

ENTWICKLUNG EINER FORMULARANWENDUNG MIT KOMPATIBILITÄTSVALIDIERUNG DER EINFACH- UND MEHRFACHAUSWAHL-EINGABEFELDER

Vorgelegt von:

Alexander Johr

Meine Adresse

Erstprüfer: Prof. Jürgen Singer Ph.D.
Zweitprüfer: Prof. Daniel Ackermann
Datum: 02.11.2020

Teil I

Implementierung

0.1 Schritt 4

Im Folgenden werden der Validierung die Bedingungen hinzugefügt, welche die Auswahloptionen untereinander haben.

0.1.1 Hinzufügen der Bedingungen zu den Auswahloptionen

Es gibt einfache Bedingungen wie beispielsweise die der Zielfläche „AL“. Dessen Auswahl kann nur dann erfolgen, wenn nicht die Kategorie „Anbau Zwischenfrucht/Untersaat“ ausgewählt ist (Listing 1).

```
79 static final al = ZielflaecheChoice("al", "AL",  
80   condition: (choices) => !choices.contains(KategorieChoice.zf_us));
```

Listing 1: XXXXX, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-4/conditional_form/lib/choices/choices.dart](#)

Doch es tauchen auch komplexe Bedingungen auf, wie etwa die Abhängigkeit der Zielfläche „Wald/Forst“ (Listing 2). Um sie auszuwählen, muss die Förderklasse eine von 3 Werten beinhalten: „Erschwernisausgleich“ (Z. 97), „Agrarumwelt-(und Klima)Maßnahme: nur Vertragsnaturschutz“ (Z. 98) oder „Agrarumwelt-(und Klima)Maßnahmen, tw. auch mit Tierwohlaspekten , aber OHNE Vertragsnaturschutz“ (Z. 99).

Gleichzeitig darf für die „Kategorie“ weder „Anbau Zwischenfrucht/Untersaat“ (Z. 100) noch „Förderung bestimmter Rassen / Sorten / Kulturen“ (Z. 101) gewählt sein.

Äußerst wichtig ist hier die Auswahl der richtigen logischen Operatoren. Innerhalb des gleichen Typs – wie etwa der „Förderklasse“ – muss das logische Oder `||` verwendet werden (Z. 97, 98, 100). Das logische Und würde hier keinen Sinn ergeben, da es unmöglich ist, in einem Einfachauswahlfeld gleichzeitig zwei Optionen ausgewählt zu haben. Um Bedingungen unterschiedlichen Typs miteinander zu verknüpfen, ist dagegen das logische und `&&` zu benutzen (Z. 99), denn die Bedingungen der „Förderklasse“ und der „Kategorie“ müssen gleichzeitig erfüllt sein. Hier ist wiederum das Nutzen des logischen Oders nicht angemessen, denn es wäre nicht ausreichend, wenn nur die Bedingungen eines der beiden Typen erfüllt wäre. Wäre also beispielsweise für die „Förderklasse“ die Option „Erschwernisausgleich“ gewählt, so wäre es völlig unerheblich, welche Auswahl für die „Kategorie“ selektiert wurde. Die Bedingung wäre trotzdem erfüllt, auch wenn für die „Kategorie“ die nicht erlaubte Option „Anbau Zwischenfrucht/Untersaat“ gewählt ist.

```
95 static final wald = ZielflaecheChoice("wald", "Wald/Forst",  
96   condition: (choices) =>  
97     (choices.contains(FoerderklasseChoice.ea) ||  
98       choices.contains(FoerderklasseChoice.aukm_nur_vns) ||  
99       choices.contains(FoerderklasseChoice.aukm_ohne_vns)) &&  
100     (!choices.contains(KategorieChoice.zf_us) ||  
101       !choices.contains(KategorieChoice.bes_kult_rass)));
```

Listing 2: XXXXX, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-4/conditional_form/lib/choices/choices.dart](#)

Für die Liste aller hinzugefügten Bedingungen siehe Anhang D auf den Seiten 20 bis 22.

Bei der Bedingungen handelt sich um eine Funktion, die einen Wahrheitswert `bool` zurückgibt und als Parameter die Menge aller bisher ausgewählten Auswahloptionen `Set<Choice>` übergeben bekommt. Die Signatur dieser Funktion wird als Typdefinition mit dem Namen

`Condition` deklariert (Listing. 3, Z. 3). Über diese Typdefinition kann sie als Instanzvariable in der Klasse `Choice` deklariert werden (Z. 8). Der Konstruktor erhält einen weiteren Parameter für die Bedingung (Z. 12).

Er ist optional, da es Auswahloption gibt, die keine Bedingung haben. Deshalb wird mit der Notation `Condition?` erreicht, dass die Bedingung auch ausgelassen werden kann und in diesem Fall `null` ist. Sollte das der Fall sein, so soll eine Standardfunktion verwendet werden. Diese Standardfunktion ist `_conditionIsAlwaysMet` (Z. 15). Unerheblich davon welche Auswahloptionen in Vergangenheit gewählt wurden, gibt diese Funktion immer `true` zurück. Denn eine Auswahloption, die keine Bedingung hat, ist immer auswählbar. Sollte die übergebene Bedingungen ausgelassen worden und damit `null` sein, so wählt die „If-null Expression“ den Ausdruck rechts von dem `??` und damit die Standardfunktion `_conditionIsAlwaysMet` aus, welche der Instanzvariablen `condition` zugewiesen wird (Z. 13). Ansonsten speichert der Konstruktor die übergebene Funktion. Aus diesem Grund ist es nicht möglich, dass die `condition` in der Instanzvariablen nur sein kann, weshalb sie ohne den Suffix `?` als Variable ohne zu null Zulässigkeit deklariert werden kann. Da der Ausdruck rechts von dem `??` nicht `null` sein kann, so kann auch der gesamte Ausdruck der vorliegenden „If-null Expression“ nicht `null` sein. Damit ist es möglich, die Instanzvariable `condition` ohne den Suffix `?` als Variable ohne Null-Zulässigkeit zu deklarieren. Die Instanzmethode `conditionMatches` ruft die übergebender Funktion für die Bedingung über die Methode `call` auf (Z. 10). Das erlaubt den Ausdruck durch vereinfacht darzustellen. Der Ausdruck `wald.condition(priorChoices)` kann dadurch durch die explizitere Schreibweise `wald.conditionMatches(priorChoices)` ersetzt werden.

```
3 typedef Condition = bool Function(Set<Choice> choices);
4
5 class Choice {
6   final String description;
7   final String abbreviation;
8   final Condition condition;
9
10  bool conditionMatches(Set<Choice> choices) => condition.call(choices);
11
12  const Choice(this.abbreviation, this.description, {Condition? condition})
13    : condition = condition ?? _conditionIsAlwaysMet;
14
15  static bool _conditionIsAlwaysMet(Set<Choice> choices) => true;
16 }
```

Listing 3: XXXXX, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-4/conditional_form/lib/choices/base/choice.dart](#)

0.1.2 Hinzufügen der Momentaufnahme aller ausgewählten Optionen im gesamten Formular

Die Menge der bisherigen Ausfülloptionen setzt sich aus den aktuellen Inhalten der Auswahlfelder zusammen. Sie ist also die Momentaufnahme aller Werte, die jeweils über die Getter-Methode `value` von allen `BehaviorSubject`-Objekten im `ViewModel` abgerufen werden kann. Doch genau diese Momentaufnahme muss immer dann neu erstellt werden, wenn sich auch nur ein Auswahlfeld ändert. Genau darum kümmert sich das `BehaviorSubject priorChoices` im `ViewModel` (Listing 4).

