	5.54
1. erwartungskonform	7. nicht erwartungskonform	2.85
1. übersichtlich	7. Verwirrend	1.46
1. aufgeräumt	7. Überladen	2.30
1. ineffektiv	7. effektiv	6.23

#### Gab es während dem Vorgang irgendwelche Schwierigkeiten?

- Verwirrung durch das Feedback
- Copy Paste von Passwort war nicht erlaubt

#### Was hat Ihnen besonders gut gefallen?

- Feedback
- Farbkonzept
- Unterstützung durch Kriterien
- Bedienbarkeit
- Anzeige Passwortstärke
- Layout

#### Wo sehen Sie Verbesserungspotenzial?

- Mehr Abstand
- Der Hinweis ist unnötig unter dem Passwortfeld
- Passwortinformationen grösser
- Zu gewöhnliche Farbwahl

#### Task 2: syntaktisch Falsche E-Mail

Beim Task 2 mussten die Teilnehmenden eine syntaktisch falsche E-Mail eingeben und beobachten, was passiert.

#### Hat das Feedback der Applikation geholfen den Fehler zu verstehen und zu beheben?

Ja, aber Abstände sind nicht proportional.

#### Wo sehen Sie Verbesserungspotenzial?

- Zeigen, was an Eingabe nicht korrekt ist
- Das Wort Malformed in invalid ändern
- Das Wort Malformed zu wenig geläufig
- Layout

### **Task 3: Eingabe example@gmail.com**

Die Teilnehmenden mussten im Task 3 die E-Mail [example@gmail.com](mailto:example@gmail.com) eingeben und beobachten, was passiert.

#### **Hat das Feedback der Applikation geholfen den Fehler zu verstehen und zu beheben?**

- Ja
- war eindeutig
- unmissverständlich
- klar

#### **Wo sehen Sie Verbesserungspotenzial?**

Sicherheitsdenken, wenn man erfährt das E-Mail schon verwendet wird.

### **Task 4: Eigene E-Mail**

Der Task 4 bestand daraus, dass die Teilnehmenden Ihre eigene E-Mail eingeben.

#### **Wurde Ihre Adresse erkannt? Wenn nicht bitte schreiben Sie Ihre Adresse auf.**

Ja, wurde erkannt.

#### **Wo sehen Sie Verbesserungspotenzial?**

Keines.

### **Task 5: Gültiges Passwort angeben**

Beim Task 5 mussten die Teilnehmenden ein valides Passwort eingeben und den Prozess bewerten, bis ein gültiges Passwort gefunden wurde.

Der Prozess wurde auf einer Skala von 1 bis 7 wie folgt bewertet:

1. unverständlich	7. Verständlich	<b>6.46</b>
1. kompliziert	7. Einfach	<b>6.46</b>
1. übersichtlich	7. Verwirrend	<b>2.46</b>

#### **Wo sehen Sie Verbesserungspotenzial?**

- Mehr Abstand bei Kriterien, führt zu mehr Übersicht
- Symbols ist nicht beliebt

### **Task 6: Bestätigen Passworteingabe im "Confirm-Password" Feld**

Die sechste und letzte Aufgabe bestand daraus, dass bei Task 5 eingegebene Passwort auch im "Confirm-Password" Feld zu erfassen.

#### **Wie gut war die Unterstützung der Applikation während der Eingabe?**

- Sehr gut
- Alles wurde erklärt
- verständliches Feedback
- Eingabe erleichtert

**Haben Sie versucht das Passwort per copy-paste einzufügen?**

- Ging leider nicht
- Enttäuscht, aber besser so

**Zusatzfragen:**

**Haben Sie den Darkmode bereits ausprobiert?**

- Ja, sehr gut, Platzhalter aber erkenntlicher machen

**Welches der beiden Exemplare gefällt Ihnen besser?**

Beim linken Exemplar handelte es sich um die Variante mit Inputfelder mit einem vollen Rand. Das Rechte Exemplar hat Inputfelder, welche nur eine "Underline» haben.

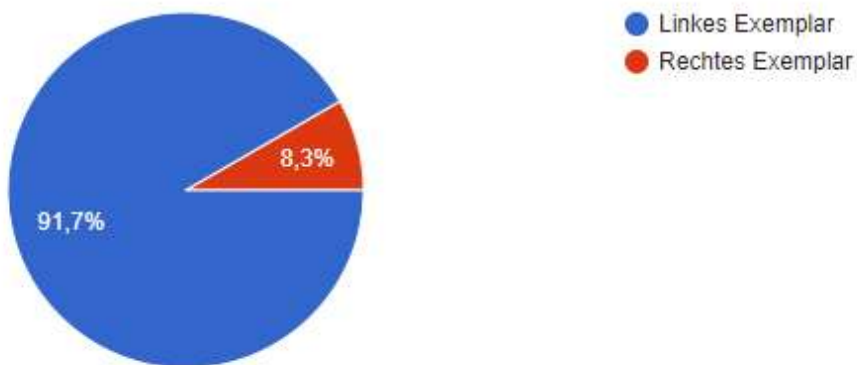


Abbildung 36: Usability Test Vergleich Varianten Registrierung

### 2.4.1.3 Login

Wie der Teil der Registrierung Komponente besteht auch der Login Teil aus verschiedenen Tasks und einzelnen Zusatzfragen, welche der Teilnehmer beantworten muss.

#### Task 1: Normaler Loginvorgang

Für den Task 1 mussten die Teilnehmenden selbstständig einen Login Vorgang vornehmen.

1. unverständlich	7. Verständlich	6.54
1. unangenehm	7. Angenehm	6.46
1. herkömmlich	7. modern	5.85
1. erwartungskonform	7. nicht erwartungskonform	2.46
1. übersichtlich	7. Verwirrend	1.92
1. aufgeräumt	7. Überladen	1.77
1. ineffektiv	7. effektiv	6.38

#### Gab es während dem Vorgang irgendwelche Schwierigkeiten?

Nein, alles Einwandfrei

#### Was hat Ihnen besonders gut gefallen?

- Nichts
- Interface
- gross und übersichtlich
- Rückmeldung
- Design
- Layout

#### Wo sehen Sie Verbesserungspotenzial?

- Show Button von Anfang an aktiv,
- Grösse der Schrift

#### Task 2: Syntaktisch falsche E-Mail-Adresse einfügen

Der Task 2 bestand daraus, eine syntaktisch falsche E-Mail-Adresse einzugeben.

#### Hat das Feedback der Applikation geholfen den Fehler zu verstehen und zu beheben?

Ja, zeigte klar wo der Fehler war

#### Wo sehen Sie Verbesserungspotenzial?

- Genaueres Feedback

#### Task 3: Korrekte E-Mail-Adresse, nicht `example@gmail.com`

Die Aufgaben beim Task 3 bestand darin, eine valide E-Mail-Adresse einzugeben, welche nicht [example@mail.com](mailto:example@mail.com) ist

#### Wurde Ihre Adresse erkannt? Wenn nicht bitte schreiben Sie Ihre Adresse auf.

Ja

#### Wo sehen Sie Verbesserungspotenzial?

Keines



#### Task 4: Beliebiges Passwort

Beim Task 4 mussten die Teilnehmenden ein beliebiges Passwort eingeben und beobachten, was passiert.

#### Haben Sie irgendwelche Bemerkungen?

- Show-Button von Anfang an freigeben lassen

#### Task 5: Login Knopf drücken

Die Teilnehmenden mussten beim Task 5 den Login Knopf drücken. Die E-Mail-Adresse sowie das Passwort wurden bei den vorhergehenden Tasks erfasst.

#### Hat das Feedback der Applikation geholfen den Fehler zu verstehen und zu beheben?

- Ja, es war klar was falsch ist
- Eventuell genauer anzeigen was falsch ist
- Es war unmissverständliches Feedback

#### Wo sehen Sie Verbesserungspotenzial?

- Kürzerer Text
- genauere Fehleranzeige

#### Zusatzfragen

#### Haben Sie den Darkmode bereits ausprobiert?

- Ja
- Nein
- Ja, funktioniert optimal
- nicht gerne aufgrund von linsen
- schön

#### Welches der beiden Exemplare gefällt Ihnen besser?

Beim linken Exemplar handelte es sich um die Variante mit Inputfelder mit einem vollen Rand. Das rechte Exemplar hat Inputfelder, welche nur eine "Underline" haben.

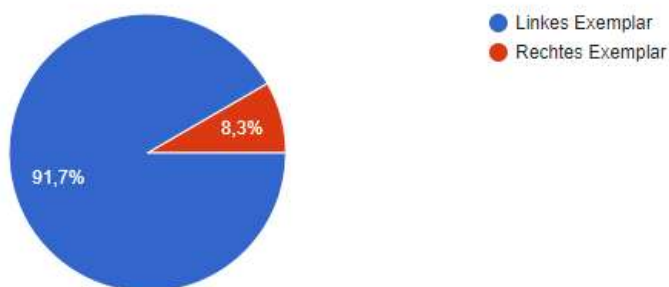


Abbildung 37: Usability Test Vergleich Varianten Login

## Allgemeine Abschlussfrage an Entwickler

### Falls Sie Entwickler sind, würden Sie dieses UI-Toolkit gebrauchen wollen?

- Ich bin kein Entwickler
- Ja, sieht angenehm aus und hat alle notwendigen Funktionalitäten
- Ja auf jeden Fall. Weil ich diesen erstens äusserst verständlich finde und alle Feedbacks optimal funktionieren. Zweitens auch, weil die Darstellung des UI Toolkit meiner Meinung nach sehr nutzerfreundlich ist.
- Kleine Änderungen würde ich vornehmen. Zum Beispiel so einen Roboter Check oder ähnliches. Sonst ist alles einfach erklärt und kurzgehalten! Sehr gut.
- Ja, würde beim Entwickeln sehr viel Zeit sparen. Gute Idee!
- Ja, da es schlicht, modern und pragmatisch ist. Der bereits vorhandene Darkmode ist ein weiteres Plus

#### *2.4.1.4 Fazit:*

Aufgrund des erhaltenen Feedbacks wurden die Komponenten angepasst. Bei beiden Komponenten ist bemängelt worden, dass die Abstände respektive die Weissräume zu klein sind. Die Abstände wurden daraufhin bei beiden Komponenten angepasst. Die Passwort Kriterien wurden aus Platzgründen bei der Registrierung Komponente nun vertikal angeordnet, da dies die Komponente übersichtlicher und angenehmer aussehen lässt. Ebenfalls wurde erwähnt, dass der Show-Password-Button zu Beginn inaktiv ist. Der Show-Password-Button wurde erst aktiv, wenn der Nutzer einen fehlgeschlagenen Login- oder Registrierungsversuch hinter sich hatte. Zusätzlich fanden die Teilnehmenden den Hinweis beim Registrierung Formular, , dass Sie ein möglichst starkes Passwort verwenden sollen, überflüssig. Diesen ist danach ebenfalls entfernt worden.

Das Feedback wurde miteinbezogen und die Komponenten wurden überarbeitet. Die finalen Komponenten werden im nächsten Abschnitt erklärt.

## 2.4.2 Login Screen

In der Abbildung 38 ist der initiale Zustand der Login Komponente ersichtlich.

Die Login Komponente besteht aus den folgenden Elementen:

- Titel
- E-Mail-Input
- Password-Input
- Show-Password-Button
- Submit-Button

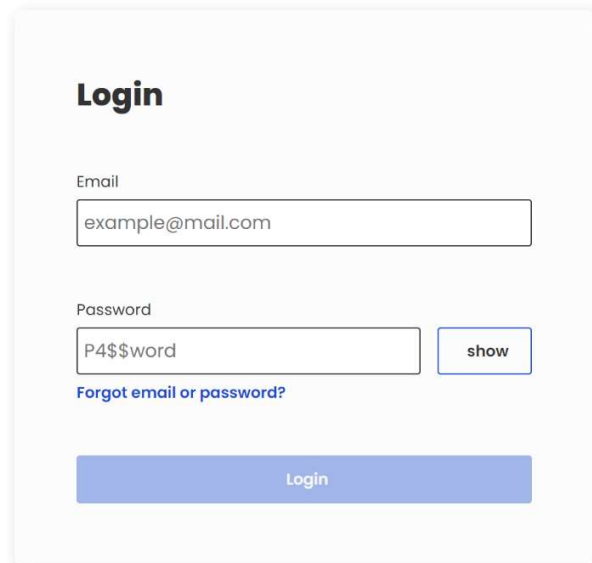


Abbildung 38: Login Komponente

### 2.4.2.1 Titel

Ein gewöhnlicher Titel, welcher den aktuellen Kontext anzeigt, in welchem man sich gerade befindet.

### 2.4.2.2 E-Mail-Input

Ein E-Mail-Input Feld mit einem Label. Gestaltungstechnisch wurde das Label bewusst in einem leichteren Schriftgewicht gewählt, damit der Nutzer nicht irritiert wird. Der Platzhalter wurde in einem helleren Grauton als der Rand des Inputs-Feld gewählt. Dem Nutzer soll damit verständlich gemacht werden, dass es sich dabei um einen Platzhalter und keinen Input handelt. Um eine valide Eingabe zu erzielen kann sich der Nutzer am Platzhalter orientieren. Der Platzhalter ist syntaktisch korrekt.

Die Eingabe des Nutzers wird validiert, sobald er den Fokus im E-Mail-Input Feld verlässt. Die Komponente validiert den Input und ändert entsprechend seinen Zustand.

#### Zustände

Das E-Mail-Inputfeld hat die folgenden Zustände:

##### Initial

Default Zustand, wenn noch keine Eingabe vorgenommen wurde und das Input Feld unberührt ist.



Abbildung 39: Login E-Mail-Input initialer Zustand

##### Aktiv

Der Zustand, wenn der User das Input Feld anvisiert und einen Input am Eintippen ist. Solange sich der Fokus im Inputfeld befindet, wird der Input nicht validiert und somit bleibt das Inputfeld aktiv.

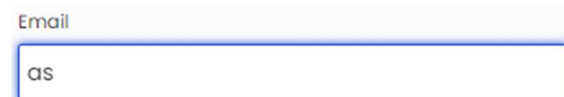


Abbildung 40: Login E-Mail-Input aktiv Zustand

### Error

Der Zustand, wenn der Fokus nicht mehr auf dem Input Feld ist und ein Input eingegeben wurde, welcher nicht valide ist.



Abbildung 41: Login E-Mail-Input Error Zustand

### Success

Der Zustand, wenn der Fokus nicht mehr auf dem Input Feld ist und ein Input eingegeben wurde, welcher valide ist.



Abbildung 42: Login E-Mail-Input Success Zustand

### Hover

Der Zustand, wenn der User über das Feld fährt aber den Fokus nicht auf das Inputfeld legt.

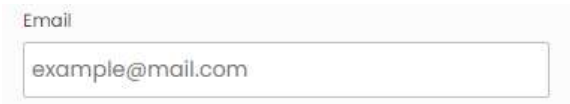


Abbildung 43: Login E-Mail-Input Hover Zustand

### 2.4.2.3 Password-Input

Ein Password-Input Feld mit einem Label sowie einem Link, falls der Nutzer seine E-Mail oder sein Password vergessen hat.

Initial wird das Passwort bei einem Password-Input Feld durch Punkte verschlüsselt, so dass das Passwort nicht ersichtlich ist. Um die Usability zu verbessern, wurde ein Show-Button implementiert, welche das Passwort ersichtlich macht. Diese Komponente wird im nächsten Teil erklärt.

#### Zustände

Das Password-Input Feld hat die folgenden Zustände:

#### Initial

Default-Zustand, wenn noch keine Eingabe vorgenommen wurde und das Input Feld unberührt ist.



Abbildung 44: Login Password-Input initialer Zustand

#### Hover

Der Zustand, wenn der User über das Feld schwebt, aber den Fokus nicht auf das Inputfeld legt.



Abbildung 45: Login Password-Input Hover Zustand

#### Aktiv

Der Zustand, wenn der User das Input Feld im Fokus ist. Solange sich der Nutzer mit dem Fokus im Input Feld befindet, bleibt das Input Feld im aktiven Zustand.

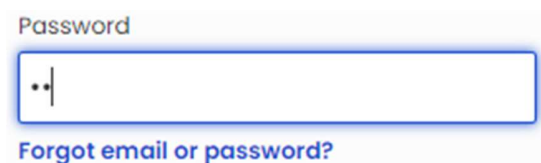


Abbildung 46: Login Password-Input aktiv Zustand

### Typed

Der Zustand, wenn der User bereits einen Input eingegeben hat, jedoch den Fokus wieder auf etwas anderes im Interface als das Password-Inputfeld legt. Zu bemerken ist, dass dieser Zustand nur auf diejenigen Inputfeldern vorkommt, auf die keine Validierung durchgeführt wird, da dieser sonst vom Success oder Error Zustand überschrieben wird.



