Teil I

Implementierung

0.1 Schritt 7

Nachdem im letzten Schritt nun die Mehrfachauswahl für die Nebenziel hinzugefügt wurde, soll in diesem Schritt die Möglichkeit geschaffen werden, benutzerdefinierte Abhängigkeiten für Auswahloptionen anzugeben. Denn die Nebenziele haben mehrere besondere Voraussetzungen:

Sollte das Hauptziel nicht gesetzt sein oder die Option *„keine Angabe/Vorgabe“* oder *„bitte um Unterstützung“* enthalten, so ist es nicht sinnvoll, dass ein tatsächliches Nebenziel gewählt wird. In diesem Fall kommen wiederum nur die Werte *„keine Angabe/Vorgabe“* oder *„bitte um Unterstützung“* infragen.

Sollte dagegen ein Hauptziel gesetzt sein, so darf das Nebenziel nicht die gleiche Option enthalten. Diese Bedingungen lassen sich nicht mit Funktion `condition` der Basisklasse `Choice` lösen.

Denn das Argument `priorChoices`, welches der Funktion `condition` übergeben wird, enthält zwar alle Auswahloptionen, die im gesamten Formular gewählt worden, gibt aber keine Auskunft darüber, von welchem Auswahlfeld sie stammen. Sollte also die Auswahloptionen *„Biodiversität“* in der Menge der `priorChoices` auftauchen, so ist unklar, ob sie im Auswahlfeld für das Hauptziel oder dem der Nebenziele gewählt wurde.

Wenn der Selektionskarte aber eine benutzerdefinierte Funktion übergeben werden könnte, welche im aufrufenden Kontext auch Zugriff auf das ViewModel hat, so könnte direkt auf die Auswahlfelder zugegriffen werden.

Zu diesem Zweck wird der Klasse `SelectionCard` die Instanzvariable `choiceMatcher` hinzugefügt (Listing 1, Z. 27). Ein Parameter des gleichen Namens wird den Hilfsmethoden `buildSelectionCard` und `buildMultiSelectionCard` welche ihn unverändert an den Konstruktor der Klasse `SelectionCard` weitergeleitet. Die entsprechenden Listing sind in Anhang A auf den Seiten 8 und 9 zu finden.

Der initialisierende Wert kann im Konstruktor gesetzt (Z. 41), aber auch ausgelassen werden, da er nicht mit dem `required`-Schlüsselwort gekennzeichnet und damit nicht verpflichtend ist. Doch aus diesem Grund kann der Parameter den Wert `null` annehmen, weshalb er mit dem Suffix `?` gekennzeichnet werden muss. In der Initialisierungsliste erfolgt die Initialisierung der Instanzvariable `choiceMatcher` (Z. 46). Sollte der im Konstruktor übergebene Parameter nicht `null` sein, so wird er der Instanzvariable zugewiesen. Ist der aber `null`, so sorgt die *„If-null Expression“* dafür, dass der Standardwert rechts von dem `??` zugewiesen wird: die Funktion `defaultChoiceMatcherStrategy 46`. Diese Funktion kapselt die Überprüfung der Abhängigkeiten – welche die Auswahloption und untereinander haben – so wie sie in den letzten Schritten durchgeführt wurde (Z. 16-18). Ihr wird die zu überprüfende Auswahloption `choice`, sowie die Menge `priorChoices` – die mit allen bisher ausgewählten Auswahloptionen im Formular gefüllt ist – übergeben (Z. 16). Die Auswahloption `choice` ruft – wie zuvor auch – die Methode `conditionMatches` auf und übergibt ihr das Objekt `priorChoices` (Z. 17). Diese Implementierung soll immer dann verwendet

```
13 typedef ChoiceMatcher<ChoiceType extends Choice> = bool Function(
14     ChoiceType choice, Set<Choice> priorChoices);
15
16 bool defaultChoiceMatcherStrategy(Choice choice, Set<Choice> priorChoices) {
17   return choice.conditionMatches(priorChoices);
18 }
19
20 const confirmButtonTooltip = 'Auswahl übernehmen';
21
22 class SelectionCard<ChoiceType extends Choice> extends StatelessWidget {
23   final String title;
24   final bool multiSelection;
25   final BehaviorSubject<BuiltSet<ChoiceType>> selectionViewModel;
26   final Choices<ChoiceType> allChoices;
27   final BehaviorSubject<Set<Choice>> priorChoices;
28   final OnSelect<ChoiceType> onSelect;
29   final OnDeselect<ChoiceType> onDeselect;
30   final String? errorText;
31   final ChoiceMatcher<ChoiceType> choiceMatcher;
32
33   SelectionCard(
34       {required this.title,
35       this.multiSelection = false,
36       required Iterable<ChoiceType> initialValue,
37       required this.allChoices,
38       required this.priorChoices,
39       required this.onSelect,
40       required this.onDeselect,
41       ChoiceMatcher<ChoiceType>? choiceMatcher,
42       this.errorText,
43       Key? key})
44       : selectionViewModel = BehaviorSubject<BuiltSet<ChoiceType>>.seeded(
45             BuiltSet.from(initialValue)),
46         this.choiceMatcher = choiceMatcher ?? defaultChoiceMatcherStrategy,
47         super(key: key);
```