Es wird mit dem Typparameter `Set<Choice>` deklariert (Z. 20) und mit einer Momentaufnahme initialisiert: einer leeren Menge) (Z. 21). Im Konstruktor des `ViewModels` wird dann auf Änderung aller `BehaviorSubject`-Objekte im `ViewModel` gehorcht. Dies wird durch die Funktion `combineLatest` des Pakets `rx.dart` ermöglicht 24. Sie erlaubt die Übergabe ei-

```

20 BehaviorSubject<Set<Choice>> priorChoices =
21     BehaviorSubject<Set<Choice>>.seeded({});
22
23 MassnahmenFormViewModel() {
24     Stream<Set<Choice>> choicesStream = Rx.combineLatest([
25         foerderklasse,
26         kategorie,
27         zielflaeche,
28         zieleinheit,
29         hauptzielsetzungLand,
30     ], (_) {
31         return {
32             if (foerderklasse.value != null) foerderklasse.value!,
33             if (kategorie.value != null) kategorie.value!,
34             if (zielflaeche.value != null) zielflaeche.value!,
35             if (zieleinheit.value != null) zieleinheit.value!,
36             if (hauptzielsetzungLand.value != null) hauptzielsetzungLand.value!,
37         };
38     });
39
40     choicesStream.listen((event) => priorChoices.add(event));
41 }

```

Listing 4: XXXX, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-4/conditional_form/lib/screens/massnahmen_detail/massnahmen_form_view_model.dart](#)

ner Kollektion von Streams. In diesem Fall alle `BehaviorSubject`-Objekte des ViewModels (Z. 25-29). Wenn auch nur einer dieser Streams ein neues Ereignis sendet, so emittiert auch der kombinierte Stream ein neues Ereignis. Dem zweiten Parameter der Funktion `combineLatest` kann als Argument eine Funktion übergeben werden, die das zu emittierende Ereignis konstruiert (Z. 30-37). Der erste Parameter dieser Funktion enthält alle letzten Ereignisse der übergebenen Streams. Doch der vorliegende Aufruf hat keine Verwendung für den Parameter. Statt eines Variablennamens wird hier ein Unterstrich `_` verwendet (Z. 30).

In Sprachen wie etwa „*JavaScript*“ und „*Python*“ ist dies gängige Praxis für die Benennung von Parametern, die nicht genutzt werden. In Kotlin und Dart wurde diese Praxis zur Konvention gemacht^{1,2}. Die anonyme Funktion gibt eine Menge zurück, in welcher alle Werte der `BehaviorSubject`-Objekte integriert werden (Z. 31-37). Das „*Collection if*“ Statement schließt dabei jeweils den Wert `null` aus (Z. 32-36). Somit taucht niemals der Wert `null` in der Menge auf und damit kann die Menge mit dem Typparameter `Choice` ohne Null-Zulässigkeit deklariert werden. Sollte ein Auswahlfeld nicht gewählt und damit der Wert des `BehaviorSubject` `null` sein, so taucht diese Option einfach nicht in der Menge auf. Sind alle Auswahlfelder nicht belegt und damit `null`, so ist die Menge leer. Doch der kombinierte Stream `choicesStream` liefert immer nur die neuen Ereignisse und speichert nicht den zuletzt übermittelten Wert. Deshalb wird das `BehaviorSubject` `priorChoices` verwendet. Die Methode `listen` horcht auf Änderungen des `choicesStream`-Objekts und fügt das übertragene Ereignis immer `priorChoices` hinzu. Damit existiert immer ein Wert für die Momentaufnahme der aktuell ausgewählten Auswahloptionen. Sie ist ursprünglich die leere Menge `{}` und nachfolgend immer das zuletzt übermittelte Ereignis des `choicesStream`.

¹Google LLC, *Dart - Effective Dart - Style - PREFER using _, __, etc. for unused callback parameters.*

²JetBrains s.r.o., *Kotlin - High-order functions and lambdas - Underscore for unused variables.*

0.1.3 Reagieren der Selektionskarte auf die ausgewählten Optionen

Dadurch, dass `priorChoices` nun im ViewModel verfügbar ist, kann es im Eingabeformular bei der Konstruktion der `SelectionCard` als Argument übergeben werden (Listing. 5, Z. 143).

```
140 builder: (field) => SelectionCard<ChoiceType>(  
141     title: allChoices.name,  
142     allChoices: allChoices,  
143     priorChoices: vm.priorChoices,
```

Listing 5: Die Ausgabe der Formularfelder, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-4/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart](#)

Die Klasse `SelectionCard` deklariert die `priorChoices` als Instanzvariable (Listing. 6, Z. 19) und initialisiert sie direkt bei der Übergabe im Konstruktor, ohne sie zu modifizieren (Z. 28).

```
15 class SelectionCard<ChoiceType extends Choice> extends StatelessWidget {  
16     final String title;  
17     final BehaviorSubject<BuiltSet<ChoiceType>> selectionViewModel;  
18     final Choices<ChoiceType> allChoices;  
19     final BehaviorSubject<Set<Choice>> priorChoices;  
20     final OnSelect<ChoiceType> onSelect;  
21     final OnDeselect<ChoiceType> onDeselect;  
22     final String? errorText;  
23  
24     SelectionCard(  
25         {required this.title,  
26         required Iterable<ChoiceType> initialValue,  
27         required this.allChoices,  
28         required this.priorChoices,  
29         required this.onSelect,  
30         required this.onDeselect,  
31         this.errorText,  
32         Key? key})
```

Listing 6: XXXX, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-4/conditional_form/lib/widgets/selection_card.dart](#)

Dadurch, dass das `BehaviorSubject` ein Stream ist, kann die Selektionskarte auf Änderungen reagieren, die sich an `priorChoices` vollziehen, obwohl diese Änderungen außerhalb der Klasse geschehen. Würde stattdessen eine Liste der bisherigen Auswahloptionen übergeben werden, so wäre diese eine Kopie. Diese Kopie hätte den Zustand einer Momentaufnahme aller bisherigen Auswahloptionen zum Zeitpunkt der Konstruktion des `SelectionCard`-Elementes. Alle Änderungen, die nach diesem Zeitpunkt an den Auswahloptionen geschehen sind, würden sich nicht darin widerspiegeln. Eine Selektionskarte würde daher auch keinen Fehler anzeigen, wenn ihre ausgewählten Optionen durch Änderungen von außen invalide werden würden. Der Grund dafür ist, dass sie noch eine alte Kopie der bisherigen Auswahloptionen verwendet.