Abbildung 47: Login Password-Input Typed Zustand

#### 2.4.2.4 Show-Password-Button

Wie im vorherigen Teil erwähnt, wurde ein Show-Password-Button erstellt, so dass der Nutzer seinen Password-Input sichtbar machen kann. Ursprünglich war der Show-Button als initialer Zustand inaktiv. Der Nutzer musste zuerst einen Login Versuch fehlschlagen, so dass der Button aktiv wurde. Jedoch wurde dies aufgrund des Feedbacks vom Usability Tests und dem Kunden geändert.

#### Zustände

Der Show-Password-Button hat die folgenden Zustände:

#### Hide

Der Zustand, in welchem der Password-Input nicht angezeigt wird. Der Text des Buttons zeigt in diesem Zustand "show" an, da ein Button immer eine Aktion anzeigt. Mit einem Klick auf den Button wechselt dieser in den nächsten Zustand.



Abbildung 48: Login Show-Password-Button Hide Zustand

#### Show

Der Zustand, in welchem der Password-Input angezeigt wird. Der Text des Buttons zeigt in diesem Zustand "hide" an.



Abbildung 49: Login Show-Password-Button Show Zustand

#### Pressed

Der Zustand, wenn der Button gedrückt wurde, aber der Klick noch nicht losgelassen wurde.

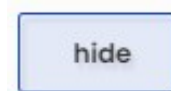


Abbildung 50: Login Show-Password-Button Pressed Zustand

#### 2.4.2.5 Submit-Button

Ein Submit-Button, mit welchem der Login-Request abgesendet werden kann. Der Button ist initial inaktiv. Er kann nur durch eine valide E-Mail Eingabe sowie eine Eingabe beim Passwort in den aktiven Zustand wechseln.

##### Zustände

###### Disabled

Der initiale Zustand. Der Submit-Button wechselt erst in den Enabled Zustand, wenn eine valide E-Mail-Adresse sowie ein Passwort eingegeben wurde.



Abbildung 51: Login Submit-Button Disabled Zustand

###### Enabled

Der Zustand, welcher nur mit einer validen E-Maileingabe sowie einer Passwordeingabe erreicht werden kann. Login-Request kann danach abgesendet werden durch Klicken des Buttons.



Abbildung 52: Login Submit-Button Enabled Zustand

###### Pressed

Der Zustand, wenn der Button gedrückt ist und noch nicht losgelassen wurde.



Abbildung 53: Login Submit-Button Pressed Button

## 2.4.3 Register Screen

In der Abbildung 54 ist der initiale Zustand unseres Registrierungsscreens abgebildet.

Die Komponente besteht aus den folgenden Teilen:

- Titel
- E-Mail-Input
- Password-Input
- Show Button
- Passwortstärke Bars
- Passwort-Kriterien
- Confirm-Password Input
- Register Button

### 2.4.3.1 Titel

Gewöhnlicher Titel in fetter Schrift. Zeigt den aktuellen Kontext an, in welchem man sich befindet.

### 2.4.3.2 E-Mail-Input

Ein E-Mail-Inputfeld mit einem Label. Gestaltungstechnisch wurde das Label bewusst in einem leichteren Schriftgewicht gewählt, damit der Nutzer nicht irritiert wird. Der Platzhalter wurde in einem helleren Grauton als der Rand des Input-Felds gewählt. Dem Nutzer soll damit verständlich gemacht werden, dass es sich dabei um einen Platzhalter und keinen Input handelt. Um eine valide Eingabe zu erzielen kann sich der Nutzer am Platzhalter orientieren. Der Platzhalter ist syntaktisch korrekt.

Die Eingabe des Nutzers wird validiert, sobald er den Fokus nicht mehr im E-Mail-Inputfeld hat. Unser Toolkit validiert den Input und ändert entsprechend seinen Zustand.

### Zustände

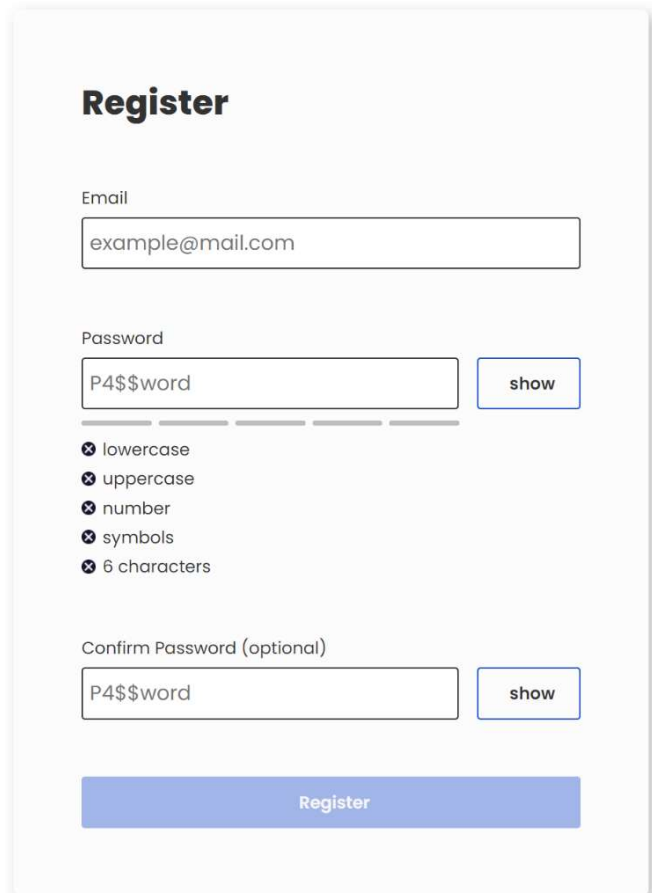
Der E-Mail-Input kann dieselben Zustände annehmen wie bereits im Abschnitt 2.4.2.2 dargelegt.

### 2.4.3.3 Password Input

Input Feld mit Typ Password. Der Typ stellt sicher, dass das Passwort beim Eingeben nicht sichtbar ist. Zusätzlich wird bei der Register Komponente überprüft, ob die Eingabe die Kriterien für ein sicheres Passwort erfüllen, bevor sich der Nutzer registrieren kann.

### Zustände

Der Password-Input kann dieselben Zustände annehmen wie bereits im Abschnitt 2.4.2.3 dargelegt.



The screenshot shows a registration form titled "Register". It contains the following elements:

- Email:** A text input field with the placeholder "example@mail.com".
- Password:** A text input field with the placeholder "P4\$\$word" and a "show" button to its right.
- Confirm Password (optional):** A text input field with the placeholder "P4\$\$word" and a "show" button to its right.
- Validation Criteria:** A list of requirements, each with a checked checkbox:
  - lowercase
  - uppercase
  - number
  - symbols
  - 6 characters
- Register Button:** A large blue button at the bottom of the form.

Abbildung 54: Register Komponente

#### 2.4.3.4 Show-Password-Button

Wie im vorherigen Teil erwähnt, wurde ein Show-Password-Button erstellt, sodass der Nutzer seinen Password-Input sichtbar machen kann. Ursprünglich war der Show-Button als initialer Zustand inaktiv. Der Nutzer musste zuerst ein Passwort eingeben, sodass der Button aktiv wurde. Dies wurde aufgrund der Auswertungen aus dem Usability Test und dem Feedback des Kunden geändert.

#### Zustände

Der Zustände des Show-Password-Button wurden bereits im Abschnitt 2.4.2.4 dargelegt.

#### 2.4.3.5 Passwortstärke Bars

Die Passwortstärke **Bars** geben die aktuelle Stärke des Passworts an. Der Input im Password-feld wird validiert und die **Balken** werden eingefärbt, wenn ein Kriterium mehr erfüllt wurde. Wenn nur ein Kriterium erfüllt wurde, ist nur ein Balken rot eingefärbt. Bei zwei bis vier Kriterien sind je nach Anzahl zwei bis vier Balken orange eingefärbt. Erst so bald alle Kriterien erfüllt wurden, sind alle Balken im Zustand Success und somit grün.

#### Zustände

Die Passwortstärke Bars haben die folgenden Zustände:

##### **Initial**

Der initiale Zustand



Abbildung 55: Register Passwortstärke Bars initialer Zustand

##### **Error**

Der Zustand, wenn nur ein Kriterium erfüllt wurde.



Abbildung 56: Register Passwortstärke Bars Error Zustand

##### **Warning**

Der Zustand, wenn nur gewisse Kriterien erfüllt sind und nicht alle.



Abbildung 57: Register Passwortstärke Bars Warning Zustand

##### **Success**

Der Zustand, wenn alle Kriterien erfüllt wurde.



Abbildung 58: Register Passwortstärke Bars Success Zustand

#### 2.4.3.6 Passwortstärke Kriterien

Kriterien, welche alle erfüllt werden müssen, damit das Passwort als valide angesehen wird. Nach jeder Eingabe wird der Input validiert und die Kriterien färben sich grün, wenn das Kriterium erfüllt ist oder rot, falls es noch nicht erfüllt wurde.

#### Zustände

Die Passwortstärke Kriterien haben die folgenden Zustände:

##### **Initial**

Der initiale Zustand, wenn noch kein Passwort Input eingegeben wurde.



Abbildung 59: Register Passwortstärke Kriterien initialer Zustand

##### **Success**

Der Zustand, wenn das Kriterium erfüllt wurde. Ersichtlich an der grünen Farbe.



Abbildung 60: Register Passwortstärke Kriterien Success Zustand



## Error

Der Zustand, wenn das Kriterium noch nicht erfüllt wurde.



Abbildung 61: Register  
Passwortstärke Kriterien Error  
Zustand

### 2.4.3.7 Confirm-Password-Input

Optionales Password-Inputfeld. Wenn der User sicher gehen will, dass er das Passwort korrekt eingegeben hat, kann er dies durch das Confirm-Password Feld bestätigen lassen. Wenn der Password-Input nicht mit dem Confirm-Password-Input übereinstimmt, wird angezeigt, dass diese nicht übereinstimmen. Beim Eingeben des Passworts wird dem User nach jedem Input angezeigt, ob er sich auf einem guten Weg befindet oder nicht.

#### Zustände

Der Confirm-Password-Input kann neben der bereits im Abschnitt 2.4.2.3 erklärten Zustände noch weitere annehmen:

#### No-Match

Der Zustand, wenn der Input im Confirm-Password-Feld nicht mit dem Input aus dem Password-Feld übereinstimmt.

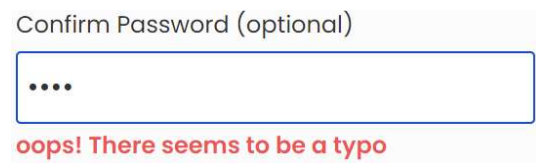


Abbildung 62: Register Confirm-Password-Input No Match Zustand

#### On-Your-Way

Der Zustand, wenn der Input im Confirm-Password-Feld teilweise mit dem Password-Input übereinstimmt. Sobald der Input an einer Stelle nicht mehr mit dem Passwort übereinstimmt, wechselt das Confirm-Password-Feld in den No-Match Zustand.

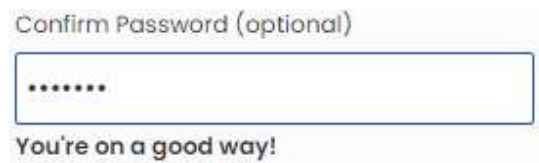


Abbildung 63: Register Confirm-Password-Input On-Your-Way Zustand

#### Match

Der Zustand, wenn der Input des Confirm-Password-Felds mit dem Input des Password-Feld übereinstimmt.



Abbildung 64: Register Confirm-Password-Input Match Zustand

### 2.4.3.8 Register Button

Ein Submit-Button, mit welchem der Register-Request abgesendet werden kann. Der Button ist per default inaktiv. Er kann nur durch eine valide E-Maileingabe sowie eine Angabe beim Passwort in den aktiven Zustand wechseln.

#### Zustände

Der Register Button kann dieselben Zustände, wie bereits im Abschnitt 2.4.2.5 dargelegt wurde, annehmen.

## 2.5 Code

### 2.5.1 CSS

Um unsere Design-Guidelines auch im Code umzusetzen, wurde das CSS aufgeteilt. Jedes der beiden Komponenten besitzt ein eigenes CSS-File, in welchem die komponentenspezifischen Styles deklariert sind. Allgemeingültige CSS-Regeln wie das Farbschema, Typografie und Buttons wurden in CSS-Dateien deklariert, welche unabhängig der Login- oder Registrierungskomponente sind.

Die allgemeinen CSS-Regeln wurden in den folgenden drei Dateien festgelegt:

#### **globals.css**

In `globals.css` werden allgemeine CSS-Regeln aufgestellt wie zum Beispiel das Aussehen der Inputfelder, Primär- und Sekundärbutton, Titel und Label. Diese gelten für alle Komponenten, die das Toolkit zurzeit anbietet.

#### **kolibriColorScheme.css**

Im `kolibriColorScheme.css` wurden als Root-Variablen alle Farben für den Lightmode deklariert, sowie als *data-theme* für den Darkmode. Dies ist eine der einfachsten Varianten, um Dark- und Lightmode in CSS umzusetzen.

Im praktischen Gebrauch kann danach ein Button oder ein Slider das *data-theme* Attribut auf "light" oder "dark" anpassen.

#### **fontStyles.css**

In der Datei `fontStyles.css` wurden die Schriftgrößen und Schriftgewicht als Root-Variablen deklariert.

### 2.5.2 Anpassbarkeit

Eine Anforderung an das Toolkit ist es, dass Entwickler das Design eigenständig anpassen können. Um sicherzustellen, dass dies möglich ist, wurde der Code auf dies geprüft. Der Entwickler kann einfach eigene CSS-Dateien erstellen und unsere Regeln überschreiben. Unsere Designvorgaben werden vom Browser ignoriert und die Komponenten sind individuell anpassbar.

Dies gibt Entwicklern enorm viel Flexibilität, da jedes Projekt andere Anforderungen mit sich bringt und andere Gestaltungsentscheidungen getroffen werden müssen.

# 3. Verpackung, Verbreitung, Strukturierung und Implementierung

Damit das Toolkit einfach zu implementieren und frei von Unabhängigkeiten ist, braucht es eine dementsprechende Strukturierung des Codes. In diesem Abschnitt werden verschiedene Strukturierungsmöglichkeiten beschrieben und auf deren Vor- und Nachteile eingegangen. Ein weiterer wichtiger Aspekt der Strukturierung ist auch die spätere Verbreitung des Codes, damit Webentwickler einen minimalen Aufwand haben, die Komponenten des Toolkits in ihr System zu integrieren. Auch die Verbreitungsart und Auslieferung der Codebasis soll möglichst unkompliziert und einfach sein.

## 3.1 Verpackungsmöglichkeiten

In diesem Abschnitt werden die verschiedenen Varianten verglichen, wie das Open-Source UI-Toolkit verpackt werden kann. Dabei wurden die zwei bekanntesten Varianten, Framework und Library, auf ihre Vor- und Nachteile überprüft.

### 3.1.1 Framework

Ein Framework besteht im Gegensatz zu einer Library nicht nur aus einzelnen Funktionen, sondern bietet, wie der Name schon sagt, eine Rahmenstruktur vor, an welche man sich halten muss. Dies verpflichtet den Entwickler dazu, gewisse Konventionen zu beherrschen und einzuhalten. Ebenfalls wird vorgegeben, wie der Code geschrieben werden muss, damit das Framework versteht, was zu tun ist. Dabei ergeben sich die folgenden Vor- und Nachteile für die Verwendung eines Frameworks.

#### *3.1.1.1 Vorteile von Frameworks*

Frameworks haben eine klare Versionierung, was die Durchführung von Updates des Toolkits vereinfacht. Durch die vorgegebene Art des Programmierens entsteht auch mehr Sicherheit und Stabilität für die Komponenten, da es kaum Anpassungsmöglichkeiten gibt.

#### *3.1.1.2 Nachteile von Frameworks*

Der Entwickler ist an die Regeln des Frameworks gebunden. Die Anwendung des Frameworks muss zuerst erlernt werden, bevor es effizient genutzt werden kann. Ein Framework ist schwer austauschbar, was die Modularität und Unabhängigkeit unseres Toolkits beeinträchtigen würde.

### 3.1.2 Library

Eine Library ist an sich eine Sammlung von wiederverwendbaren Funktionen, welche ein Entwickler verwenden und schnell in seinen Code einbauen kann. Dem Entwickler werden keine Vorgaben zur Anwendung einer Library gelegt.

#### 3.1.2.1 Vorteile von Libraries

Funktionen aus einer Library sind schnell austauschbar. Der Entwickler behält die Kontrolle seiner Applikation und muss keine Konventionen oder Regeln befolgen. Das Setup einer Library ist schnell und unkompliziert durch einen einfachen Import möglich.

#### 3.1.2.2 Nachteile von Libraries

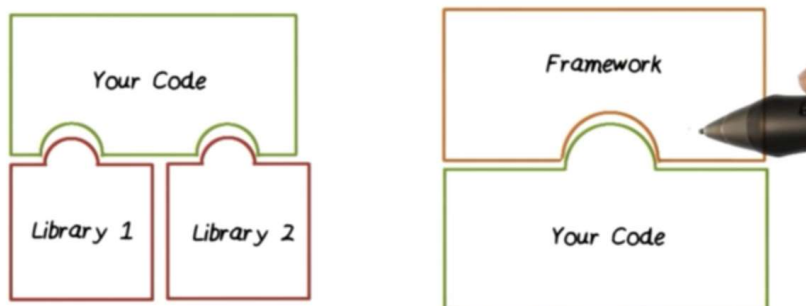
Die Funktionen einer Library können nicht vom Entwickler verändert oder erweitert werden. Da der Entwickler die Kontrolle der Applikation behält, können in bestimmten Fällen Nebenwirkungen beim Ausführen des Codes entstehen und somit zu einem falschen Ergebnis führen.

### 3.1.3 Fazit

Aufgrund der Vor- und Nachteile der geprüften Varianten wurde entschieden, dass UI-Toolkit als Library zu verpacken. Die Vorteile für den Entwickler sowie für zukünftige Kontributoren zum Projekt überwiegen dessen eines Frameworks.

Wie in der untenstehenden Abbildung 65 visualisiert, wird eine Library in den Code eines Entwicklers integriert, hingegen wird bei einem Framework der eigene Code in das Framework eingebunden. Damit kann ein Entwickler die Kontrolle über den Entwicklungsprozess behalten und trotzdem das Toolkit verwenden. [15]

#### Inversion of control



Libraries plug into your code, Your code plugs into a framework.

Abbildung 65: Kontrollschema zwischen Framework und Library

## 3.2 Verbreitungsmöglichkeiten

Damit Entwickler das Toolkit verwenden können, muss dieses in einer bestimmten Form ausgeliefert werden. Im Rahmen dieser Arbeit wurden NPM, CDN und Downloadables miteinander verglichen, da diese für die Auslieferung von Libraries geeignet sind. Nachdem diese auf ihre Vor- und Nachteile überprüft wurden, wird dieser Abschnitt mit einem Fazit abgeschlossen.

### 3.2.1 NPM

NPM, ehemals auch Node Package Manager genannt, ist ein Packet Manager für Node.js. Seit dem Jahr 2020 gehört NPM zu Github und ist somit indirekt Teil des Microsoftkonzerns.

Um abzuklären, ob NPM ein geeignetes Werkzeug für das Toolkit ist, wurde die Login Komponente in ein neues Repository übertragen und den Code so geändert, dass er auf NPM hochgeladen werden kann. Der Grund dafür lag daran, dass kein HTML oder CSS in ein NPM-Paket verpackt werden kann. Dementsprechend wurde der Code in eine einzige JavaScript-Datei verlagert und das HTML und CSS dem DOM dynamisch beigefügt.

Das NPM-Package kann unter dem folgenden Link aufgerufen werden:

<https://www.npmjs.com/package/kolibree>.

Sowie unser Repository:

<https://github.com/fabianhaef/Kolibree>

#### 3.2.1.1 Erstellung eines NPM Pakets

Der Prozess um ein Projekt als Paket auf NPM hochzuladen, ist sehr einfach und geradlinig. Um ein Paket auf NPM hochzuladen, muss Node auf dem Rechner installiert sein. Das Projekt, welches man hochladen möchte, soll mit dem `npm init` Befehl in der Kommandozeile als Node-Projekt initialisiert sein. Dies ist notwendig, um eine `package.json` Datei zu erstellen, die das Projekt mit einer Versionsnummer ausstattet und beschreibt.

Danach kann das Projekt mit dem Befehl `npm publish` auf NPM publiziert werden. Soll nach einer Änderung das Paket nochmals hochgeladen werden, muss in der `package.json` Datei die Versionsnummer aktualisiert werden. Das Paket sollte nun auf `npmjs.com/package/<projektname>` zu finden sein und für andere Entwickler mit dem Befehl `npm i <projektname>` herunterladbar sein.

Es ist zu empfehlen, das Projekt mit einer Readme Datei zu beschreiben und erklären. Ein seriöses Softwareprojekt verfügt in jedem Fall über eine Readme Datei. Dieses Readme wird dann auch automatisch auf der `npmjs.com` Seite verwendet. [16]

### 3.2.1.2 Vorteile von NPM

Das Erstellen von Paketen ist sehr einfach und verständlich. Durch die Versionsnummer, welche in der package.json Datei eingegeben wird, können neuere Versionen vom Toolkit einfach erstellt werden. Damit wird gewährleistet, dass ein Überblick über Erweiterungen und Veränderungen existiert.

### 3.2.1.3 Nachteile von NPM

Node muss auf dem Rechner des Entwicklers und der zukünftigen Kontributoren installiert sein, damit das Hoch- und Herunterladen des Toolkits möglich ist. Dies können Sicherheitsrisiken mit sich bringen, da durch den Befehl von `npm i <projektname>` diverse Skripts auf dem Rechner gestartet werden. Zusätzlich wird somit die Unabhängigkeit unseres Toolkits verletzt, da auf die Verfügbarkeit des NPM Servers gezählt werden muss.

## 3.2.2 CDN

Unter einem Content Delivery Network versteht man ein verteiltes Netzwerk von Servern, welches vor allem die Ladezeit von Webinhalten verkürzt. Indem die physische Distanz zwischen dem Nutzer und dem Ort des Inhalts reduziert wird, kann der Inhalt schnell zurückgesendet werden. Ohne ein CDN müsste der Ursprungsserver jede Anfrage bearbeiten und beantworten, was den Datenverkehr erhöht sowie eine hohe Belastung für den Server bedeutet. Die Dateien, die über einen CDN-Link zur Verfügung gestellt werden, können mit einem einfachen Link- oder Script-Element im HTML eingebunden werden. So können CSS-Dateien im Head-Element und JavaScript-Dateien am Ende des Body-Elements eingefügt werden. In der Abbildung 66 sieht man ein Beispiel dazu. [17]

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <meta name="robots" content="index, follow">
  <meta name="description" content="">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/gh/fabianhaef/
ip6-web-ui-toolkit@cdn-production/style.css">
  <title>Web-Ui-Toolkit</title>
</head>

<body>

  <div class="login"></div>

  <script type="module" src="./script.js"></script>
  <script type="module" src="https://cdn.jsdelivr.net/gh/fabianhaef/
ip6-web-ui-toolkit@cdn-production/script.js"></script>
</body>
```



Abbildung 66: Verwendung von CDN in der HTML-Datei

In dieser Arbeit wurde jsDeliver für den Versuch, ein eigenes CDN zu erstellen, verwendet. Mit jsDeliver können Dateien direkt aus dem Github Repository in ein gehostetes CDN von jsDeliver verwandelt werden. Dabei wurden zwei Ansätze verfolgt. [18]

#### 3.2.2.1 Erster Ansatz:

Die Komponenten werden in einer JavaScript-Datei gebündelt und somit wird nur ein Link für die JavaScript-Datei und ein Link für die CSS-Datei verwendet.

Die Codebasis für diesen Ansatz kann unter diesem Link abgerufen werden:

<https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/flafd6385ffe5c45e4f09b7beece7b72e1f88620>

##### 3.2.2.1.1 Vorteile vom ersten Ansatz

Man kann das Toolkit schnell in den eigenen Code einbinden, da man nur zwei Links verwenden muss. Somit ist die Integration des Toolkits unkompliziert und einfach.

##### 3.2.2.1.2 Nachteile vom ersten Ansatz:

Die JavaScript-Datei benötigt bei mehreren Komponenten schnell viel Speicherplatz. Dadurch dass diese grosse Datei geladen werden muss, beeinträchtigt dies die Performance der Applikation

#### 3.2.2.2 Zweiter Ansatz:

Aufteilung der Komponenten in einzelne JavaScript-Dateien (CSS und HTML wird dem DOM dynamisch hinzugefügt). Die Codebasis für diesen Ansatz kann unter diesem Link abgerufen werden:

<https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/cdn-production>

##### 3.2.2.2.1 Vorteile vom zweiten Ansatz:

Der Entwickler kann sich gezielt die Komponenten aussuchen, die er benötigt und diese in seine Applikation einbinden. Die Dateien sind kleiner und können somit effizienter geladen werden. Die Library kann übersichtlicher gestaltet werden, indem einzelne JavaScript-Dateien den Namen der Komponente beinhalten können.

##### 3.2.2.2.2 Nachteile vom zweiten Ansatz:

Falls ein Entwickler viele unserer Komponenten gleichzeitig verwenden möchte, so müssen auch viele Links eingebunden werden, was die Übersichtlichkeit des Codes beeinträchtigen könnte.

#### 3.2.2.3 Vorteile von CDN:

CDN kann ebenfalls versioniert werden, indem man mithilfe von Github Releases neue Versionen erstellt, die man anschliessend auf CDN hochladen kann. [19]

Die Versionsnummer ist ebenfalls im Link ersichtlich. Die Einbindung in eine bestehende Applikation ist einfach durchführbar und braucht im Gegensatz zu NPM keinerlei Vorinstallationen. Entwickler, die eine ältere Version verwenden, können diesen weiterhin verwenden, selbst bei Erneuerungen. Somit ist gewährleistet, dass keine Komponenten einer älteren Version nicht mehr unterstützt werden und somit weiterhin einwandfrei funktionieren.

#### *3.2.2.4 Nachteile von CDN:*

Der Entwickler hat kein Zugriff auf die Codebasis des Toolkits und kann somit keine Änderungen vornehmen. Es besteht eine Abhängigkeit auf den CDN-Server und die Annahme, dass dieser die Dateien jederzeit zur Verfügung stellt.

### 3.2.3 Downloadables

Um das Toolkit als Downloadables zur Verfügung zu stellen, wird lediglich das Repository publiziert und die Entwickler können anschliessend die benötigten Dateien direkt vom Repository herunterladen und es in ihren Projekten einfügen. Somit können die Entwickler direkt den Inhalt bearbeiten und so das Aussehen und die Funktionalität bei Bedarf anpassen.

#### *3.2.3.1 Vorteile für Downloadables:*

Der Entwickler hat die volle Kontrolle der Komponenten und kann selbstständig Anpassungen durchführen, wenn notwendig. Da die Dateien nicht über einen Server geladen werden müssen, steigt somit auch die Performance der Applikation. Es bestehen keine Abhängigkeiten von Drittanbietern. Auch ist die Applikation des Entwicklers unabhängig von zukünftigen Versionen des Toolkits. Der Zugang zu den Dateien und die Einbindung ist einfach.

#### *3.2.3.2 Nachteile für Downloadables:*

Mehr Freiheit für den Entwickler bedeutet auch mehr Verantwortung. Somit muss der Entwickler selbst die Wartung dieser Dateien übernehmen. Das Projekt des Entwicklers hat zusätzliche Dateien, die er aus dem Toolkit in seinem Projekt integrieren muss. Die Versionierung müsste selbstständig durchgeführt werden.

### 3.2.4 Fazit

Während der Analyse wurde auf ein Kompromiss zwischen CDN und Downloadables eingegangen. Der Grund dafür ist, dass sowohl die Einfachheit der CDN-Links als auch die Möglichkeit, die Komponenten selbst anpassen zu können, überzeugend sind. Somit wurde zu einem Entschluss gekommen, dass die Komponente per CDN zugänglich gemacht werden sollen und da das Toolkit ohnehin ein open-source Projekt ist, wird auch die Möglichkeit bestehen, die Dateien aus dem Repository herunterzuladen.



## 3.3 Strukturierung des Codes

In diesem Abschnitt werden unterschiedliche Möglichkeiten zur Strukturierung des Codes untersucht, die für das Toolkit am besten geeignet sind. Der Fokus lag darauf, dass zukünftige Kontributoren das Toolkit einfach ergänzen und erweitern können sowie auf die Modularität der Komponente, damit so wenig Codeduplikation wie möglich entsteht. Somit bleibt der Code übersichtlich und strukturiert. Dafür wurden bekannte Methoden angeschaut wie Web Components, normales HTML, CSS und JavaScript sowie ein Design-Pattern namens Projektor-Pattern.

### 3.3.1 Web Components

Eine Webkomponente erlaubt es uns benutzerdefinierte, wiederverwendbare und gekapselte HTML-Elemente (auch Tags genannt) in Webapplikationen zu verwenden. Es werden weder spezielle Frameworks noch Libraries benötigt, aber können zusammen verwendet werden. Diese Eigenschaften entsprechen den Anforderungen, weshalb dieser Ansatz weiter untersucht wurde.

#### 3.3.1.1 Merkmale einer Webkomponente

Eine Komponente besteht aus mehreren Bestandteilen, die am Ende eine Einheit bilden, welche einfach in einem HTML-Datei als eigener Tag verwendet werden kann. Diese Bestandteile bestehen aus Custom Elements, Shadow-DOM und HTML Templates. Ein weiteres Merkmal der Webkomponente ist die Implementierung in einem Objektorientierten Programmierstil. Dies fällt sofort auf, da Custom Elements eine Erweiterung von der Klasse HTML-Element ist und diese somit von dieser Klasse erben. Auf die einzelnen Bestandteile wird nun genauer eingegangen.

#### **Custom Elements**

Mit den Custom Elements kann man seine eigene HTML-Elemente erstellen. Diese Elemente haben ihre eigene Lifecycle Methoden, die nun detaillierter beschrieben werden.

#### **Custom Elements Lifecycle Methoden**

Lifecycle Methoden werden zu unterschiedlichen Zeitpunkten automatisch aufgerufen. Es stehen insgesamt vier Methoden für die Custom Elements zur Verfügung.

*constructor()*: Da die Custom Elements von der Klasse HTML-Element erweitert werden, braucht es einen Konstruktor. Dieser wird aufgerufen, sobald eine neue Instanz des Elements erstellt wird.

*connectedCallback()*: Wird aufgerufen, sobald das Element dem DOM hinzugefügt wird.

*disconnectedCallback()*: Wird aufgerufen, sobald das Element aus dem DOM entfernt wird.

*attributeChangedCallback(attributeName, oldValue, newValue)*: In Webkomponenten können bestimmte Attribute definiert werden. Dies sind Variablen, die an die Komponente gebunden sind. Es wird später in diesem Abschnitt etwas genauer darauf eingegangen. Diese Methode wird aufgerufen, sobald ein Attribut

hinzugefügt, verändert, entfernt oder ersetzt wurde. Dabei wird der Name des Attributs dem *attributeName* Parameter übergeben, sowohl den alten (*oldValue*) als auch den neuen Wert (*newValue*).

### **Shadow-DOM**

Dank dem Shadow-DOM können Styles, Funktionalitäten und die Struktur des Elements verkapselt werden. Dies verhindert, dass vom Entwickler bereits erstellte Styles, die zufälligerweise einen gleichen Selektor haben, das Styling des Custom Elements verändern.

### **HTML Templates**

In den HTML Templates, definiert man die Struktur und das Styling der Komponente, die wie erwähnt vom Shadow-DOM eingekapselt werden. Somit kann sowohl HTML und CSS verwendet werden. Zusätzlich hat man die Möglichkeit, die Verkapselung des Shadow-DOMs etwas aufzulockern, indem man mit den so genannten *Slots*, dem Entwickler erlaubt ein Element von seiner HTML-Datei aus hinzuzufügen. Diese Möglichkeit wird später genauer erklärt und demonstriert.

Als Endprodukt hat man eine ganze Komponente, die aus mehreren Teilkomponenten oder andere HTML-Elementen bestehen kann und verkapselt in einer externen HTML-Datei wiederverwendet werden kann. [20]

#### 3.3.1.2 Beispiel Counter

Anhand eines einfachen Beispiels wurde die Theorie in die Praxis umgesetzt, um die Einfachheit und Flexibilität der Webkomponente besser einschätzen zu können. Der Code dieses Beispiels kann unter diesem Link aufgerufen werden: <https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/webcomponent>

Dazu wurde eine kleine Applikation erstellt, die aus einem Titel, einem Zustand (Counted: 10) und einem Button besteht (siehe Abbildung 67).

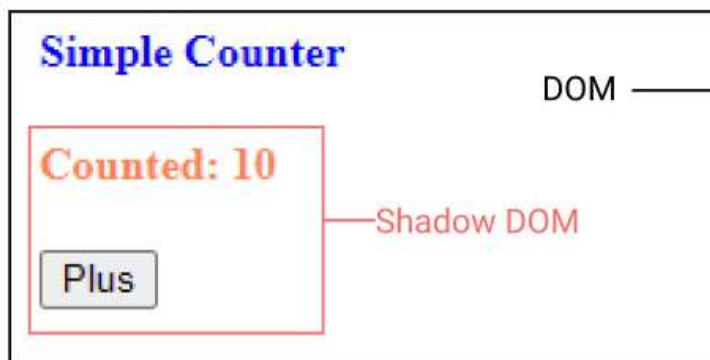


Abbildung 67: Interface der Beispielapplikation Counter implementiert mit Webkomponente

Der Titel "Simple Counter" sitzt ausserhalb des Shadow-DOMs, während der Counter und der Button Teil des Shadow-DOMs sind. Die Funktion des Buttons besteht darin, den Counter bei jedem Klick um eins zu erhöhen. Der Anfangszustand des Counters wurden mithilfe eines Attributs auf den Wert 10 gesetzt, um die Funktionalität von Attributen veranschaulichen zu können.

## Index.html

Die HTML-Datei besteht aus folgenden Elementen (siehe Abbildung 68).