Eine andere Möglichkeit wäre, eine Setter-Methode zu implementieren, die den Wert der bisherigen Auswahloptionen neu setzt. Doch das Programm verwaltet keine Referenzen auf alle gebauten Selektionskarten. Somit kann auch nicht über eine Referenz eine Setter-Methode aufgerufen werden, denn eine solche Referenz existiert nicht. Die übliche Vorgehensweise wäre in Flutter das gesamte Widget neu zu zeichnen. Bei Einsatz eines „*Stateful-Widgets*“ und Zustandsänderungen über die `setState`-Methode würde dies das Neuzeichnen des gesamten Formulars bedeuten.

Performante ist es dagegen, wenn nur die Inhalte der Selektionskarten ausgetauscht werden. Anstatt ausschließlich auf die Änderungen der eigenen Auswahloptionen zu reagieren, horcht der `StreamBuilder` nun auf den Stream `priorChoices` (Listing. 7, Z. 52) und damit auf die Änderungen aller Auswahlfelder. Vor der Konstruktion der Karte wird nun überprüft, ob einer der ausgewählten Auswahloptionen in `selectedChoices` eine invalide Auswahl enthält (Z. 55-56). Das kann über die Funktion `any` herausgefunden werden, indem für jede ausgewählte Option die Methode `conditionMatches` mit der Menge aller ausgewählten Optionen im gesamten Formular aufgerufen wird (Z. 56). Die rote Farbe der Selektionskarte wurde bereits bei der Validierung im letzten Schritt verwendet, wenn der dem Konstruktor ein `errorText` übergeben wurde. Nun wird diese Bedingung erweitert. Sollte es auch nur eine falsche Selektion geben oder aber der `errorText` gesetzt sein, so ist die Karte rot. Anderenfalls wird dem Parameter `tileColor` `null` übergeben (Z. 70). `null` bedeutet, dass keine Farbe übergeben und damit die Standardfarbe verwendet wird.

```
51 return StreamBuilder(  
52   stream: priorChoices,  
53   builder: (context, snapshot) {  
54     final selectedChoices = selectionViewModel.value;  
55     final bool wrongSelection = selectedChoices  
56       .any((c) => !c.conditionMatches(priorChoices.value));  
57  
58     return Card(  
59       child: Column(  
60         crossAxisAlignment: CrossAxisAlignment.start,  
61         children: [  
62           ListTile(  
63             focusNode: focusNode,  
64             title: Text(title),  
65             subtitle: Text(  
66               selectedChoices.map((c) => c.description).join(", "),  
67             trailing: const Icon(Icons.edit),  
68             onTap: navigateToSelectionScreen,  
69             tileColor:  
70               wrongSelection || errorText != null ? Colors.red : null,  
69
```

Listing 7: XXXX, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-4/conditional_form/lib/widgets/selection_card.dart](#)

0.1.4 Reagieren des Auswahlbildschirms auf die ausgewählten Optionen

Der Auswahlbildschirm wird im Folgenden um zwei weitere Funktionalitäten erweitert (Listing 8). Sollten durch neue Selektionen im Formular bereits selektierte Optionen im Auswahlbildschirm nun invalide sein, so werden diese rot gefärbt. Weiterhin erscheinen invalide Optionen, die nicht ausgewählt sind, am Ende der Liste ohne Checkbox zum Auswählen. Außerdem erhält die Option ein Kreuz-Icon als Indikator dafür, dass sie nicht ausgewählt werden kann.

Zu diesem Zweck konstruiert der `StreamBuilder` vor der Rückgabe des `ListView`s zwei Mengen. Die Menge `selectedAndSelectableChoices` (Z. 95) beinhaltet alle Auswahloptionen, die entweder selektiert oder selektierbar sind. Dies beinhaltet auch Optionen, die invalide und trotzdem selektiert sind. Die zweite Menge `unselectableChoices` (Z. 96) dagegen beinhaltet alle Optionen, die invalide sind, und nicht selektiert sind.

Eine Schleife iteriert über alle verfügbaren Optionen, welche der Auswahl Bildschirm anzeigt (Z. 90-105). Sollte die Option in den selektierten Optionen enthalten (Z. 99), oder aber mit den Selektionen aller anderen Auswahlfelder kompatibel sein, so wird sie der Menge `selectedAndSelectableChoices` hinzugefügt (Z. 101). In jedem anderen Fall wird die Option Teil der Menge `unselectableChoices` (Z. 103).

Für die Konstruktion der `CheckboxListTile`-Elemente wurde zuvor die Menge aller Auswahloptionen verwendet. Nun wird stattdessen nur die Menge der selektierbaren und selektierten Auswahloptionen genutzt (Z. 108). Neben dem Vergleich, ob die Option selektiert ist (Z. 109), erfolgt nur noch ein weiterer Vergleich, ob die Option inkompatibel mit den ausgewählten Optionen aller anderen Auswahlfelder ist (Z. 111). Das Ergebnis des Vergleiches wird in der lokalen Variable `selectedButDoesNotMatch` gespeichert.

Sollte diese Variable `true` sein, so erscheint das `CheckboxListTile`-Element mit einem rot eingefärbten im Hintergrund. Der Benutzer hat über die Checkbox dann die Möglichkeit, diese Auswahl zu deselektieren. Da das hinterlegte `ViewModel` durch diese Deselektion direkt aktualisiert wird (Z. 122-123), so baut der `StreamBuilder` auch den `ListView` neu. Die deselektierte Option wird dann Teil von der Menge `unselectableChoices` (Z. 103) sein. So erscheint sie dann – ganz genau wie alle anderen unselektierbaren Auswahloptionen – ohne roten Hintergrund aber auch ohne anklickbare Checkbox am Ende der Liste (Z. 134-142).