```
<head>
  <link rel="stylesheet" href="style.css">
  <title>IP6 Web UI-Toolkit Web Components</title>
</head>

<body>
  <h4>Simple Counter</h4>
  <my-counter start-count="10">
    <div slot="buttonText">Plus</div>
  </my-counter>

  <script src="script.js"></script>
</body>
```

Custom Element

Abbildung 68: Index.html Datei der Webkomponente

Wie man sieht, kann man nun über das Attribut *start-count* den Anfangszustand des Counters setzen. Zusätzlich wird mithilfe des *Slots* der Text für den Button definiert. Noch deutlicher zu sehen ist dies, wenn man die Struktur im Entwicklertool innerhalb des Browsers begutachtet (siehe Abbildung 69).

```
<body>
  <h4>Simple Counter</h4>
  <my-counter start-count="10">
    #shadow-root (open)
      <style> h4 { color: coral; } </style>
      <div class="counter-label">
        <h4>Counted: 10</h4>
        <button>
          <slot name="buttonText">_</slot>
        </button>
      </div>
      <div slot="buttonText">Plus</div>
    </my-counter>
  <script src="script.js"></script>
  <!-- Code injected by live-server -->
  <script type="text/javascript">_</script>
</body>
```

Abbildung 69: Ansicht der Webkomponente im Entwicklertool innerhalb des Browsers

## Script.js

In der JavaScript-Datei kann man gut erkennen, wie die drei Bestandteile Custom Element, Shadow-DOM und HTML Templates miteinander kombiniert werden.

```
const template = document.createElement('template')
template.innerHTML = `
  <style>
    h4 {
      color: coral;
    }
  </style>
  <div class="counter-label">
    <h4></h4>
    <button><slot name="buttonText" /></button>
  </div>
`
```

Abbildung 70: HTML Template in script.js

In der Abbildung 70 erkennt man, dass das Template aus einem Template-Element besteht und darin wird sowohl das CSS als auch das HTML für die Komponente festgelegt. Mit dem Slot-Element (Markierung 1) erlaubt man es, dass man von aussen (im index.html) dieses Element durch ein vom Entwickler selbst gewähltes Element ersetzen kann. Im Falle dieses Beispiels wurde ein Div-Element eingesetzt mit dem inneren Text von "Plus" (ersichtlich in der vorherigen Abbildung 69). Dies führt dazu, dass innerhalb des im Shadow-DOM definierten Button-Element dieses Div-Element hinzugefügt wird und somit der Text "Plus" im Browser angezeigt wird. Dies gibt dem Toolkit die Möglichkeit, verschiedene Teile einer Komponente, von aussen bearbeitbar zu machen.

```

class Counter extends HTMLElement { 1
  constructor() {
    super()

    this.attachShadow({ mode: 'open' })
    this.shadowRoot.appendChild(template.content.cloneNode(true)) 3
  }

  this.count = this.getAttribute('start-count')
  ? parseInt(this.getAttribute('start-count')) 4
  : 0

  this.shadowRoot.querySelector('h4').innerText = `Counted: ${this.count}`
}

increment() {
  this.count += 1
  this.shadowRoot.querySelector('h4').innerText = `Counted: ${this.count}`
}

connectedCallback() {
  this.shadowRoot.querySelector('button')
    .addEventListener('click', () => this.increment()) 5
}

disconnectedCallback() {
  this.shadowRoot.querySelector('button')
    .removeEventListener() 6
}
}

window.customElements.define('my-counter', Counter) 7

```

Abbildung 71: Custom Element in script.js

Der Rest des Codes wurde nun in verschiedenen Segmenten unterbrochen, um die Struktur besser erklären zu können.

1: Wie bereits erwähnt, werden Webkomponente objekt-orientiert implementiert. Somit muss jedes Custom Element eine Erweiterung eines HTML-Elements sein.

2: Hier sieht man die erste Lifecycle Methode, den *constructor*. Hier werden verschiedene Attribute definiert und der Shadow-DOM initialisiert.

3: Der Shadow-DOM wird mit der Konfiguration "mode: open" für die Entwicklertools in den Browser ersichtlich gemacht. Anschliessend wird das zuvor erstellte Template beigefügt.

4: Mit der *getAttribute(attributeName)* Funktion kann man die Werte der in der Custom Element zuvor definierten Attributen, wie *start-count* (siehe index.html in Abbildung 68), abfragen.

5 & 6: In diesem Beispiel sieht man eine mögliche Anwendung der beiden Lifecycle Methoden *connectedCallback()* und *disconnectedCallback()*. Es wird ein *EventListener* erstellt, sobald die Komponente dem DOM hinzugefügt wird und entfernt diesen wieder, wenn die Komponente aus dem DOM gelöscht wird.

7: Nachdem das Custom Element erstellt wurde, wird diese mit der *define* Methode definiert.

Um die Verkapselung des Shadow-DOMs zu visualisieren, wurde eine CSS-Datei erstellt, die das H4-Element selektiert und das *color* Attribut mit dem Wert *blue* überschreibt (siehe Abbildung 72).

```
h4 {  
  color: blue;  
}
```

Abbildung 72: CSS-Überschreibung des h4 Elements

Man erkennt jedoch, dass der Text "Counted: 10" (ersichtlich in der Abbildung 67) nicht davon betroffen ist, da dieser im Shadow-DOM verkapselt wurde.

Da die Anwendung einer Webkomponente verständlicher wurde, werden nun die Vor- und Nachteile analysiert.

### 3.3.1.3 Vorteile von Webkomponenten

Die Integration in ein bestehendes Projekt kann sehr leicht vorgenommen werden und dank des Shadow-DOMs besteht kein wesentliches Risiko, dass die bestehende Applikation beschädigt wird. Jede Komponente stellt ein eigenes HTML-Element dar und ist verkapselt, was die Unabhängigkeit verstärkt und die Kompatibilität in Frameworks und Libraries gewährleistet.

### 3.3.1.4 Nachteile von Webkomponenten

Um eine neue Komponente zu erstellen, muss viel Vorwissen über Webkomponente bestehen. Der Shadow-DOM hat neben seinen Vorteilen auch einige Nachteile. Zum Beispiel kann das Styling nicht vom Entwickler angepasst werden, wenn dies gewünscht ist. Der Implementationsaufwand von Komponenten kann sehr gross sein, da wegen des Shadow-DOMs die Komponente verkapselt sind. Dies verhindert, dass Komponente, die sich nur sehr wenig unterscheiden, modular implementiert werden können und somit eine komplett neue Implementierung benötigen. Angenommen man hätte eine Komponente, die für eine Business Applikation verwendet werden könnte, aber auch für eine alltägliche Applikation. Sie unterscheiden sich lediglich an der Funktionalität und Struktur. Nichtsdestotrotz muss man zwei Custom Elemente von Grund auf erstellen, die sich nur minimal unterscheiden. Dies beeinträchtigt die Modularität, welche in unserem Toolkit angestrebt wird.

### 3.3.2 HTML, CSS, JS

Die einfachste Variante für die Umsetzung eines Toolkits wäre es, diese in klaren Dateien zu strukturieren. Somit wäre der Aufbau von statischen Elementen im DOM in einer HTML-Datei, das Styling in einer CSS-Datei und die dynamischen Elemente und die Logik der Applikation in einer JavaScript-Datei.

Die für den ersten Usability Test verwendeten Login und Register Komponenten, wurden mit klaren HTML, CSS und JavaScript-Dateien realisiert.

Die Repositories zu den jeweiligen Komponenten können unter den folgenden Links abgerufen werden:

Login Komponente: <https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/login>

Register Komponente: <https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/register>

Die Prototypen wurden zusätzlich für Testzwecke auf Netlify publiziert und können unter den folgenden Links geöffnet und ausprobiert werden:

Login Link: <https://login--web-ui-toolkit.netlify.app/>

Register Link: <https://register--web-ui-toolkit.netlify.app/>

Wie im Abschnitt 2.4.1 Analyse Usability Test bereits erwähnt, wurden zwei verschiedene Styling Varianten erstellt, um die Popularität der beiden zu testen.

Das Repository ist recht einfach aufgebaut und man erkennt schnell, welche Dateien zu welcher Variante gehört. Dabei fällt auf, dass durch die zwei erstellten Komponentenvarianten viel duplizierter Code entstanden ist und es schnell klar wurde, dass eine bessere Strukturierung des Codes verwendet werden muss. Mit dieser Struktur wäre die Wartung und Ergänzung der Komponenten sehr erschwert und kann schnell zu Fehlern führen. Auch die Bindings von den verschiedenen Elementen wie Input Feldern und deren Zuständen sind sehr unübersichtlich. Zusätzlich ist es schwer zu verstehen, wie die Kommunikation dahinter funktioniert. Ebenso erschwert dies die Arbeit von zusammenhängenden Komponenten und die Gewährleistung der Modularität zwischen ähnlichen Komponenten.

#### 3.3.2.1 Vorteile von HTML, CSS und JS

Die Struktur ist simpel und für jeden Entwickler bekannt, somit ist auch die Erstellung von neuen Komponenten leicht. Der Aufbau, das Styling und die Logik der Komponenten sind klar getrennt.

#### 3.3.2.2 Nachteile von HTML, CSS und JS

Die Verbreitung kann sehr schnell sehr mühsam werden, wenn mehrere Komponenten in ein System integriert werden soll. Bereits für eine kleine Komponente müsste man mindestens drei Dateien in seinem System einbauen, damit die Komponente funktioniert.

Bei mehreren ähnlichen Komponenten kann schwer ein Zusammenhang gezogen werden, was dazu führt, dass viel duplizierter Code entsteht und die Modularität eingeschränkt wird.

### 3.3.3 Projektor-Pattern

Das Projektor-Pattern ist ein Design-Pattern, welches an der FHNW im Modul WebClients behandelt wurde [21]. Die Hauptstärke dieses Patterns ist die Modularität zwischen Projektoren und Präsentationsmodellen. Dabei können auf ein Model verschiedene Projektoren angewendet werden, welche sowohl das Aussehen als auch die Funktionalität der Applikation beeinflussen. Allerdings bleibt das Binding stets statisch an einem Ort erhalten. Mit dem Zusammenspiel des Observer-Patterns, werden Abhängigkeiten zwischen den einzelnen UI-Elementen verkapselt und sind für alle Projektoren zugänglich. Somit wird gewährleistet, dass trotz verschiedenen Projektoren, die Applikation stabil und funktional bleibt.

#### 3.3.3.1 Vorteile von Projektor-Pattern

Komponenten, die viele Ähnlichkeiten haben und sich nur wenig im Aussehen und der Funktionalität unterscheiden, können dasselbe Model verwenden und besitzen lediglich ihren eigenen Projektor. Somit wird die Modularität und Austauschbarkeit der einzelnen Komponenten gewährleistet, ohne viel Code duplizieren oder abändern zu müssen. Die Abhängigkeiten zwischen den UI-Elementen in einer Komponente werden ebenfalls durch die Verwendung des Observer-Patterns optimal verwaltet, was zur höheren Stabilität des ganzen Toolkits beiträgt. Die Codebasis kann in einzelnen Bestandteilen wie Model, View, Controller und Projektor aufgeteilt werden, was die Übersichtlichkeit enorm verstärkt. Durch diese Strukturierung kann die Erstellung einer neuen Komponente einfach durchgeführt werden.

#### 3.3.3.2 Nachteile von Projektor-Pattern

Da das Pattern noch nicht weit verbreitet ist, müssen zukünftige Entwickler, die neue Komponenten erstellen möchten, sich in das Pattern einarbeiten und ein Verständnis dafür aufbauen.

### 3.3.4 Fazit

Während der Implementation von allen drei Varianten war es erkenntlich, dass die Vorteile vom Projektor-Pattern die Vorteile der anderen übertrumpfte. Da man durch die Robustheit der Abhängigkeiten innerhalb einer Komponente und der Einfachheit des Austauschs von verschiedenen Projektoren, die wichtigsten Aspekte des Toolkits erreichen kann, wurde das Projektor-Pattern ausgesucht. Dieses Pattern wird mithilfe der Implementation im nächsten Abschnitt vertiefter erklärt.



## 3.4 Projektor-Pattern: Aufbau und Implementation

### 3.4.1 Aufbau des Projektor-Patterns

Das Projektor-Pattern besteht aus den drei wesentlichen Bestandteilen Controller, Model und Projektoren, die man als View ansehen kann. Es beruht somit auf das MVC-Entwurfsmuster, die auch mit den Datenflüssen übereinstimmen.

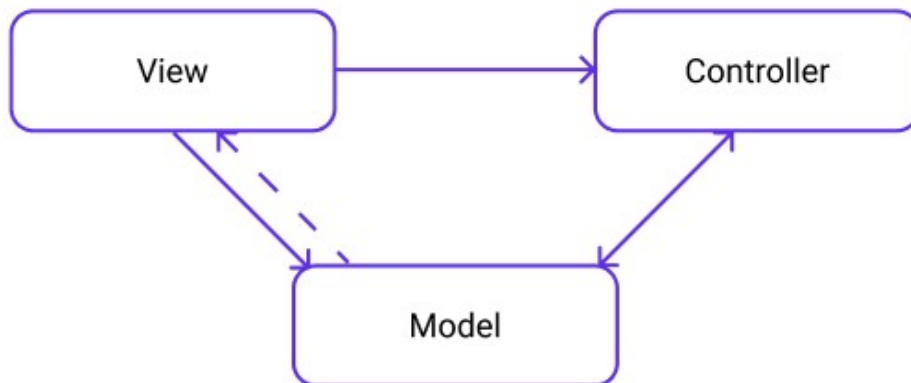


Abbildung 73: MVC Entwurfsmuster

Um ein besseres Verständnis für das Projektor-Pattern aufbauen zu können, wird nun die einzelnen Teile eingegangen und die Implementation anhand der Login Komponente angeschaut. Damit man es visuell besser versteht, kann man auf die Abbildung 73 zurückgegriffen werden.

Die Codebasis für diese Komponente kann unter diesem Link aufgerufen werden:

<https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/projector-pattern/login>

Das Bild zeigt eine UI-Komponente für das Login. Oben steht 'Login'. Darunter befindet sich ein Formular mit zwei Eingabefeldern: 'Email' (mit dem Wert 'examplemail.com') und 'Password' (mit dem Wert 'P4\$\$word'). Ein 'show'-Button ist rechts neben dem Passwortfeld. Unter dem Email-Feld steht 'Malformed Email'. Unter dem Password-Feld steht 'Forgot email or password?'. Ein 'Login'-Button befindet sich am unteren Rand.

Abbildung 74: UI der Login Komponente realisiert mit dem Projektor-Pattern

## Controller

```
/**
 * The Login Controller, which encapsulates the Model
 * @typedef {function(service): object} LoginController
 * @returns {{
 *   onLoginAdd: function(): number,
 *   addLogin: function(): void
 * }}
 */
const LoginController = service => {

  /**...
  const login = () => {...
  }

  const loginModel = ObservableList([])

  /**
   * Adds a new login to the login model
   * @returns {object} - The Login model and its externally available attribute functions
   */
  const addLogin = () => {
    const newLogin = login()
    loginModel.add(newLogin)
    return newLogin
  }

  return {
    onLoginAdd: loginModel.onAdd,
    addLogin: addLogin,
  }
}
```

Abbildung 75: login.js Datei des Projektor-Patterns

Im Controller gekapselt, befindet sich das Model, auf das danach eingegangen wird. Der Controller verwaltet das Model und fügt es einer *ObservableList* hinzu. Dies gewährleistet, dass auch das Hinzufügen einer neuen Login Komponente observiert werden kann. Dies wird verwendet, um die View darüber zu informieren, wann die Komponente gerendert werden soll. Dieser Code Abschnitt befindet sich in der Datei login.js.

## Model

Wir bleiben in der Datei login.js und schauen uns nun das gekapselte Model an.

```
/**
 * Holds all the Attributes of the Login component and makes them partially externally available
 * @typedef {object} Login
 * @property {Attribute} emailAttr - Emailadress
 * @property {Attribute} pwAttr - Password
 * @property {Attribute} formAttr - Indicates whether the form is valid or not
 * @property {Attribute} loginSuccessAttr - Indicates whether the Login attempt was successful or not
 * @property {Attribute} notificationAttr - Notification message when the user tries to Login
 * @returns {object} - Login Model
 */
const Login = () => {
  const emailAttr = Attribute('')
  const pwAttr = Attribute('')
  const formAttr = Attribute(false) // Checks whether all inputs of the Login form are valid
  const loginSuccessAttr = Attribute(false) // True if a login attempt is successful
  const notificationAttr = Attribute('') // Notification message for the user upon login attempt

  const updateFormValidity = () => {
    const result = emailAttr.getObs(VALUE).getValue() && pwAttr.getObs(VALUE).getValue()
    formAttr.getObs(VALID).setValue(result)
  }

  emailAttr.getObs(VALID).onChange( updateFormValidity )
  pwAttr.getObs(VALUE).onChange( updateFormValidity )

  emailAttr.setValidator( input => /.+@.+.+/.test(input) )

  return {
    getFormValidity: formAttr.getObs(VALID).getValue,
    onFormValidityChanged: formAttr.getObs(VALID).onChange,
    getEmail: emailAttr.getObs(VALUE).getValue,
    setEmail: emailAttr.getObs(VALUE).setValue,
    onEmailChanged: emailAttr.getObs(VALUE).onChange,
    getEmailValidity: emailAttr.getObs(VALID).getValue,
    onEmailValidityChanged: emailAttr.getObs(VALID).onChange,
    getPassword: pwAttr.getObs(VALUE).getValue,
    setPassword: pwAttr.getObs(VALUE).setValue,
    onPasswordChanged: pwAttr.getObs(VALUE).onChange,
    getPwVisibility: pwAttr.getObs(VISIBILITY).getValue,
    setPwVisibility: pwAttr.getObs(VISIBILITY).setValue,
    onPwVisibilityChanged: pwAttr.getObs(VISIBILITY).onChange,
    onLogin: service.loginAttempt,
    getLoginSuccess: loginSuccessAttr.getObs(VALID).getValue,
    setLoginSuccess: loginSuccessAttr.getObs(VALID).setValue,
    onLoginSuccessChanged: loginSuccessAttr.getObs(VALID).onChange,
    onNotificationChanged: notificationAttr.getObs(VALUE).onChange,
    getNotification: notificationAttr.getObs(VALUE).getValue,
    setNotification: notificationAttr.getObs(VALUE).setValue,
  }
}
```

Abbildung 76: Eingliederung des Modells

Im Model werden alle Daten und Attribute verwaltet, die von einer oder mehreren Komponenten verwendet werden können. Der Grund für die Verkapselung besteht darin, dass eine klare Trennung zwischen der View und dem Model entsteht und somit die View über den Controller Informationen des Modells erhält. In der Abbildung 76 haben wir das Model in verschiedene Segmente unterteilt.

Wie man es im ersten Segment erkennt, verwendet die Login Komponente die folgenden Attribute:

- EmailAttribute
- PwAttribute
- FormAttribute
- LoginSuccessAttribute
- notificationAttribute

Die Attribute sind mit dem Observer-Pattern verknüpft und haben somit Zugriff auf Funktionen wie *setValue(value)*, *getValue()* und *onChange()*. Um diese Methoden aufrufen zu können, muss man zuerst den Observer des Attributs mit *getObs(obsName)* holen. Beispiele dazu gibt es in den Segmenten zwei und drei.

Jedes einzelne Attribut kann eine beliebige Anzahl an Observer mit einer einheitlichen Benennung anlegen. Dieser verwaltet dann einen bestimmten Wert und soll andere Beobachter darüber informieren, sobald sich dieser Wert verändert. Somit können Input Felder wie die E-Mail und Passwort Input Felder den Wert vom Attribut *emailAttr* und *pwAttr* holen und diesen ändern, sobald der User eine Eingabe durchführt.

Andere Komponente oder Attribute wie das *formAttr* können anschliessend auf diese Veränderungen reagieren. Somit werden Verknüpfung nicht mehr über die UI Elemente durchgeführt, sondern über die Observer. Dies führt zu mehr Stabilität im Code und Flexibilität im User Interface, denn selbst wenn ein Element ersetzt werden sollte oder komplett entfernt wird, läuft die Applikation immer noch einwandfrei. Dieser Vorteil kann für die Modularität des Toolkits ausgenutzt werden.

Die Implementierung des Observable kann unter diesem Link angeschaut werden: <https://github.com/fabianhaef/ip6-web-ui-toolkit/blob/projector-pattern/observable/observable.js>

Im zweiten Segment sieht man, wie das oben genannte Beispiel in der Praxis umgesetzt werden kann. Die Funktion *updateFormValidity()* wird aufgerufen, sobald sich der Observer des E-Mail Attributs mit dem Namen "VALID" oder der Observer des Passwort Attributs mit dem Namen "VALUE" verändert. Das Aufrufen dieser Funktion führt anschliessend dazu, dass überprüft wird, ob die eingegebene E-Mail valide ist und ein Passwort eingetragen wurde. Das Resultat wird danach dem Observer des Form Attributs mit dem Namen "VALID" übergeben.

Später im Projektor wird man sehen, dass das *formAttr* verwendet wird, um den Status des Login Buttons zu setzen, da dieser zu Beginn deaktiviert ist und erst bei einer validen Eingabe von Passwort und E-Mail aktiviert werden soll.

Eine Anwendung des Attributs in dieser Form nennt man auch Derived Attribute, da dieser vom E-Mail und Passwort Attribut abgeleitet wird und somit Code Duplikationen verhindern kann.

Attribute können neben Observer auch Validatoren erstellen. Der *setValidator(callback)* Funktion wird eine Callback Funktion übergeben, die eine Validierung durchführt. Die Callback Funktion soll einen booleschen Wert zurückgeben, damit dieser dem Observer des Attributs übergeben werden kann. Um diesen Prozess besser zu verstehen, wird in der Abbildung 77 die Implementation des *setValidator(callback)* gezeigt, welches sich in der *presentationModel.js* Datei befindet.

```

/**
 * Sets the callback function to the VALID observable, which then tests the input everytime the VALUE Observable gets changed
 * @param {callback} validate - A callback function which returns true if the input is valid or false if its invalid
 * @returns {void}
 */
const setValidator = validate => getObs(VALUE).onChange( val => getObs(VALID).setValue(validate(val)));

```

Abbildung 77: SetValidator Funktion

Es ist ersichtlich, dass sobald der Wert des *VALUE* Observers des Attributs ändert, die Callback Funktion aufgerufen wird und das Ergebnis davon dem *VALID* Observer übergeben wird. Somit wird bei jeder neuen Eingabe des Users überprüft, ob die eingegebene E-Mail syntaktisch valide ist.

Im vierten und letzten Segment sieht man, welche Operationen nach aussen freigegeben werden sollen. Somit kann genau gesteuert werden, welche Werte von aussen geändert, gelesen oder observiert werden dürfen und welche nicht.

## Projektor

Beim Projektor-Pattern spielen die Projektoren eine wesentliche Rolle. Die Projektoren bestimmen, wie das User Interface aufgebaut ist und welche Funktionalitäten sie beinhalten soll. Eine Komponente kann aus mehreren verschachtelten Projektoren bestehen. So kann ein einfaches Element wie das Input Feld ein Projektor sein und ein Form-Element aus verschiedenen Input Feld Projektoren bestehen und gleichzeitig ein eigener Projektor sein. Diese Eigenschaft ist sehr wirkungsvoll, um die Modularität unserer Komponenten zu gewährleisten.

Einfach zu erkennen ist das, wenn man nun eine zweite Variante des Input Feld Projektors erstellt, die wesentlich mehr Funktionalität aufweist, wie zum Beispiel mit einer Unterstützung der Eingabe durch verschiedene Vorschläge, Validierung des Inputs oder die Dirty State überprüfen kann. Ersetzt man nun im Form Projektor die alten Input Feld Projektoren mit den neuen hat man ein völlig neues User Interface. Dies erlaubt es, Projektoren wieder zu verwenden und für bestimmte Domänen, wie Business-Controls, zu spezifizieren. Es wurde dazu ein Proof of Concept erstellt, welches in einem späteren Abschnitt in dieser Arbeit vorkommen wird.

Bei der Login Komponente wurden viele kleine Projektoren erstellt, die "Subprojektoren" genannt wurden. Diese Subprojektoren werden am Ende beim Hauptprojektor zusammengestellt und an richtigem Ort und Stelle dem DOM beigefügt. Der Grund für das Erstellen von Subprojektoren war die Übersicht der einzelnen verwendeten Projektoren beibehalten zu können. Auch Anpassungen und Erweiterungen an den Komponenten wird damit vereinfacht, da man schnell erkennt, wo die dazugehörigen Funktionalitäten der einzelnen Projektoren sind.

## Hauptprojektor

Beginnend mit dem Hauptprojektor wird nun die grobe Aufteilung der einzelnen UI-Elemente in den Subprojektoren aufgezeigt. Dafür wurde in der Abbildung 78 die Implementation erneut in einzelne Segmente aufgeteilt.

```
/**
 * The main Projector which uses all sub projectors in this module and ties them together into a single UI
 * @param {LoginController} loginController
 * @param {HTMLElement} rootElement - The root element which will be populated with the entire login component
 * @param {object} login - Holds all attributes of the login model
 */
const loginProjector = (loginController, rootElement, login) => {

  // -----Login Button----- 1
  const loginButtonElement = loginSubmitButtonProjector(login)

  // -----Login Title----- 2
  const titleElement = loginTitleProjector()

  // -----Input Elements----- 3
  const [ emailInputElement, emailLabelElement ] = loginEmailProjector(login, 'Email')
  const [ passwordInputElement, passwordLabelElement ] = loginPasswordProjector(login, 'Password')

  // -----Notification Elements----- 4
  const emailValidNotificationElement = loginNotificationProjector()
  setupEmailValidNotification(login, emailValidNotificationElement, emailInputElement)

  const loginNotificationElement = loginNotificationProjector()
  setupLoginNotification(login, loginNotificationElement)
  loginNotificationElement.id = 'loginNotification'

  // -----Show Button----- 5
  const showButtonElement = loginShowButtonProjector(login)

  // -----Forgot Email or Password Link----- 6
  const forgotLinkElement = loginLinkProjector('Forgot email or password?')

  // -----Container Elements-----
  const emailInputContainer = containerProjector([emailInputElement, emailValidNotificationElement], 'emailInputContainer')
  const passwordInputContainer = containerProjector([passwordInputElement, showButtonElement], 'passwordInputContainer')

  // -----Form Element----- 7
  const formElement = document.createElement('form')

  // -----Setting up the HTML----- 8
  rootElement.appendChild(titleElement)
  rootElement.appendChild(formElement)

  formElement.appendChild(loginNotificationElement)
  formElement.appendChild(emailLabelElement)
  formElement.appendChild(emailInputContainer)
  formElement.appendChild(passwordLabelElement)
  formElement.appendChild(passwordInputContainer)
  formElement.appendChild(forgotLinkElement)
  formElement.appendChild(loginButtonElement)
}
```

Abbildung 78: Hauptprojektor

Die acht Segmente zeigen deutlich den Aufbau des Hauptprojektors auf. Im achten Segment wird die von den Projektoren erstellten Elemente im DOM eingebunden. In den Segmenten eins bis sieben sieht man die Anwendung von Subprojektoren, auf die anschliessend vertiefter eingegangen werden.

Das vierte Segment zeigt noch eine weitere Massnahme zur Übersichtlichkeit der Implementation auf. Setups wurden für spezielle Funktionalitäten für spezifische UI-Elemente verwendet. Soll zum Beispiel genau dieses Notification-Element eine spezielle Aufgabe erfüllen, wurde dies in einer Setup Funktion verlagert. Funktionen, die hingegen für alle Notification Elemente gelten, sind im Projektor. Auf diese Auseinandersetzung wird nun erweitert eingegangen.

## Subprojektoren

Die Subprojektoren können sowohl sehr einfach als auch sehr komplex aufgebaut sein. Dies bedeutet, dass einfache Subprojektoren nichts weiter als nur ein HTML-Element erstellen können und komplexere Subprojektoren zusätzlich mehrere Abhängigkeiten und Funktionalitäten einbinden.

In den Abbildungen 79 und 80 sieht man je ein Beispiel eines einfachen und eines komplexen Subprojektor.

```
/**
 * Generates a header 2 Element for the Login component
 * @returns {HTMLElement}
 */
const loginTitleProjector = () => {
  const h2Element = document.createElement('h2')
  h2Element.innerHTML = 'Login'
  return h2Element
}
```

Abbildung 79: Einfacher Subprojektor, Titel der Komponente

```
/**
 * Generates an input element of type submit. Additionally binds various values to the attribute and event listeners.
 * @param {object} Login - Holds all attributes of the Login model
 * @returns {HTMLElement}
 */
const loginSubmitButtonProjector = Login => {
  const inputElement = document.createElement('input')
  inputElement.type = 'submit'
  inputElement.value = 'Login'

  inputElement.classList.add('primary')

  inputElement.onclick = event => {
    event.preventDefault()

    login.onLogin(login.getEmail(), login.getPassword())
      .then(result => {
        result.token // Response should contain a token if login was successful
        ? login.setLoginSuccess(true)
        : login.setLoginSuccess(false)
      })
      .catch(err => {
        login.setLoginSuccess(false)
      })
  }

  login.onFormValidityChanged(
    valid => valid
    ? inputElement.disabled = false
    : inputElement.disabled = true
  )

  return inputElement
}
```

Abbildung 80: Komplexer Subprojektor, Submit Button

## Setups

Wie bereits erwähnt, werden in den Setups spezifische Funktionalitäten einzelner Elemente hinzugefügt. In der Login Komponente existieren zwei verschiedene Notification-Elemente. Eines, um den User zu benachrichtigen, wenn seine eingegebene E-Mail-Adresse invalide ist und eines, um den User zu informieren, ob sein Login Versuch erfolgreich war oder nicht. Beide Elemente haben also dieselbe Aufgabe, aber unterschiedliche Ausführungsarten. Aus diesem Grund werden diese spezifischen Funktionalitäten in sogenannten Setups übertragen.

```
/**
 * Checks whether the email is valid or not. Notification gets only updated when the input field loses focus.
 * @param {object} login - Holds all attributes of the login model
 * @param {HTMLElement} notificationElement - A paragraph HTMLElement which contains messages for the user
 * @param {HTMLElement} emailInputElement - The email input Element which is being validated whenever its changed
 */
const setupEmailValidNotification = (login, notificationElement, emailInputElement) => {
  emailInputElement.addEventListener('change', () => {
    if(!login.getEmail()) return notificationElement.innerHTML = ''
    login.getEmailValidity()
      ? notificationElement.innerHTML = ''
      : notificationElement.innerHTML = 'Malformed Email'
  })
}
```

Abbildung 81: Setup der Benachrichtigung der E-Mail Validation

```
/**
 * Checks whether the login was successful or not and updates notification to the user.
 * @param {object} login - Holds all attributes of the login model
 * @param {HTMLElement} notificationElement - A paragraph HTMLElement which contains messages for the user
 */
const setupLoginNotification = (login, notificationElement) => {
  login.onLoginSuccessChanged( () => {
    if(login.getLoginSuccess()) {
      login.setNotification('Logged in successfully!')
      notificationElement.classList.add('success')
    } else if(login.getPassword()){
      login.setNotification('Sorry we could not process your login attempt')
      notificationElement.classList.remove('success')
    } else {
      login.setNotification('')
      notificationElement.classList.remove('success')
    }
  })

  login.onNotificationChanged( () => notificationElement.innerHTML = login.getNotification())
}
```

Abbildung 82: Setup der Login Benachrichtigung



## View

Die View ist in diesem Fall eine sehr kleine Instanz und hat die Aufgabe, dem Projektor die benötigte Information zu übergeben und die Komponente zur richtigen Zeit zu rendern. Diese befindet sich ebenfalls in der login.js Datei.