Solche unselektierbaren Optionen werden schlicht als `ListTile`-Element statt als `CheckCoxListTile` gezeichnet (Z. 135-139). Damit fehlt ihnen die Checkbox zum Selektieren. Über den Parameter `leading` kann jedoch anstelle der Checkbox ein beliebiges Widget – in diesem Fall ein Icon – eingefügt werden. `Icons.close` zeichnet ein Kreuz-Symbol, um zu signalisieren, dass diese Option nicht anwählbar ist.


```

88   title: Text(title),
89 ),
90 body: StreamBuilder(
91   stream: selectionViewModel,
92   builder: (context, snapshot) {
93     final selectedChoices = selectionViewModel.value;
94
95     Set<ChoiceType> selectedAndSelectableChoices = {};
96     Set<ChoiceType> unselectableChoices = {};
97
98     for (ChoiceType c in allChoices) {
99       if (selectedChoices.contains(c) ||
100         c.conditionMatches(priorChoices.value)) {
101         selectedAndSelectableChoices.add(c);
102       } else {
103         unselectableChoices.add(c);
104       }
105     }
106
107     return ListView(children: [
108       ...selectedAndSelectableChoices.map((ChoiceType c) {
109         bool isSelected = selectedChoices.contains(c);
110         bool selectedButDoesNotMatch =
111           !c.conditionMatches(priorChoices.value);
112
113         return CheckboxListTile(
114           key: Key(
115             "valid choice ${allChoices.name} - ${c.abbreviation}"),
116           controlAffinity: ListTileControlAffinity.leading,
117           title: Text(c.description),
118           tileColor: selectedButDoesNotMatch ? Colors.red : null,
119           value: isSelected,
120           onChanged: (selected) {
121             if (selected != null) {
122               selectionViewModel.value =
123                 selectionViewModel.value.rebuild((b) {
124                   b.replace(isSelected ? [] : [c]);
125                 });
126             if (selected) {
127               onSelect(c);
128             } else {
129               onDeselect(c);
130             }
131           }
132         );
133       }).toList(),
134       ...unselectableChoices.map((Choice c) {
135         return ListTile(
136           key: Key(
137             "invalid choice ${allChoices.name} - ${c.abbreviation}"),
138           title: Text(c.description),
139           leading: const Icon(Icons.close));
140       }).toList()
141     ]);

```

Listing 8: XXXX, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-4/conditional_form/lib/widgets/selection_card.dart](#)

0.1.5 Hinzufügen der Momentaufnahme zur Validierung

Alle bisher eingefügten Vergleiche hatten lediglich den Zweck, die invaliden Optionen einzufärben und von der Selektion durch den Benutzer auszuschließen. Doch noch sind sie nicht Teil der Validierung des Formulars. Sollte der Benutzer die aktuell eingetragene Maßnahmen im abgeschlossenen Status abspeichern wollen, so kann dies auch mit invaliden Optionen erfolgen. Um das zu verhindern, wird noch ein Vergleich zu der anonymen Funktion hinzugefügt, welche als Argument dem Parameter `validator` des `FormField` übergeben wird (Listing 9). Sollte auch nur eine der selektierten Optionen `choices` die ihr hinterlegte Bedingungen nicht erfüllen (Z. 132), so speichert die lokale Variable `atLeastOneValueInvalid` den Wert `true` ab (Z. 131).

In dem Fall gibt die Funktion die entsprechende Fehlermeldung an den Benutzer zurück (Z. 135). Somit ist es nun auch nicht mehr möglich, eine Maßnahme abzuspeichern, wenn sie invalide Auswahloptionen enthält. Erst wenn alle Auswahlfelder gefüllt sind und die gefüllten Optionen alle die jeweils hinterlegten Bedingungen erfüllen, so werden die `validator`-Funktionen `null` statt einer Fehlermeldung zurückgeben (Z. 138). Nur dann kann eine Maßnahme mit dem Status „abgeschlossen“ gespeichert werden.

```
121 return FormField(  
122   validator: (_) {  
123     Iterable<Choice> choices = {  
124       if (selectionViewModel.value != null) selectionViewModel.value!  
125     };  
126  
127     if (choices.isEmpty) {  
128       return "Feld ${allChoices.name} enthält keinen Wert!";  
129     }  
130  
131     bool atLeastOneValueInvalid =  
132       choices.any((c) => !c.conditionMatches(vm.priorChoices.value));  
133  
134     if (atLeastOneValueInvalid) {  
135       return "Wenigstens ein Wert im Feld ${allChoices.name} enthält ist fehlerhaft!";  
136     }  
137  
138     return null;  
139   },  
140   builder: (field) => SelectionCard<ChoiceType>(
```

Listing 9: Die Ausgabe der Formularfelder, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-4/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart](#)

0.2 Schritt 5

Im letzten Schritt wurde das primäre Problem der Formularanwendung gelöst: Auswahloptionen sollen nur dann anwählbar sein, wenn sie die ihr hinterlegte Bedingung erfüllen. Darüber hinaus können nur Maßnahmen gespeichert werden, deren Auswahloptionen untereinander kompatibel sind.

Durch das Lösen dieses Problems ist ein neues Problem entstanden: Alle Selektionskarten müssen bei einer Selektion neu gezeichnet werden. Bei einer geringen Anzahl von Auswahlfeldern sollte das noch keine gravierenden Auswirkungen auf das Laufzeitverhalten der Applikation haben. Doch je zahlreicher die Auswahlfelder werden, desto länger dauert die Aktualisierung der Oberfläche.

Das Problem kann folgendermaßen entschärft werden: Noch bevor das Widget `SelectionCard` den `StreamBuilder` in der `build`-Methode zurückgibt, wird ein neuer Stream namens `validityChanged` erstellt (Listing. 10, Z. 51-54).

Es handelt sich um eine sogenannte Transformation des Streams `priorChoices`, welcher die Momentaufnahme aller ausgewählten Optionen im gesamten Formular übermittelt. Immer dann, wenn der Stream `priorChoices` ein neues Ereignis sendet, geschieht für die Abwandlung dieses Streams folgendes: Die Methode `map` wandelt jedes Ereignis in ein neues Objekt um (Z. 52). Die aktuelle Momentaufnahme der Auswahloptionen im Formular wird dazu im Parameter `choices` gespeichert. Bei der Umwandlung des Ereignisses werden die ausgewählten Optionen der aktuellen Selektionskarte über `selectionViewModel.value` abgerufen. Sollte es sich beispielsweise bei der aktuellen Selektionskarte um das Auswahlfeld der „Kategorie“ handeln, so könnte der ausgewählte Wert „Düngemanagement“ sein. Für den Wert oder die Werte wird nun überprüft, ob sie mit der neuen Momentaufnahme der Selektionen im Formular kompatibel sind. Wurde also beispielsweise bei der neuen Selektion in der „Förderklasse“ nun „Ökolandbau“ ausgewählt, so würde die Option „Düngemanagement“ nun invalide werden, da sie nur mit der Förderklasse „Agrarumwelt-(und Klima)Maßnahme: nur Vertragsnaturschutz“ bzw. „Agrarumwelt-(und Klima)Maßnahmen, tw. auch mit Tierwohl-aspekten, aber OHNE Vertragsnaturschutz“ kompatibel ist. Die Methode `map` wandelt also das neue Ereignis der Momentaufnahme aller Selektionen im Formular in einen einzigen Wahrheitswert um. Ist der Wahrheitswert `true`, bedeutet dies, dass alle ausgewählten Optionen in der aktuellen Selektionskarte valide sind. Ist er dagegen `false`, so ist wenigstens eine der Auswahloption mit den restlichen Auswahloptionen der anderen Auswahlfelder im Formularen nicht kompatibel.