```
/**
 * Renders the Login component as soon as a new Login is being added
 * @param {LoginController} loginController
 * @param {HTMLElement} rootElement - The root element which will contain the whole Login component
 */
const LoginView = (loginController, rootElement) => {
  const render = login => loginProjector(loginController, rootElement, login)
  loginController.onLoginAdd(render)
}
```

Abbildung 83: View

Als weitere Referenz zum Projektor-Pattern existiert ein Vortrag von Herrn Dierk König an der W-Jax 2015. In seiner Präsentation stellt er das Projektor-Pattern vor und demonstriert diese mit einer JavaFX Anwendung.

Die Session kann unter dem folgenden Link aufgerufen werden: <https://entwickler.de/java/effiziente-oberflaechen-mit-dem-projektor-pattern/>

## 4. Proof of Concept

In diesem Proof of Concept, wird die Hauptstärke des Projektor-Patterns aufgezeigt. Es geht hauptsächlich darum, dass im Toolkit einige Komponente für mehrere verschiedene Anwendungszwecke verwendet werden sollen. Dafür soll lediglich das Aussehen und die Funktionalität der Komponente angepasst werden. Das Model und die Abhängigkeiten hingegen bleiben die gleichen. Dies wird beim Projektor-Pattern mit den sogenannten Projektoren realisiert.

Für das Proof of Concept wurde ein einfaches Beispiel eines Formulars gewählt. Es wurden zwei verschiedene Formulare erstellt, die sich sowohl beim Aussehen als auch bei einigen Funktionen unterscheiden.

Für den Aufbau der Forms wurden einige Änderungen beim Model vorgenommen. Wie man in der Abbildung 84 sieht, werden "Attribute Configs" verwendet, um die Generierung eines Forms zu vereinfachen.

```
/**
 * All configurations which are necessary for the attribute and the model, to correctly display the form
 * @type {object[]}
 */
const ALL_ATTRIBUTE_CONFIGS = [
  { id: 'firstname', type: 'text', placeholder: 'Placeholder' },
  { id: 'lastname', type: 'text' },
  { id: 'eyeColor', type: 'color' },
  { id: 'birthDate', type: 'date' },
  { id: 'date', type: 'date' },
  { id: 'time', type: 'time' },
  { id: 'number', type: 'number' },
  { id: 'color', type: 'color' },
  { id: 'range', type: 'range' },
  { id: 'phone', type: 'tel', placeholder: '123-45-678' },
  { id: 'radio1', type: 'radio', name: 'radio-example' },
  { id: 'radio2', type: 'radio', name: 'radio-example' },
  { id: 'favColor', type: 'text' },
  { id: 'submit', type: 'submit' },
  { id: 'place', type: 'text' },
]
```

Abbildung 84: Attribute Configs

Diese Konfigurationen enthalten die Informationen für die einzelnen Input Felder des Formulars. Diese werden anschliessend bei der Erstellung der View an den Projektor übergeben.

```
/**
 * Renders the form as soon as a form is being added
 * @param {FormController} FormController
 * @param {HTMLElement} rootElement - The root element which will contain the whole form
 */
const FormView = (FormController, rootElement) => {
  const render = form => formProjector(FormController, rootElement, form, ALL_ATTRIBUTE_CONFIGS)
  FormController.onFormAdd(render)
}
```

Abbildung 85: Übergabe der Konfigurationen über die View

Für das Proof of Concept wurde auch die `setGroup()` Funktion der Attribute ausgenutzt, damit einzelne Attribute in Gruppen beigefügt werden können und dementsprechend deren Elemente im User Interface

gruppiert angezeigt werden können. Somit kann man Input Felder, die für gewisse Themengebiete wie Personalien, Meeting oder weiteres eingruppiert werden.

#### 4.1 Form Varianten

In diesem Abschnitt werden die Unterschiede der beiden Formular-Varianten aufgezeigt und anschliessend beschrieben, wie man zwischen den Varianten wechseln kann.

##### 4.1.1 Formular Variante 1

The form is titled "Formular Variante 1" and is divided into two main sections: "Personalia" and "Meeting".

**Personalia Section:**

- First Name:
- Last Name:
- Eye Color:
- Date of Birth:
- Number:
- Color:
- Range:
- Phone:
- Radio 1:
- Radio 2:

**Meeting Section:**

- Date:
- Time:
- Fav. Color:
- Place:

At the bottom of the form is a button labeled "Submit this form".

Abbildung 86: UI der ersten Formular Variante

In der ersten Variante sieht man nun, dass gruppierte Elemente mit einem grauen Hintergrund versehen sind und einen Übertitel haben. Elemente, die keiner Gruppe angehören haben weder einen Übertitel noch eine Hintergrundfarbe. Das Layout der Form besteht aus zwei Spalten. Eine zusätzliche Funktionalität wurde dem Input Feld "Fav. Color" gegeben, und zwar wird dort die Eingabe mit Vorschlägen unterstützt, wie man es in der Abbildung 87 sieht.

This image shows a close-up of the "Fav. Color" input field. The field is a text input with a dropdown arrow on the right. A dropdown menu is open, displaying four color suggestions: "Yellow", "Red", "Orange", and "Green". The suggestions are listed vertically on a dark background. To the left of the dropdown, a "Submit this" button is partially visible.

Abbildung 87: Eingabevorschläge beim Fav. Color Input Feld

Diese Funktionalität wird oft in Domänen verwendet, die häufig ähnliche Eingaben erfordern, wie zum Beispiel bei Business-Controls.

#### 4.1.2 Form Variante 2

The image shows a web form with two main sections: 'Meeting' and 'Personalia'. The 'Meeting' section contains four input fields: 'Date' (with a date mask 'tt.mm.jjjj' and a calendar icon), 'Time' (with a time mask '--:--' and a clock icon), 'Fav. Color' (an empty text input), and 'Place' (an empty text input). The 'Personalia' section contains four input fields: 'First Name' (with a 'Placeholder' text), 'Last Name' (with 'SomeLastName' text), 'Eye Color' (with a blacked-out text), and 'Date of Birth' (with a date mask 'tt.mm.jjjj' and a calendar icon). Below these sections is a 'Submit this form' button.

Abbildung 88: UI der zweiten Formular Variante

Wir sehen nun, dass das zweite Formular dieselben Elemente und Gruppierungen besitzt. Jedoch unterscheiden die sich sowohl beim Aussehen als auch in der Funktionalität. So wurde zum Beispiel die Gruppierung mit einem Rahmen versehen und nicht gruppierte Elemente ausgelassen. Ebenfalls besteht das Layout aus nur einer Spalte. Beim Input Feld "Fav. Color" wurde auch die Funktionalität mit den vorgeschlagenen Eingaben entfernt. Zusätzlich wurde auch die Reihenfolge der Gruppen verändert.

Das Einzige, was sich an der Implementation geändert hat, sind die Projektoren. Dies zeigt, wie einfach bestehende Komponente für neue Domänen abgeändert und eingesetzt werden können. Um zwischen den Varianten zu wechseln, muss lediglich der Import beim Controller geändert werden.

```
import { formProjector, pageCss } from './mainProjector/formProjectorV1.js'  
import { formProjector, pageCss } from './mainProjector/formProjectorV2.js'
```

Abbildung 89: Import beider Form Varianten

Die Implementierung der beiden Projektoren können unter diesem Link eingesehen werden:  
<https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/projector-pattern/form/mainProjector>

# 5. Testing

Für das Testen der Komponenten wurde ein eigenes Test-Framework erstellt, welches sich an dem Test-Framework aus dem Modul WebClient orientiert [22]. Das Framework hat grosse Ähnlichkeiten mit dem bekannten Test-Framework namens Jest. Das Testing ist so aufgebaut, dass man in einer Testing-Suite die einzelnen Tests hinzufügt und diese am Ende nacheinander durchlaufen lässt. Das Ergebnis wird, anders als bei Jest, nicht auf der Konsole ausgegeben, sondern auf eine HTML-Seite im Browser angezeigt. Für die Komponente wurden Integrationstests durchgeführt, um zu gewährleisten, dass die Komponenten als Ganzes einwandfrei funktionieren.

Um die Funktionalität der Tests besser erklären zu können, werden sowohl das Framework als auch Testbeispiele aufgezeigt.

## 5.1 Test-Framework

Das Framework kann in drei Bestandteile eingegliedert werden. Diese sind die Suites, Asserts und die Auswertung und Ausgabe der Tests.

Den Code für das Framework kann unter diesem Link eingesehen werden:

<https://github.com/fabianhaef/ip6-web-ui-toolkit/blob/projector-pattern/test/test.js>

### **Suite**

Eine *Suite* beschreibt, in welchem Kontext die Tests geschrieben werden. Diese beinhaltet alle dazugehörigen Tests und führt diese am Ende einzeln durch. Somit können Tests, die der Login Komponente gehören, in einer Suite mit dem Namen "Login" hinzugefügt werden. Dies soll zur Verbesserung der Übersichtlichkeit und der Auswertung des Tests beitragen, da man so schneller erkennen kann, welche Tests zu welcher Komponente angehören.

```

const Suite = (suiteName) => {
  const tests = []
  const suite = {
    test: (testName, callback) => test(`${suiteName} - '${testName}'`, callback),
    add: (testName, callback) => tests.push(Test(testName)(callback)),
    run: () => {
      const suiteAssert = Assert()
      tests.forEach(test => test(logic)(suiteAssert))
      total += suiteAssert.results.length
      if(suiteAssert.results.every(id)){
        report(`suite ${suiteName}`, suiteAssert.results)
      } else {
        tests.forEach(test => suite.test(test(name), test(logic)))
      }
    }
  }
  return suite
}

```

Abbildung 90: Implementation der Suite

Wie man in der Abbildung 90 sieht, stellt die Suite drei Funktionen, *test(testName, callback)*, *add(testName, callback)* und *run()*, zur Verfügung. Mit der *test(...)* Funktion können Tests direkt ausgeführt werden, sodass man die *run()* Funktion nicht aufrufen muss. Diese Tests werden jedoch nicht gebündelt und werden einzeln ausgegeben. Bei der *add(...)* Funktion jedoch werden die Tests zuerst in einem Array gespeichert und erst beim Aufrufen der *run()* Funktion werden diese sequenziell ausgeführt und auf dem Browser ausgegeben. Um einen Vergleich zeigen zu können, wurden drei Beispieltests mit der *test*-Funktion erstellt, die in der Abbildung 91 zu sehen sind.

```

const testSuite = Suite('testSuite')

testSuite.test('test1', assert => {
  assert.true(true)
  assert.is(1, 1)
})

testSuite.test('test2', assert => {
  assert.true(true)
  assert.is(1, 1)
})

testSuite.test('test3', assert => {
  assert.true(true)
  assert.is(1, 1)
})

```

Abbildung 91: Beispieltests mit der test-Funktion

Insgesamt werden drei Tests ausgeführt mit je zwei *Asserts*. Diese werden wie folgt auf dem Browser ausgegeben.

```

2 tests in testSuite - 'test1' ok
2 tests in testSuite - 'test2' ok
2 tests in testSuite - 'test3' ok

```

Abbildung 92: Test Suite Ausgabe mit test-Funktion

Man erkennt schnell, dass bei einer grösseren Applikation die Liste schnell gross und unübersichtlich werden kann. Zum Vergleich dazu wurden die Tests der Register Komponente ausgeführt, die aus 12 Tests mit insgesamt 48 *Asserts* besteht.

```

48 tests in suite register      ok

```

Abbildung 93: Test Suite Ausgabe mit add-Funktion

Man sieht auf einem Blick, dass alle Tests erfolgreich sind, wogegen man beim vorherigen Beispiel die Liste etwas genauer anschauen müsste, um sicher zu sein, dass alle Tests korrekt sind. Wenn ein Test fehlschlägt, sieht dies jedoch bei beiden Fällen gleich aus.

```

2 tests in testSuite - 'test1' ok
! Failing tests in testSuite - 'test2'
! Test # 1 failed
2 tests in testSuite - 'test3' ok

```

Abbildung 94: Ein fehlschlagender Test mit der test-Funktion

```

4 tests in register - 'Email input gets validated correctly' ok
8 tests in register - 'Password input gets validated correctly' ok
2 tests in register - 'Correct Email notification gets displayed when email is invalid' ok
10 tests in register - 'Password is not valid if criterion is not met' ok
1 tests in register - 'Correct notification gets displayed when password and confirm password are the same' ok
1 tests in register - 'Correct notification gets displayed when password and confirm password are NOT the same' ok
1 tests in register - 'Correct notification gets displayed when password and confirm password are almost the same' ok
3 tests in register - 'Form becomes valid when email and password is valid' ok
3 tests in register - 'Form becomes invalid when email is valid but password is invalid' ok
3 tests in register - 'Form becomes invalid when email and password is invalid' ok
! Failing tests in register - 'Show buttons toggle password input type'
! Test # 6 failed
6 tests in register - 'PaswordStrength changes when password get changed ' ok

```

Abbildung 95: Ein fehlschlagender Test mit der add-Funktion

## Assert

Die *Assert*-Funktion ist der wichtigste Bestandteil des Test-Frameworks. Mit dieser Methode werden verschiedene Funktionalitäten, Ergebnisse und Methoden einer Komponente überprüft. Mit unterschiedlichen Funktionen können verschiedene Zustände überprüft werden, wie zum Beispiel ob eine Aussage wahr oder falsch ist. In der Abbildung 96 kann man die Implementation der *Assert*-Funktion einsehen.

```

const Assert = () => {
  const results = []
  return {
    results: results,
    true: (testResult) => {
      if(!testResult) { console.error('test failed') }
      results.push(testResult)
    },
    is: (actual, expected) => {
      const testResult = actual === expected
      if(!testResult) {
        console.error(`test failure. Got '${actual}', expected '${expected}'`)
      }
      results.push(testResult)
    },
    isNot: (actual, expected) => {
      const testResult = actual !== expected
      if(!testResult) {
        console.error(`test failure. Got '${actual}', expected '${expected}'`)
      }
      results.push(testResult)
    }
  }
}

```

Abbildung 96: Implementation der Asserts

Wie man sieht, wurden drei verschiedene Testmethoden implementiert. Die *true(testResult)* Funktion überprüft, ob eine Aussage wahr oder falsch ist. Die *is(actual, expected)* Funktion testet auf die Gleichheit beider Einträge und die *isNot(actual, expected)* auf deren Ungleichheit. Man kann die *Assert*-Funktion mit weiteren Testmethoden beliebig erweitern. Die Ergebnisse werden anschliessend einem Array hinzugefügt.

### Auswertung und Ausgabe der Tests

Dieser Teil des Frameworks ist zuständig dafür, dass die Ergebnisse korrekt auf dem Browser angezeigt werden. Die Implementierung ist auf der folgenden Abbildung 97 zu sehen.



```

const report = (origin, ok) => {
  const extend = 20
  if(ok.every(elem => elem)){
    write(` ${padLeft(ok.length, 3)} tests in ${padRight(origin, extend)} ok`)
    return
  }
  let reportLine = `Failing tests in ${padRight(origin, extend)}`
  write(` ! ${reportLine}`)
  for(let i = 0; i < ok.length; i++){
    if(!ok[i]){
      write(` ! Test #${padLeft(i+1, 3)} failed`)
    }
  }
}

const write = (message) => {
  const out = document.getElementById('out');
  out.innerHTML += message + "\n";
}

```

Abbildung 97: Implementation der *Report* und *Write* Funktionen

## 5.2 Testbeispiele der Login Komponente

Um den Ansatz für das Schreiben neuer Tests für weitere Komponenten erklären zu können, werden die Tests der Login Komponente als Beispiel genommen.

Die Tests können unter diesem Link eingesehen werden: <https://github.com/fabianhaef/ip6-web-ui-toolkit/blob/projector-pattern/login/test/loginTest.js>

Die Tests sind in zwei Segmenten aufgeteilt. Ein Segment besteht aus dem Setup, um die Komponente zu erstellen und aus den Tests, welche die Funktionalität der Komponente testet.

### 5.2.1 Setup

Im Setup wurden sowohl die *Suite* für die Tests initialisiert als auch die Komponente selbst erstellt. Die Erstellung der Komponente wurde in eine Funktion gepackt, damit diese in den Tests aufgerufen werden können. Der Grund dafür liegt daran, dass wir noch keine Lifecycle Methoden wie *beforeEach()* oder *afterEach()*, in unserem Framework implementiert haben. Das Setup sieht wie folgt aus.

```

// Setup context
const loginSuite = Suite('login')

const setUpLoginContext = (withContainer = false) => {
  const loginContainer = document.createElement('div')

  const service = loginService()
  const loginController = LoginController(service)

  LoginView(loginController, loginContainer)

  const Login = loginController.addLogin()

  return withContainer ? [Login, loginContainer] : Login
}

```

Abbildung 98: Test Setup

Bei der Erstellung der Komponente wird als Rückgabewert das Login Model und falls gewünscht auch das Container-Element, welches die Komponente enthält, zurückgegeben. Somit können über das Model auf die Attribute der Komponente zugegriffen werden und über das Container-Element auf die DOM-Elemente der Komponente.

## 5.2.2 Tests

```

loginSuite.add('Email input gets validated correctly', assert => {

  const Login = setUpLoginContext()

  Login.setEmail('valid@mail.com')

  assert.is(Login.getEmail(), 'valid@mail.com')
  assert.true(Login.getEmailValidity())

  Login.setEmail('invalidMail.com')

  assert.is(Login.getEmail(), 'invalidMail.com')
  assert.true(!Login.getEmailValidity())
})

```

Abbildung 99: Implementation eines Tests der Login Komponente mit der Verwendung des Models

In der Abbildung 99 sieht man, wie die Komponente zu Beginn erstellt und das Model verwendet wird, um verschiedene Werte einzugeben. In diesem Fall wird überprüft, ob die Validierung der E-Mails korrekt ausgewertet wird. Auch ersichtlich ist, dass wir alle unsere Tests der *loginSuite* hinzugefügt haben. Die Tests werden am Ende der Datei mit *loginSuite.run()* ausgeführt und auf dem Browser ausgegeben.

```

loginSuite.add('User gets notified when login attempt has failed', assert => {

  const [Login, loginContainer] = setUpLoginContext(true)

  Login.setEmail('some@mail.com')
  Login.setPassword('somePassword')

  const emailInput = loginContainer.querySelector('#email')

  fireEvent(emailInput, 'change')
  fireEvent(loginContainer.querySelector('#password'), 'input')

  const loginBtn = loginContainer.querySelector('input[type="submit"]')

  loginBtn.click()

  Login.setLoginSuccess(false)

  assert.is(Login.getEmail(), 'some@mail.com')
  assert.is(Login.getPassword(), 'somePassword')
  assert.is(Login.getNotification(), 'Sorry we could not process your login attempt')
})

```

Abbildung 100: Implementation eines Tests der Login Komponente mit der Verwendung des Models und des Container Elements

Im zweiten Beispiel in der Abbildung 100 sieht man die Anwendung des Container-Elements. Hier wird überprüft, ob die Notifikation die korrekte Nachricht für den User enthält. Die *fireEvent()* Funktion, welches hier verwendet wurde, ist eine selbst erstellte Hilfsmethode, um Events auf bestimmte HTML Elemente abfeuern zu können. Die Implementation dazu kann in der folgenden Abbildung 101 eingesehen werden.

```

const fireEvent = (el, eventType) => {
  const event = new Event(eventType)
  el.dispatchEvent(event)
}

```

Abbildung 101: Implementation der FireEvent Funktion

### 5.2.3 Ausgabe auf dem Browser

Die dazu gehörige HTML-Datei sieht wie folgt aus.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title> Login Tests </title>
</head>
<body>

  <pre id="out"></pre>

  <script type="module" src="./loginTest.js"></script>

</body>
</html>
```

Abbildung 102: HTML-Datei für Tests

Die Seite ist sehr einfach und schnell aufgebaut. Es werden lediglich die zwei Elemente `pre` und `script` benötigt. Das `pre`-Element mit der `id` "out" enthält die ausgegebenen Ergebnisse aller Tests und das `script`-Element verweist auf die Testdatei der zu testenden Komponente.

## 6. Weiteres Vorgehen

### 6.1 Mehr Komponenten aus verschiedenen Domänen

Damit das UI-Toolkit für ein breiteres Feld von Anwendungen interessant ist, sollte es um mehr Komponenten aus den Domänen Business-Controls, Scientific-Controls und Engineering-Controls ergänzt werden. Für unsere Projektarbeit haben wir als Basis für das Web-UI-Toolkit allgemein verwendbare Komponenten erstellt und ein Konzept für die Weiterentwicklung erstellt. Auf diesem Konzept kann nun aufgebaut werden damit das Toolkit eine reichhaltige Auswahl an hochwertigen Komponenten anbietet.

### 6.2 Veröffentlichbarkeit des Projektor-Pattern als CDN

Die erarbeiteten Komponenten müssen noch in ein CDN verpackt werden, damit diese von Webentwicklern verwendet werden können.

### 6.3 Kompatibilität mit existierenden Frameworks und Libraries

Damit unser Web-Ui-Toolkit frei verwendbar ist und auch mit existierenden Frameworks und Libraries wie zum Beispiel Django oder React verwendet werden kann, muss geprüft werden, ob unsere Komponenten in diesen verwendbar sind.

### 6.4 Open Source Lizenz

Um das Web-UI-Toolkit als Open-Source Projekt zu veröffentlichen, muss eine passende Open-Source Lizenz evaluiert und ausgewählt werden. Damit ein Projekt Open-Source ist muss die Software frei sein, modifizierbar sein und frei geteilt werden können. Es gibt eine Vielzahl von existierenden Lizenzen, welche sich in einigen Aspekten unterscheiden.

### 6.5 ReadMe.md für Release

Es gehört standardmässig zu einem Open-Source Projekt dazu eine ReadMe-Datei zu erstellen. Darin werden Informationen zur Installation, Konfiguration und Lizenz und weiteren dokumentiert. Das Dokument muss nicht unbedingt die einzige Dokumentation sein, bildet aber die Grundlage, um das UI-Toolkit zu installieren und grundlegende Konzepte zu verstehen.

# 7. Reflexion

In diesem Kapitel reflektieren wir unsere Projektarbeit. Der Abschnitt Ergebnisse umfasst das von uns erreichte Ergebnis. Anschliessend werden wir unsere Erkenntnisse aus dieser lehrreichen Arbeit darlegen.

## 7.1 Ergebnisse

### 7.1.1 Design-Guide

Damit unser Web-Ui-Toolkit ansprechend und durchgehend gleich aussieht haben wir einen Design-Guide erstellt. An diesem können sich nachfolgende Gruppen orientieren und Ihre Komponenten anhand dieser Guidelines entwickeln. Der Design-Guide enthält Angaben zum Branding wie Farben und Typographie. Zusätzlich enthalten sind Beispielkomponente sowie Designkomponenten spezifisch für das Designtool Figma. Damit können in kürzester Zeit neue Prototypen erstellt werden.

### 7.1.2 Variantenvergleich Auslieferung / Verpackung

Wir haben verglichen, in welcher Form wir unser Toolkit an die Entwickler ausliefern wollen und wie diese verpackt sind. Dabei haben wir die verschiedenen Möglichkeiten angeschaut und auf ihre Vor- und Nachteile verglichen.

### 7.1.3 Projektor-Pattern

Als Verpackungsform für unser Toolkit haben wir uns für das Projektor-Pattern entschieden (siehe Abschnitt 3.3.3 Projektor-Pattern). Wir haben dafür eine Codebasis erstellt, welche einfach und kostengünstig um weitere Komponenten ergänzt werden kann. Unsere beiden Komponenten – Login und Registrierung – haben wir beide ebenfalls bereits im Projektor-Pattern erstellt.

### 7.1.4 Proof of Concept

In unserem Proof of Concept haben wir gezeigt, dass Komponenten schnell und einfach durch andere Komponenten ersetzt werden können. Dies erleichtert das Erstellen von Oberflächen erheblich, da dadurch der Entwicklungsaufwand verringert wird und Komponenten mehrfach verwendet werden können.

### 7.1.5 Testframework

Um unsere Komponenten zu testen haben wir ein eigenes Testframework aufgebaut. Dieses enthält eine Testsuite, welche verschiedene Methoden zur Verfügung stellt, um Komponenten zu testen. Das Testframework kann je nach Bedürfnissen angepasst werden, damit jeder Komponente entsprechend getestet werden kann.

## 7.2 Erkenntnisse

### 7.2.1 Erstellen des Design-Guides

Da wir diesen Prozess noch nie selbst durchlaufen konnten, haben wir vieles dabei gelernt. Der gestalterische Teil hat uns sehr viel Freude gemacht, da wir das Prototyping der Komponenten und abstimmen der einzelnen Farben sehr interessant und lehrreich fanden. Uns wurde klar, dass mit einem guten Design ein Produkt hochwertiger und besser erscheinen lässt.

### 7.2.2 Verteilung / Verpackung

Wir haben uns bisher noch nie sehr stark mit der Verteilung von Open-Source Software beschäftigt. Darum war es für uns sehr lehrreich uns darüber Gedanken zu machen. Uns wurde bewusst, dass jede Vorgehensweise ihre Vor- und Nachteile mit sich bringt.

### 7.2.3 JavaScript ES6

Die Verwendung des JavaScript ES6 Standards war für uns ein wichtiger Punkt bei der Projektauswahl. Durch das viele Pair-Programming und die Besprechungen mit unseren Kunden konnten wir unsere ES6-Fähigkeiten verbessern.

# Anhang A – Kontaktdaten

## **Kunde**

Name Fachhochschule Nordwestschweiz, Prof. Dierk König  
E-Mail [dierk.koenig@fhnw.ch](mailto:dierk.koenig@fhnw.ch)  
Adresse Bahnhofstrasse 6, 5210 Windisch

## **Betreuer**

Name Prof. Dierk König  
E-Mail [dierk.koenig@fhnw.ch](mailto:dierk.koenig@fhnw.ch)  
Firma FHNW  
Adresse Bahnhofstrasse 6, 5210 Windisch

Name Fabian Affolter  
E-Mail [fabian.affolter@fhnw.ch](mailto:fabian.affolter@fhnw.ch)  
Adresse Bahnhofstrasse 6, 5210 Windisch



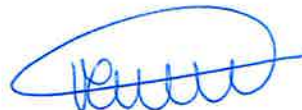
# Anhang B – Ehrlichkeitserklärung

Hiermit erklären wir, die vorliegende Projektarbeit selbständig, ohne Hilfe Dritter und nur unter Benutzung der angegebenen Quellen verfasst zu haben.

Brugg – 20.08.2021



Dusan Mistic



Fabian Häfliger

# Glossar

**Achromatopsie:** Unfarbigkeit (totale Farbenblindheit)

**afterEach:** Lifecycle Methode die hauptsächlich in Test-Frameworks bekannt ist. Diese wird nach jedem einzelnen Test ausgeführt. Dies erlaubt es gewisse Anweisungen, die nach jedem Test ausgeführt werden müssen, nur einmal aufzuschreiben.

**API:** Als API (Application Programming Interface) wird eine Programmierschnittstelle bezeichnet. Diese macht es möglich, dass Software andere Software verwendet, indem sie auf diese Schnittstelle zugreift.

**beforeEach:** Lifecycle Methode die hauptsächlich in Test-Frameworks bekannt ist. Diese wird vor jedem einzelnen Test ausgeführt. Dies erlaubt es gewisse Anweisungen, die vor jedem Test ausgeführt werden müssen, nur einmal aufzuschreiben.

**Derived Attribute:** Derived Attribute: Dieser Begriff wird häufiger im Bereich der Datenmanagementsystem verwendet als in der Softwareentwicklung. Dabei handelt es sich um Attribute, die auf eine oder mehrere andere Attribute basieren und können somit von denen abgeleitet werden. [23]

**Deuteranopie:** Gründblindheit

**Div-Element:** Ein Div-Element ist ein HTML-Element, welches als ein allgemeiner Container verwendet wird. Dem Inhalt wird keine semantische Bedeutung zugeordnet. [24]

**Django:** Django ist ein Web-Framework, welches auf die Programmiersprache Python basiert. [25]

**DOM:** Das DOM, oder Englisch für Dokumenten-Objekt-Modell, stellt HTML oder XML-Dokumente als eine Baumstruktur dar und ist für den Programmierer zugänglich. So können einzelne Objekte, die in der DOM enthalten sind, angesprochen und verändert werden. [26]

**Entwickler-Tools:** Ein Werkzeug, welches in jedem Webbrowser enthalten ist. Damit können die gerade angezeigten Webseiten auf ihre Struktur und Aufbau analysiert werden. Dieses Werkzeug ist essenziell für einen Webentwickler. [27]

**Figma:** Ein Webbasiertes Design Tool, welches für die Erstellung von Designkonzepten und Prototypen verwendet werden kann.

**Frontend:** Teil der Applikation, mit welcher der User interagieren kann, zum Beispiel eine grafische Oberfläche.

**Kommandozeile:** Eine Kommandozeile, auch Befehlszeile genannt, ist ein Programm, welches eine Textzeile als Eingabe entgegennimmt und daraus bestimmte Befehle ausführt oder Werte zurückgibt. [28]

**Monochromatisches Farbschema:** Ein monochromatisches Farbschema enthält eine Primärfarbe, welche mit Grautönen und Hilfsfarben ergänzt wird. Die Primärfarbe wird in verschiedenen Tönen mit unterschiedlicher Deckkraft verwendet.

**MVC:** Das MVC ist ein weit verbreitetes Entwurfsmuster, welches für eine effiziente Entwicklung von Benutzeroberflächen verwendet wird. Hiermit wird der Aufbau in drei Hauptkomponente Model, View und Controller aufgeteilt. Dies kontrolliert den Datenfluss und gewährleistet eine bessere Übersichtlichkeit des Projekts. [29]

**Observer-Pattern:** Das Observer-Pattern ist ein weit verbreitetes Entwurfsmuster, das verwendet wird, um eine oder mehrere Beobachter über Änderungen eines Subjekts zu informieren. Ohne dieses Pattern müssten die Beobachter regelmäßig Abfragen über den Zustand des Subjekts ausführen. Dies verbraucht jedoch unnötige Rechenleistung. [30]

**Pair-Programming:** Pair-Programming ist eine agile Arbeitstechnik, bei der zwei Programmierer zusammenarbeiten. Ein Programmierer schreibt den Code, während der andere sich der Problemstellung widmet.

**Plug-In:** Ein Softwareprogramm, welches die Funktionalität einer anderen Software erweitert. [31]

**Protanopie:** Rotblindheit

**Python:** Eine Programmiersprache die oft als Skriptsprache verwendet wird. [32]

**React.js:** React.js ist eine Javascript Softwarebibliothek zum Erstellen von komponentenbasierten Frontend-Applikationen.

**ReadMe:** Eine Datei, die in der Softwareentwicklung oft verwendet wird, um allgemeine Informationen oder Beschreibungen über ein Projekt festzuhalten. Es können normale Text Dateien oder Markdown Dateien sein.

**Shadow-DOM:** Ein Shadow-DOM ist eine von aussen nicht erreichbare DOM. Diese sind vom Programmierer nicht zugriff- und veränderbar.

**Tritanopie:** Blaublindheit

**Usability Test:** Ein Test, die bestimmte Nutzergruppen zu bestimmten Handlungen auf einer Applikation auffordert. Die Nutzer können währenddessen beobachtet werden oder müssen Fragen bezüglich der Einfachheit der Applikation beantworten. Die Ergebnisse können allfällige Mängel an Verständnis in einer Applikation aufdecken, die anschliessend verbessert werden können.

# Akronyme

**API:** Application Programming Interface

**CDN:** Content Delivery Network

**DOM:** Document Object Model

**MVC:** Model View Controller

**NPM:** Node Package Manager

# Linksammlung:

Sämtliche aufgeführte Links wurden im Verlaufe der Arbeit erwähnt.