Der resultierende Stream wird weiter transformiert: Durch die Funktion `distinct` (Z. 54) werden nur Ereignisse gesendet, sofern sie sich von dem letzten Ereignis unterscheiden. Ein Beispiel: Für die „Kategorie“ ist „Düngemanagement“ ausgewählt. Für die „Förderklasse“ ist „Erschwernisausgleich“ im letzten Ereignis ausgewählt worden. „Düngemanagement“ ist mit „Erschwernisausgleich“ nicht kompatibel, weshalb das letzte Ereignis des durch `map` transformierten Streams `false` war. Nun wird für die „Förderklasse“ eine weitere Selektion vorgenommen: „Ökolandbau“ wird ausgewählt. Auch diese Option ist mit „Düngemanagement“ nicht kompatibel. Der durch `map` transformierten Stream wird also erneut ein Ereignis mit dem Wert `false` senden. Doch bereits das letzte Ereignis war `false`. Die Methode `distinct` verhindert, dass dieses redundante Ereignis weitergeleitet wird. Nun erfolgt noch eine weitere Selektion: Für die „Förderklasse“ wird „Agrarumwelt-(und Klima)Maßnahme: nur Vertragsnaturschutz“ selektiert. Nun ist die „Kategorie“ „Düngemanagement“ mit der neuen Selektion kompatibel. Der aus der Methode `map` resultierende Stream liefert dieses Mal den Wert `true`. Das letzte Ereignis hatte den Wert `false`. Die Werte der beiden letzten Ereignisse unterscheiden sich also, was dazu führt, dass die Methode `distinct` das veränderte Ereignis nicht filtert sondern weiterleitet.

Der Stream `validityChanged` sendet also immer genau dann Ereignisse, wenn sich etwas an der Validität der Auswahloptionen der aktuellen Selektionskarte ändert. Doch dieser Stream kann nicht für den `StreamBuilder` benutzt werden. Denn wenn sich die Auswahl in der aktuellen Selektionskarte ändert und die Validität dadurch unverändert bleibt, so erfolgt kein neues Zeichnen der Selektionskarte. Deshalb ist eine Kombination der Streams `validityChanged` und `selectionViewModel` erforderlich. Das `BehaviorSubject` `needsRepaint` soll als diese Kombination fungieren (Z. 56). Es wird mit dem Wert (Z. true) initialisiert. Es ist unerheblich, welcher Wert in dem Stream aktuell gespeichert ist. Lediglich dass ein neues Ereignis hinzugefügt wird, um die Aktualisierung der Oberfläche auszulösen, ist wesentlich. Mit der Methode `listen` wird nun sowohl auf den Stream `validityChanged` (Z. 57) als auch auf `selectionViewModel` (Z. 58) gehorcht. Jedes empfangene Ereignis wird dabei dem `BehaviorSubject` `needsRepaint` hinzugefügt.

Dadurch, dass `needsRepaint` für den `StreamBuilder` verwendet wird (Z. 61), zeichnet sich die Selektionskarte immer dann neu, wenn sich die beinhaltenden Auswahloptionen oder aber dessen Validität ändert.

```

51 final validityChanged = priorChoices
52   .map((choices) =>
53     selectionViewModel.value.any((c) => !c.conditionMatches(choices)))
54   .distinct();
55
56 final needsRepaint = BehaviorSubject.seeded(true);
57 validityChanged.listen((value) => needsRepaint.add(true));
58 selectionViewModel.listen((value) => needsRepaint.add(true));
59
60 return StreamBuilder(
61   stream: needsRepaint,
62   builder: (context, snapshot) {
63     final selectedChoices = selectionViewModel.value;
64     final bool wrongSelection = selectedChoices
65       .any((c) => !c.conditionMatches(priorChoices.value));
66
67     return Card(
68       child: Column(
69         crossAxisAlignment: CrossAxisAlignment.start,
70         children: [
71           ListTile(

```

Listing 10: XXXX, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-5/conditional_form/lib/widgets/selection_card.dart](#)

Dieses Verhalten kann auch bei Ausführung der Applikation im Debugmodus in Android Studio beobachtet werden. Der „Flutter Performance“-Tab gibt eine Übersicht über die Anzahl der im letzten Frame neu gezeichneten Widgets (Abb. 1).

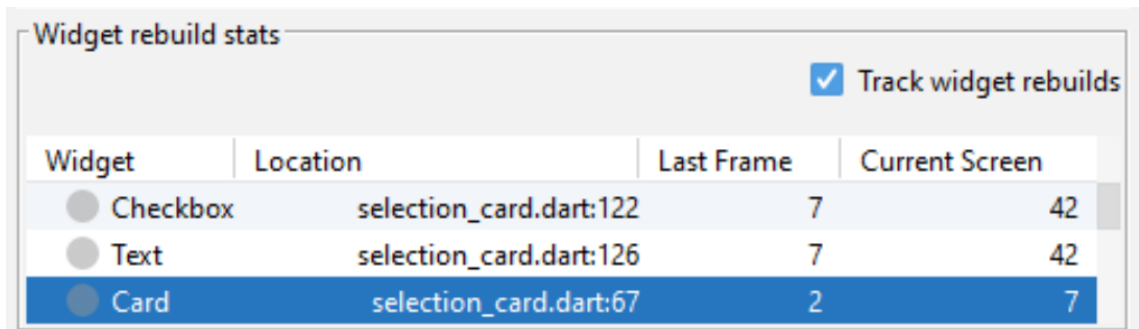
Widget rebuild stats			
			<input checked="" type="checkbox"/> Track widget rebuilds
Widget	Location	Last Frame	Current Screen
Checkbox	selection_card.dart:122	7	56
Text	selection_card.dart:126	7	56
Card	selection_card.dart:67	1	6

Abbildung 1: XXXXX, Quelle: Eigene Abbildung

Angenommen für die „Förderklasse“ ist „Agrarumwelt-(und Klima)Maßnahme: nur Ver-

tragsnaturschutz“ und für die „Kategorie“ ist „Düngemanagement“ ausgewählt. Wenn nun für die Förderklasse die Option „Agrarumwelt-(und Klima)Maßnahmen, tw. auch mit Tierwohlaspekten, aber OHNE Vertragsnaturschutz“ selektiert wird, so ist im „Flutter Performance“-Tab zu beobachten, dass das Widget Card nur einmal neu gezeichnet wurde.

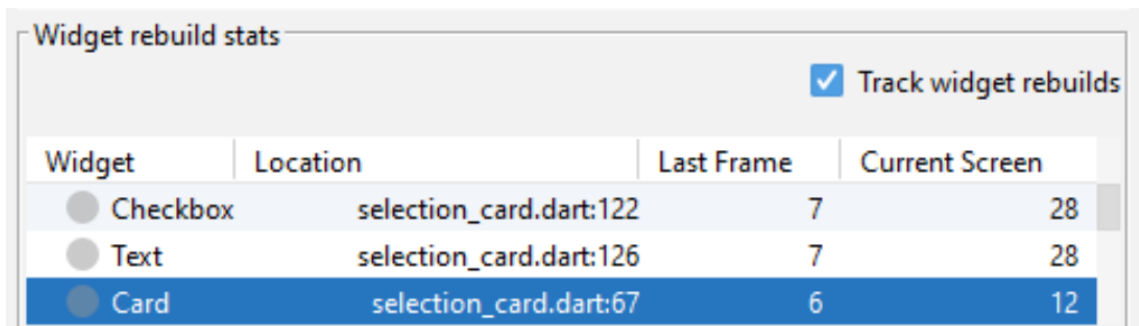
Das ergibt Sinn, denn es hat sich nichts an der Validität eines anderen Auswahlfeld geändert. Lediglich die Selektionskarte für die Förderklasse muss neu gezeichnet werden, da sich seine Selektion angepasst hat. Wird nun aber die „Förderklasse“ „Ökolandbau“ ausgewählt, so ist zu beobachten, dass das Card Widget zweimal gebaut wurde: Einmal für die Selektionskarte der „Förderklasse“, da sich dessen ViewModel änderte; Ein weiteres Mal für die Selektionskarte der „Kategorie“, da die Auswahl „Düngemanagement“ nicht länger valide ist und die Karte deshalb mit einem roten Hintergrund eingefärbt werden muss (Abb. 2).