- Designguide    [figma.com, Kolibri Designguide, URL: https://www.figma.com/community/file/1006878219241348539/Kolibri-Styleguide](https://www.figma.com/community/file/1006878219241348539/Kolibri-Styleguide)
- Repository     [github.com, IP6-Web-UI-Toolkit Repository, URL: https://github.com/fabianhaef/ip6-web-ui-toolkit](https://github.com/fabianhaef/ip6-web-ui-toolkit)
- Seite 19:        [docs.google.com, Usability Test Antworten, URL: https://docs.google.com/forms/d/1HjBBkD2XP8qSAnVEOCwynZsPwsG-FXRDrTQ1kI2EBj8/edit#responses](https://docs.google.com/forms/d/1HjBBkD2XP8qSAnVEOCwynZsPwsG-FXRDrTQ1kI2EBj8/edit#responses)
- Seite 37:        [npmjs.com, Kolibree, URL: https://www.npmjs.com/package/kolibree](https://www.npmjs.com/package/kolibree)
- Seite 37:        [github.com, Kolibree, URL: https://github.com/fabianhaef/Kolibree](https://github.com/fabianhaef/Kolibree)
- Seite 39:        [github.com, IP6-Web-UI-Toolkit, URL: https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/flafd6385ffe5c45e4f09b7beece7b72e1f88620](https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/flafd6385ffe5c45e4f09b7beece7b72e1f88620)
- Seite 39:        [github.com, IP6-Web-UI-Toolkit, URL: https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/cdn-production](https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/cdn-production)
- Seite 42:        [github.com, IP6-Web-UI-Toolkit, URL: https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/webcomponent](https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/webcomponent)
- Seite 47:        [github.com IP6-Web-UI-Toolkit, URL: https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/login](https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/login)
- Seite 47:        [github.com IP6-Web-UI-Toolkit, URL: https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/register](https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/register)
- Seite 47:        [netlify.app, Login Komponent, URL: https://login--web-ui-toolkit.netlify.app/](https://login--web-ui-toolkit.netlify.app/)
- Seite 47:        [netlify.app, Register Komponent, URL: https://register--web-ui-toolkit.netlify.app/](https://register--web-ui-toolkit.netlify.app/)
- Seite 49:        [github.com, IP6-Web-UI-Toolkit, URL: https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/projector-pattern/login](https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/projector-pattern/login)
- Seite 52:        [github.com, IP6-Web-UI-Toolkit, URL: https://github.com/fabianhaef/ip6-web-ui-toolkit/blob/projector-pattern/observable/observable.js](https://github.com/fabianhaef/ip6-web-ui-toolkit/blob/projector-pattern/observable/observable.js)
- Seite 57:        [entwickler.de, Effiziente Oberflächen mit dem Projektor Pattern, URL: https://entwickler.de/java/effiziente-oberflaechen-mit-dem-projektor-pattern/](https://entwickler.de/java/effiziente-oberflaechen-mit-dem-projektor-pattern/)
- Seite 60:        [github.com, IP6-Web-UI-Toolkit, URL: https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/projector-pattern/form/mainProjector](https://github.com/fabianhaef/ip6-web-ui-toolkit/tree/projector-pattern/form/mainProjector)
- Seite 61:        [github.com, IP6-Web-UI-Toolkit, URL: https://github.com/fabianhaef/ip6-web-ui-toolkit/blob/projector-pattern/test/test.js](https://github.com/fabianhaef/ip6-web-ui-toolkit/blob/projector-pattern/test/test.js)
- Seite 65:        [github.com, IP6-Web-UI-Toolkit, URL: https://github.com/fabianhaef/ip6-web-ui-toolkit/blob/projector-pattern/login/test/loginTest.js](https://github.com/fabianhaef/ip6-web-ui-toolkit/blob/projector-pattern/login/test/loginTest.js)

# Abbildungsverzeichnis

Abbildung 1: Kolibri Logo .....	6
Abbildung 2: Primärfarbe Light- und Darkmode.....	8
Abbildung 3: Graustufen im Lightmode.....	8
Abbildung 4: Graustufen im Darkmode.....	8
Abbildung 5: Hilfsfarben im Lightmode .....	9
Abbildung 6: Hilfsfarben im Darkmode .....	9
Abbildung 7: Umfrageergebnisse aus Quelle [8].....	10
Abbildung 8: Logos von bekannten Finanzinstituten [9].....	11
Abbildung 9: Credit Suisse Logo [10] .....	11
Abbildung 10: Vergleich Farbfehlsichtigkeit der Primärfarbe im Lightmode.....	12
Abbildung 11: Vergleich Farbfehlsichtigkeit der Hilfsfarbe «Success» im Lightmode .....	12
Abbildung 12: Vergleich Farbfehlsichtigkeit der Hilfsfarbe "Warning" im Lightmode .....	12
Abbildung 13: Vergleich der Farbfehlsichtigkeit der Hilfsfarbe "Error" im Lightmode.....	12
Abbildung 14: Vergleich Farbfehlsichtigkeit der Primärfarbe im Darkmode .....	13
Abbildung 15: Vergleich Farbfehlsichtigkeit der Hilfsfarbe "Success" im Darkmode .....	13
Abbildung 16: Vergleich Farbfehlsichtigkeit der Hilfsfarbe "Warning" im Darkmode .....	13
Abbildung 17: Vergleich Farbfehlsichtigkeit der Hilfsfarbe "Error" im Darkmode.....	13
Abbildung 18: Stark Auswertung Primärfarbe Lightmode.....	14
Abbildung 19: Stark Auswertung Primärfarbe Darkmode.....	14
Abbildung 20: Auswertung Stark- Plug-In Darkmode .....	15
Abbildung 21: Auswertung Stark- Plug-In Lightmode.....	15
Abbildung 22: Zustände Primarbuttons .....	15
Abbildung 23: Login Komponente .....	16
Abbildung 24: Register Komponente .....	16
Abbildung 25: Register Komponente falsche E-Mail Eingabe.....	16
Abbildung 26: Beispielanwendung der Hilfsfarbe "Success" .....	17
Abbildung 27: Beispielanwendung der Hilfsfarbe "Error" .....	17
Abbildung 28: Inputfelder Skizzen.....	18
Abbildung 29: erster Login Prototyp .....	18
Abbildung 30: erster Register Prototyp .....	18
Abbildung 31: Usability Test Altersklassen .....	19
Abbildung 32: Usability Test IT-Affinität.....	19
Abbildung 33: Usability Test Passwort Manager .....	20
Abbildung 34: Usability Test Einloggen .....	20
Abbildung 35: Usability Test Internetstunden.....	20
Abbildung 36: Usability Test Vergleich Varianten Registrierung.....	23
Abbildung 37: Usability Test Vergleich Varianten Login.....	25
Abbildung 38: Login Komponente .....	27
Abbildung 39: Login E-Mail-Input initialer Zustand.....	27
Abbildung 40: Login E-Mail-Input aktiv Zustand.....	27
Abbildung 41: Login E-Mail-Input Error Zustand .....	28
Abbildung 42: Login E-Mail-Input Success Zustand .....	28
Abbildung 43: Login E-Mail-Input Hover Zustand.....	28
Abbildung 44: Login Password-Input initialer Zustand.....	28
Abbildung 45: Login Password-Input Hover Zustand.....	28

Abbildung 46: Login Password-Input aktiv Zustand.....	28
Abbildung 47: Login Password-Input Typed Zustand.....	29
Abbildung 48: Login Show-Password-Button Hide Zustand.....	29
Abbildung 49: Login Show-Password-Button Show Zustand.....	29
Abbildung 50: Login Show-Password-Button Pressed Zustand.....	29
Abbildung 51: Login Submit-Button Disabled Zustand.....	30
Abbildung 52: Login Submit-Button Enabled Zustand.....	30
Abbildung 53: Login Submit-Button Pressed Button.....	30
Abbildung 54: Register Komponente.....	31
Abbildung 55: Register Passwortstärke Bars initialer Zustand.....	32
Abbildung 56: Register Passwortstärke Bars Error Zustand.....	32
Abbildung 57: Register Passwortstärke Bars Warning Zustand.....	32
Abbildung 58: Register Passwortstärke Bars Success Zustand.....	32
Abbildung 59: Register Passwortstärke Kriterien initialer Zustand.....	32
Abbildung 60: Register Passwortstärke Kriterien Success Zustand.....	32
Abbildung 61: Register Passwortstärke Kriterien Error Zustand.....	33
Abbildung 62: Register Confirm-Password-Input No Match Zustand.....	33
Abbildung 63: Register Confirm-Password-Input On-Your-Way Zustand.....	33
Abbildung 64: Register Confirm-Password-Input Match Zustand.....	33
Abbildung 65: Kontrollschema zwischen Framework und Library.....	36
Abbildung 66: Verwendung von CDN in der HTML-Datei.....	38
Abbildung 67: Interface der Beispielapplikation Counter implementiert mit Webkomponente.....	42
Abbildung 68: Index.html Datei der Webkomponente.....	43
Abbildung 69: Einsicht der Webkomponente im Entwicklertool innerhalb des Browsers.....	43
Abbildung 70: HTML Template in script.js.....	44
Abbildung 71: Custom Element in script.js.....	45
Abbildung 72: CSS-Überschreibung des h4 Elements.....	46
Abbildung 73: MVC Entwurfsmuster.....	49
Abbildung 74: UI der Login Komponente realisiert mit dem Projektor-Pattern.....	49
Abbildung 75: login.js Datei des Projektor-Patterns.....	50
Abbildung 76: Eingliederung des Models.....	51
Abbildung 77: SetValidator Funktion.....	53
Abbildung 78: Hauptprojektor.....	54
Abbildung 79: Einfacher Subprojektor, Titel der Komponente.....	55
Abbildung 80: Komplexer Subprojektor, Submit Button.....	55
Abbildung 81: Setup der Benachrichtigung der E-Mail Validation.....	56
Abbildung 82: Setup der Login Benachrichtigung.....	56
Abbildung 83: View.....	57
Abbildung 84: Attribute Configs.....	58
Abbildung 85: Übergabe der Konfigurationen über die View.....	58
Abbildung 86: UI der ersten Formular Variante.....	59
Abbildung 87: Eingabevorschlage beim Fav. Color Input Feld.....	59
Abbildung 88: UI der zweiten Formular Variante.....	60
Abbildung 89: Import beider Form Varianten.....	60
Abbildung 90: Implementation der Suite.....	62
Abbildung 91: Beispieltests mit der test-Funktion.....	62
Abbildung 92: Test Suite Ausgabe mit test-Funktion.....	63
Abbildung 93: Test Suite Ausgabe mit add-Funktion.....	63
Abbildung 94: Ein fehlschlagender Test mit der test-Funktion.....	63

Abbildung 95: Ein fehlschlagender Test mit der add-Funktion.....	63
Abbildung 96: Implementation der Asserts .....	64
Abbildung 97: Implementation der <i>Report</i> und <i>Write</i> Funktionen .....	65
Abbildung 98: Test Setup .....	66
Abbildung 99: Implementation eines Tests der Login Komponente mit der Verwendung des Models .....	66
Abbildung 100: Implementation eines Tests der Login Komponente mit der Verwendung des Models und des Container Elements.....	67
Abbildung 101: Implementation der FireEvent Funktion.....	67
Abbildung 102: HTML-Datei für Tests.....	68



# Quellenverzeichnis

- [1] wikipedia.org, Kolibris, URL: <https://de.wikipedia.org/wiki/Kolibris>, abgerufen am 10.08.2021
- [2] usabilitygeek.com, Colors in UI Design: A Guide for Creating the Perfect UI, URL: <https://usabilitygeek.com/colors-in-ui-design-a-guide-for-creating-the-perfect-ui/#:~:text=Any%20UI%20Design%20guidelines%20are,for%20a%20brand%20or%20product.&text=You%20can%20employ%20a%20UX,always%20supports%20better%20information%20readability>, abgerufen am 12.08.2021
- [3] wikipedia.org, Farbenfehlsichtigkeit, URL: <https://de.wikipedia.org/wiki/Farbenfehlsichtigkeit>, abgerufen am 10.08.2021
- [4] Flexikon.doccheck.com, Fehlsichtigkeit, URL: <https://flexikon.doccheck.com/de/Fehlsichtigkeit>, abgerufen am 10.08.2021
- [5] prototypr.io, The Greys in UX Design, URL: <https://blog.prototypr.io/the-greys-in-ux-design-3a11ef9077ad>, abgerufen am 10.08.2021
- [6] prototypr.io, Using gray shade and tint in ui design, URL: <https://blog.prototypr.io/using-gray-shade-and-tint-in-ui-design-589d0e638dfd>, abgerufen am 12.08.2021
- [7] wikipedia.org, Blau, URL: <https://de.wikipedia.org/wiki/Blau>, abgerufen am 10.08.2021
- [8] joehallock.com, Colour Assignment, Undergraduate Thesis, University of Washington, URL: [http://www.joehallock.com/?page\\_id=1281](http://www.joehallock.com/?page_id=1281), abgerufen am 11.08.2021
- [9] bidcreative.com, The Logo Blues, URL: <https://www.bidcreative.com/the-logo-blues/>, abgerufen am 10.08.2021
- [10] wikipedia.org, Credit Suisse Logo, URL: [https://de.wikipedia.org/wiki/Datei:Credit\\_Suisse\\_Logo.svg](https://de.wikipedia.org/wiki/Datei:Credit_Suisse_Logo.svg), abgerufen am 10.08.2021
- [11] figma.com, Color Blind Plugin, URL: <https://www.figma.com/community/plugin/733343906244951586>, abgerufen am 10.08.2021
- [12] medium.com, designing for colour blindness, URL: <https://medium.com/@sidgtl/designing-for-colour-blindness-b74a9d012ef2>, abgerufen am 10.08.2021
- [13] figma.com, Stark Plugin, URL: <https://www.figma.com/community/plugin/732603254453395948/Stark>, abgerufen am 10.08.2021
- [14] donau-uni.ac.at, Farbgestaltung und Ihre kulturelle Bedeutung, URL: [https://imbstudent.donau-uni.ac.at/mmd\\_education12\\_2/](https://imbstudent.donau-uni.ac.at/mmd_education12_2/), abgerufen am 10.08.2021
- [15] betterprogramming.pub, Libraries vs Frameworks - What's the difference, URL: <https://betterprogramming.pub/libraries-vs-frameworks-whats-the-difference-5f28c53dcffe>, abgerufen am 14.08.2021
- [16] freecodecamp.org, What is npm? A node package manager tutorial for beginners, URL: <https://www.freecodecamp.org/news/what-is-npm-a-node-package-manager-tutorial-for-beginners/>, abgerufen am 14.08.2021
- [17] medium.com, CDN explained. Why, when and how to use it for your website, URL: <https://medium.com/pixboost/cdn-explained-why-when-and-how-to-use-it-for-your-website-7d360a93cc04>, abgerufen am 04.08.2021
- [18] jsdelivr.com, Open-Source CDN, URL: <https://www.jsdelivr.com/>, abgerufen am 14.08.2021
- [19] docs.github.com, managing releases in a repository, URL: <https://docs.github.com/en/github/administering-a-repository/releasing-projects-on-github/managing-releases-in-a-repository>, abgerufen am 14.08.2021
- [20] youtube.com, Traversy Media, Web Components Crash Course, URL: <https://www.youtube.com/watch?v=PCWaFLy3VUo>, abgerufen am 14.08.2021
- [21] github.com, Web-Engineering FHNW, webcl-fs-2, URL: <https://github.com/WebEngineering-FHNW/webcl-fs21-2/tree/main/week6/todo>, abgerufen am 14.08.2021

- [22] github.com, Web-Engineering FHNW, webcl-hs20, test, URL: <https://github.com/WebEngineering-FHNW/webcl-hs20/tree/master/week14/test>, abgerufen am 14.08.2021
- [23] exploredatabase.com, What is derived attribute in dbms and how to create derived attributes, URL: <https://www.exploredatabase.com/2017/02/what-is-derived-attribute-in-dbms-and-how-to-create-derived-attributes.html>, abgerufen 19.08.2021
- [24] developer.mozilla.org, Div, URL: <https://developer.mozilla.org/de/docs/Web/HTML/Element/div>, abgerufen 14.08.2021
- [25] ibm.com, Django-Explained, URL: <https://www.ibm.com/cloud/learn/django-explained>, abgerufen am 14.08.2021
- [26] wikipedia.org, Document Object Model, URL: [https://de.wikipedia.org/wiki/Document\\_Object\\_Model](https://de.wikipedia.org/wiki/Document_Object_Model), abgerufen 14.08.2021
- [27] netzstrategie.com, Entwicklertools im Browser, URL: <https://netzstrategie.com/blog/webwissen-entwicklertools-im-browser>, abgerufen am 14.08.2021
  
- [28] wikipedia.org, Kommandozeile, URL: <https://de.wikipedia.org/wiki/Kommandozeile>, abgerufen 14.08.2021
- [29] computerweekly.com, Model View Controller MVC, URL: <https://www.computerweekly.com/de/definition/Model-View-Controller-MVC>, abgerufen am 14.08.2021
- [30] tornau.name, Das Observer Pattern / Beobachter-Muster, URL: <http://www.tornau.name/2014/02/das-observer-pattern-beobachter-muster>, abgerufen am 14.08.2021
- [31] softselect.de, Plug-In, URL: <http://www.softselect.de/business-software-glossar/plugin>, abgerufen am 14.08.2021
- [32] wikipedia.org, Python (Programmiersprache), URL: [https://de.wikipedia.org/wiki/Python\\_\(Programmiersprache\)](https://de.wikipedia.org/wiki/Python_(Programmiersprache)), abgerufen am 14.08.2021