Widget	Location	Last Frame	Current Screen
Checkbox	selection_card.dart:122	7	42
Text	selection_card.dart:126	7	42
Card	selection_card.dart:67	2	7

Abbildung 2: XXXXX, Quelle: Eigene Abbildung

Ohne die Änderungen in diesem Schritt zeigt der „Flutter Performance“-Tab, dass sich bei jeder Auswahl einer Option sechs Card-Elemente aktualisieren (Abb. 3). Das ist der Fall, weil es in Summe sechs Auswahlfelder gibt.



Widget	Location	Last Frame	Current Screen
Checkbox	selection_card.dart:122	7	28
Text	selection_card.dart:126	7	28
Card	selection_card.dart:67	6	12

Abbildung 3: XXXXX, Quelle: Eigene Abbildung

Teil II

Anhang

Teil III

Implementierung

A Schritt 1 Anhang

Ich wurde zwischengeparkt

Strategie Entwurfsmuster Das Strategie Entwurfsmuster ist ein Verhaltensmuster. Es erlaubt Algorithmen zu kapseln und auszutauschen. Die Typdefinition `OnSelectCallback` (Z. 4) kann nach dem Strategie-Entwurfsmuster als die Schnittstelle namens „*Strategie*“ interpretiert werden. Sie definiert, welche Voraussetzung an die Schnittstelle gegeben ist. In diesem Fall ist die Voraussetzung, dass es sich um eine Funktion ohne Rückgabewert handelt, der eine Maßnahme als erstes Argument übergeben wird. Dementsprechend ist der Parameter `onSelect` im Konstruktor die sogenannte „*konkrete Strategie*“, die dieser Schnittstelle entsprechen muss. Der „*Kontext*“ ist schließlich die aufrufende Oberfläche `MassnahmenMasterScreen`. Die konkrete Strategie, die der Übersichts-Bildschirm der Tabelle übergibt, verwendet die selektierte Maßnahme, um damit die Eingabemaske zu öffnen.

Ich wurde zwischengeparkt

```
48 test('Storage with one Massnahme deserialises without error', () {
49   var json = {
50     "massnahmen": [
51       {
52         "guid": "test massnahme id",
53         "letzteBearbeitung": {
54           "letztesBearbeitungsDatum": 0,
55           "letzterStatus": "bearb"
56         },
57         "identifikatoren": {"massnahmenTitel": "Massnahme 1"}
58       }
59     ]
60   };
61
62   var expectedStorage = Storage();
63   expectedStorage =
64     expectedStorage.rebuild((b) => b.massnahmen.add(Massnahme((b) => b
65       ..guid = "test massnahme id"
66       ..identifikatoren.massnahmenTitel = "Massnahme 1"
67       ..letzteBearbeitung.update((b) {
68         b.letztesBearbeitungsDatum =
69           DateTime.fromMillisecondsSinceEpoch(0, isUtc: true);
70       })))));
71
72   var actualStorage = serializers.deserializeWith(Storage.serializer, json);
73
74   expect(actualStorage, equals(expectedStorage));
```

Listing 11: Ein automatisierter Testfall überprüft, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-1/conditional_form/test/data_model/storage_test.dart](#)

B Schritt 2 Anhang

C Schritt 3 Anhang


```

5 class FoerderklasseChoice extends Choice {
6   static final oelb = FoerderklasseChoice("oelb", "Ökolandbau");
7   static final azl = FoerderklasseChoice("azl", "Ausgleichszulage");
8   static final ea = FoerderklasseChoice("ea", "Erschwerenausgleich");
9   static final aukm_nur_vns = FoerderklasseChoice("aukm_nur_vns",
10     "Agrarumwelt-(und Klima)Maßnahme: nur Vertragsnaturschutz");
11   static final aukm_ohne_vns = FoerderklasseChoice("aukm_ohne_vns",
12     "Agrarumwelt-(und Klima)Maßnahmen, tw. auch mit Tierwohlaspekten, aber OHNE
13     ↳ Vertragsnaturschutz");
14   static final twm_ziel = FoerderklasseChoice(
15     "twm_ziel", "Tierschutz/Tierwohlmaßnahmen mit diesem als Hauptziel");
16   static final contact =
17     FoerderklasseChoice("contact", "bitte um Unterstützung");
18   FoerderklasseChoice(String abbreviation, String description,
19     {bool Function(Set<Choice> choices)? condition})
20     : super(abbreviation, description);
21 }
22
23 final foerderklasseChoices = Choices<FoerderklasseChoice>({
24   FoerderklasseChoice.oelb,
25   FoerderklasseChoice.azl,
26   FoerderklasseChoice.ea,
27   FoerderklasseChoice.aukm_nur_vns,
28   FoerderklasseChoice.aukm_ohne_vns,
29   FoerderklasseChoice.twm_ziel,
30   FoerderklasseChoice.contact
31 }, name: "Förderklasse");
32
33 class KategorieChoice extends Choice {
34   static final zf_us =
35     KategorieChoice("zf_us", "Anbau Zwischenfrucht/Untersaat");
36   static final anlage_pflege =
37     KategorieChoice("anlage_pflege", "Anlage/Pflege Struktur");
38   static final dungmang = KategorieChoice("dungmang", "Düngemanagement");
39   static final extens = KategorieChoice("extens", "Extensivierung");
40   static final flst = KategorieChoice("flst", "Flächenstilllegung/Brache");
41   static final umwandlg = KategorieChoice("umwandlg", "Nutzungsumwandlung");
42   static final bes_kult_rass = KategorieChoice(
43     "bes_kult_rass", "Förderung bestimmter Rassen / Sorten / Kulturen");
44   static final contact = KategorieChoice("contact", "bitte um Unterstützung");
45
46   KategorieChoice(String abbreviation, String description)
47     : super(abbreviation, description);
48 }
49
50 final kategorieChoices = Choices<KategorieChoice>({
51   KategorieChoice.zf_us,
52   KategorieChoice.anlage_pflege,
53   KategorieChoice.dungmang,
54   KategorieChoice.extens,
55   KategorieChoice.flst,
56   KategorieChoice.umwandlg,
57   KategorieChoice.bes_kult_rass,
58   KategorieChoice.contact
59 }, name: "Kategorie");

```

Listing 12: Die Mengen foerderklasseChoices und kategorieChoices , Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-2/conditional_form/lib/choices/choices.dart](#)

```

61 class ZielflaecheChoice extends Choice {
62   static final ka = ZielflaecheChoice("ka", "keine Angabe/Vorgabe");
63   static final al = ZielflaecheChoice("al", "AL");
64   static final gl = ZielflaecheChoice("gl", "GL");
65   static final lf = ZielflaecheChoice("lf", "LF");
66   static final dk_sk = ZielflaecheChoice("dk_sk", "DK/SK");
67   static final hff = ZielflaecheChoice("hff", "HFF");
68   static final biotop_le =
69     ZielflaecheChoice("biotop_le", "Landschaftselement/Biotop o.Ä.");
70   static final wald = ZielflaecheChoice("wald", "Wald/Forst");
71   static final contact = ZielflaecheChoice("contact", "bitte um Unterstützung");
72
73   ZielflaecheChoice(String abbreviation, String description)
74     : super(abbreviation, description);
75 }
76
77 final zielflaecheChoices = Choices<ZielflaecheChoice>({
78   ZielflaecheChoice.ka,
79   ZielflaecheChoice.al,
80   ZielflaecheChoice.gl,
81   ZielflaecheChoice.lf,
82   ZielflaecheChoice.dk_sk,
83   ZielflaecheChoice.hff,
84   ZielflaecheChoice.biotop_le,
85   ZielflaecheChoice.wald,
86   ZielflaecheChoice.contact
87 }, name: "Zielfläche");
88
89 class ZieleinheitChoice extends Choice {
90   static final ka = ZieleinheitChoice("ka", "keine Angabe/Vorgabe");
91   static final m3 = ZieleinheitChoice("m3", "m³ (z.B. Gülle)");
92   static final pieces =
93     ZieleinheitChoice("pieces", "Kopf/Stück (z.B. Tiere oder Bäume)");
94   static final gve = ZieleinheitChoice("gve", "GV/GVE");
95   static final rgve = ZieleinheitChoice("rgve", "RGV");
96   static final ha = ZieleinheitChoice("ha", "ha");
97   static final contact = ZieleinheitChoice("contact", "bitte um Unterstützung");
98
99   ZieleinheitChoice(String abbreviation, String description)
100     : super(abbreviation, description);
101 }
102
103 final zieleinheitChoices = Choices<ZieleinheitChoice>({
104   ZieleinheitChoice.ka,
105   ZieleinheitChoice.m3,
106   ZieleinheitChoice.pieces,
107   ZieleinheitChoice.gve,
108   ZieleinheitChoice.rgve,
109   ZieleinheitChoice.ha,
110   ZieleinheitChoice.contact
111 }, name: "Zieleinheit");

```

Listing 13: Die Mengen zielflaecheChoices und zieleinheitChoices, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-2/conditional_form/lib/choices/choices.dart](#)

```

113 class ZielsetzungLandChoice extends Choice {
114   static final ka = ZielsetzungLandChoice("ka", "keine Angabe/Vorgabe");
115   static final bsch = ZielsetzungLandChoice("bsch", "Bodenschutz");
116   static final wsch = ZielsetzungLandChoice("wsch", "Gewässerschutz");
117   static final asch = ZielsetzungLandChoice("asch", "Spezieller Artenschutz");
118   static final biodiv = ZielsetzungLandChoice("biodiv", "Biodiversität");
119   static final struktkviel =
120     ZielsetzungLandChoice("struktkviel", "Erhöhung der Strukturvielfalt");
121   static final genet_res = ZielsetzungLandChoice("genet_res",
122     "Erhaltung genetischer Ressourcen (Pflanzen, z. B. im Grünland, und Tiere, z. B.
123     ↳ bedrohte Rassen)");
124   static final tsch = ZielsetzungLandChoice(
125     "tsch", "Tierschutz/Maßnahmen zum Tierwohl im Betrieb");
126   static final klima = ZielsetzungLandChoice("klima", "Klima");
127   static final contact =
128     ZielsetzungLandChoice("contact", "bitte um Unterstützung");
129
130   ZielsetzungLandChoice(String abbreviation, String description)
131     : super(abbreviation, description);
132 }
133
134 final _zielsetzungLandChoices = {
135   ZielsetzungLandChoice.ka,
136   ZielsetzungLandChoice.bsch,
137   ZielsetzungLandChoice.wsch,
138   ZielsetzungLandChoice.asch,
139   ZielsetzungLandChoice.biodiv,
140   ZielsetzungLandChoice.struktkviel,
141   ZielsetzungLandChoice.genet_res,
142   ZielsetzungLandChoice.tsch,
143   ZielsetzungLandChoice.klima,
144   ZielsetzungLandChoice.contact
145 };
146
147 final hauptzielsetzungLandChoices = Choices<ZielsetzungLandChoice>(
148   _zielsetzungLandChoices,
149   name: "Hauptzielsetzung Land");

```

Listing 14: Die Menge hauptzielsetzungLandChoices, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-2/conditional_form/lib/choices/choices.dart](#)

```

35 void saveRecord() {
36   ScaffoldMessenger.of(context)
37     ..hideCurrentSnackBar()
38     ..showSnackBar(
39       const SnackBar(content: Text('Massnahme wird gespeichert ...')));
40
41   model.putMassnahmeIfAbsent(vm.model);
42 }

```

Listing 15: Die Maßnahmencharakteristika Selektionskarten werden ergänzt, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-3/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart](#)

D Schritt 4 Anhang

```
33 class KategorieChoice extends Choice {
34   static final zf_us = KategorieChoice(
35     "zf_us", "Anbau Zwischenfrucht/Untersaat",
36     condition: (choices) =>
37       choices.contains(FoerderklasseChoice.aukm_ohne_vns));
38   static final anlage_pflege = KategorieChoice(
39     "anlage_pflege", "Anlage/Pflege Struktur",
40     condition: (choices) =>
41       choices.contains(FoerderklasseChoice.aukm_nur_vns) ||
42       choices.contains(FoerderklasseChoice.aukm_ohne_vns));
43   static final dungmang = KategorieChoice("dungmang", "Düngemanagement",
44     condition: (choices) =>
45       choices.contains(FoerderklasseChoice.aukm_nur_vns) ||
46       choices.contains(FoerderklasseChoice.aukm_ohne_vns));
47   static final extens = KategorieChoice("extens", "Extensivierung");
48   static final flst = KategorieChoice("flst", "Flächenstilllegung/Brache",
49     condition: (choices) =>
50       choices.contains(FoerderklasseChoice.aukm_nur_vns) ||
51       choices.contains(FoerderklasseChoice.aukm_ohne_vns));
52   static final umwandlg = KategorieChoice("umwandlg", "Nutzungsumwandlung",
53     condition: (choices) =>
54       choices.contains(FoerderklasseChoice.aukm_nur_vns) ||
55       choices.contains(FoerderklasseChoice.aukm_ohne_vns));
56   static final bes_kult_rass = KategorieChoice(
57     "bes_kult_rass", "Förderung bestimmter Rassen / Sorten / Kulturen",
58     condition: (choices) => !choices.contains(FoerderklasseChoice.ea));
59   static final contact = KategorieChoice("contact", "bitte um Unterstützung");
60
61   KategorieChoice(String abbreviation, String description,
62     {bool Function(Set<Choice> choices)? condition})
63     : super(abbreviation, description, condition: condition);
64 }
```

Listing 16: XXXXX, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-4/conditional_form/lib/choices/choices.dart](#)

```

77 class ZielflaecheChoice extends Choice {
78   static final ka = ZielflaecheChoice("ka", "keine Angabe/Vorgabe");
79   static final al = ZielflaecheChoice("al", "AL",
80     condition: (choices) => !choices.contains(KategorieChoice.zf_us));
81   static final gl = ZielflaecheChoice("gl", "GL");
82   static final lf = ZielflaecheChoice("lf", "LF");
83   static final dk_sk = ZielflaecheChoice("dk_sk", "DK/SK",
84     condition: (choices) => !choices.contains(FoerderklasseChoice.twm_ziel));
85   static final hff = ZielflaecheChoice("hff", "HFF");
86   static final biotop_le = ZielflaecheChoice(
87     "biotop_le", "Landschaftselement/Biotop o.Ä.",
88     condition: (choices) =>
89       (choices.contains(FoerderklasseChoice.azl) ||
90         choices.contains(FoerderklasseChoice.ea) ||
91         choices.contains(FoerderklasseChoice.aukm_nur_vns) ||
92         choices.contains(FoerderklasseChoice.aukm_ohne_vns)) &&
93       (!choices.contains(KategorieChoice.zf_us) ||
94         !choices.contains(KategorieChoice.bes_kult_rass)));
95   static final wald = ZielflaecheChoice("wald", "Wald/Forst",
96     condition: (choices) =>
97       (choices.contains(FoerderklasseChoice.ea) ||
98         choices.contains(FoerderklasseChoice.aukm_nur_vns) ||
99         choices.contains(FoerderklasseChoice.aukm_ohne_vns)) &&
100       (!choices.contains(KategorieChoice.zf_us) ||
101         !choices.contains(KategorieChoice.bes_kult_rass)));
102   static final contact = ZielflaecheChoice("contact", "bitte um Unterstützung");
103
104   ZielflaecheChoice(String abbreviation, String description,
105     {bool Function(Set<Choice> choices)? condition})
106     : super(abbreviation, description, condition: condition);
107 }

```

Listing 17: XXXXX, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-4/conditional_form/lib/choices/choices.dart](#)

```

121 class ZieleinheitChoice extends Choice {
122   static final ka = ZieleinheitChoice("ka", "keine Angabe/Vorgabe");
123   static final m3 = ZieleinheitChoice("m3", "m3 (z.B. Gülle)",
124     condition: (choices) =>
125       (choices.contains(FoerderklasseChoice.aukm_nur_vns) ||
126         choices.contains(FoerderklasseChoice.aukm_ohne_vns)) &&
127       (choices.contains(KategorieChoice.dungmang) ||
128         choices.contains(KategorieChoice.extens)) &&
129       (!choices.contains(ZielflaecheChoice.ka) &&
130         !choices.contains(ZielflaecheChoice.contact)));
131   static final pieces = ZieleinheitChoice(
132     "pieces", "Kopf/Stück (z.B. Tiere oder Bäume)",
133     condition: (choices) =>
134       (choices.contains(FoerderklasseChoice.aukm_nur_vns) ||
135         choices.contains(FoerderklasseChoice.aukm_ohne_vns) ||
136         choices.contains(FoerderklasseChoice.twm_ziel)) &&
137       (!choices.contains(KategorieChoice.zf_us) ||
138         !choices.contains(KategorieChoice.flst) ||
139         !choices.contains(KategorieChoice.umwandlg)) &&
140       (!choices.contains(ZielflaecheChoice.ka) &&
141         !choices.contains(ZielflaecheChoice.contact)));
142   static final gve = ZieleinheitChoice("gve", "GV/GVE",
143     condition: (choices) =>
144       (choices.contains(FoerderklasseChoice.aukm_nur_vns) ||
145         choices.contains(FoerderklasseChoice.aukm_ohne_vns) ||
146         choices.contains(FoerderklasseChoice.twm_ziel)) &&
147       (!choices.contains(KategorieChoice.zf_us) ||
148         !choices.contains(KategorieChoice.anlage_pflege) ||
149         !choices.contains(KategorieChoice.flst) ||
150         !choices.contains(KategorieChoice.umwandlg)) &&
151       (!choices.contains(ZielflaecheChoice.ka) &&
152         !choices.contains(ZielflaecheChoice.contact)));
153   static final rgve = ZieleinheitChoice("rgve", "RGV",
154     condition: (choices) =>
155       (choices.contains(FoerderklasseChoice.aukm_nur_vns) ||
156         choices.contains(FoerderklasseChoice.aukm_ohne_vns) ||
157         choices.contains(FoerderklasseChoice.twm_ziel)) &&
158       (!choices.contains(KategorieChoice.zf_us) ||
159         !choices.contains(KategorieChoice.anlage_pflege) ||
160         !choices.contains(KategorieChoice.flst) ||
161         !choices.contains(KategorieChoice.umwandlg)) &&
162       (!choices.contains(ZielflaecheChoice.ka) &&
163         !choices.contains(ZielflaecheChoice.contact)));
164   static final ha = ZieleinheitChoice("ha", "ha",
165     condition: (choices) =>
166       !choices.contains(ZielflaecheChoice.ka) &&
167       !choices.contains(ZielflaecheChoice.contact));
168   static final contact = ZieleinheitChoice("contact", "bitte um Unterstützung");
169
170   ZieleinheitChoice(String abbreviation, String description,
171     {bool Function(Set<Choice> choices)? condition})
172     : super(abbreviation, description, condition: condition);
173 }

```

Listing 18: XXXXX, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-4/conditional_form/lib/choices/choices.dart](#)

E Schritt 6 Anhang

F Schritt 6 Anhang

Literatur

Google LLC. *Dart - Effective Dart - Style - PREFER using _, __, etc. for unused callback parameters*. URL: https://web.archive.org/web/20210728114518/https://dart.dev/guides/language/effective-dart/style#prefer-using-_-__-etc-for-unused-callback-parameters (besucht am 08.08.2021) (Zitiert auf der Seite 5).

JetBrains s.r.o. *Kotlin - High-order functions and lambdas - Underscore for unused variables*. URL: http://web.archive.org/web/20210331062820if_/https://kotlinlang.org/docs/lambdas.html#underscore-for-unused-variables (besucht am 08.08.2021) (Zitiert auf der Seite 5).