Listing 1: XXXX, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-7/conditional_form/lib/widgets/selection_card.dart

werden, wenn kein benutzerdefinierter `choiceMatcher` übergeben wurde. An dem Namen `defaultChoiceMatcherStrategy` wird offensichtlich, um welches Entwurfsmuster es sich hierbei handelt: das *„Strategie-Entwurfsmuster“*.

Strategie-Entwurfsmuster Das Strategie-Entwurfsmuster ist ein Verhaltensmuster der Gang of Four. Es erlaubt Algorithmen zu kapseln und auszutauschen[1]. Abbildung 1 zeigt das UML-Diagramm des *„Strategie-Entwurfsmusters“*.

Die Typdefinition `ChoiceMatcher` (Z. 13) 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 mit dem Rückgabewert `bool` handelt, der als erstes Argument eine Auswahloption – der Parameterbezeichner lautet `choice` – und als zweites Argument eine Menge von Auswahloptionen – der Parameterbezeichner ist `priorChoices` – übergeben wird. Sollte der Parameter `choiceMatcher` gesetzt sein, so tauscht er die standardmäßig genutztes

[1] Vgl. Gamma u. a., *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*, S. 373.

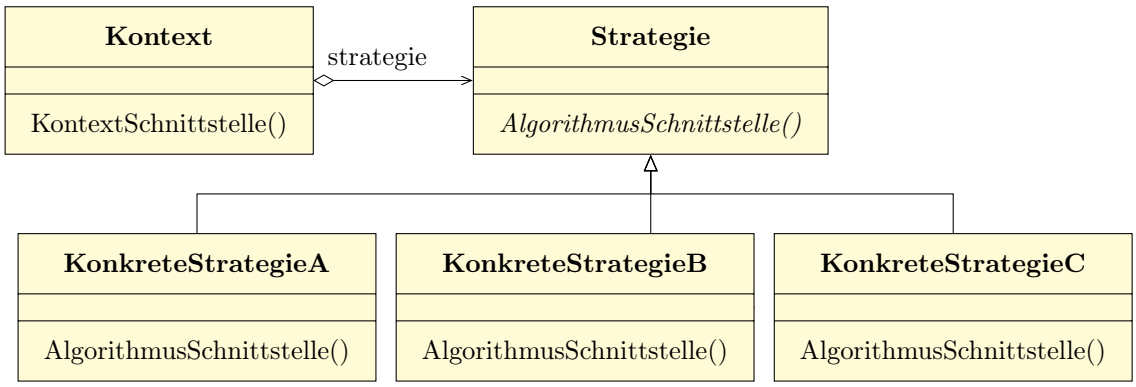

Abbildung 1: UML Diagramme, Quelle: Eigene Abbildung

Strategie `defaultChoiceMatcherStrategy` durch die benutzerdefinierte Strategie aus (Z. 46). Beide werden nach dem Strategie-Entwurfsmuster als *„konkrete Strategien“* bezeichnet. Im Entwurfsmuster gibt es noch den Akteur *„Kontext“*, wobei es sich um die aufrufende Klasse handelt, welche die Strategien verwendet. In diesem Fall ist das die *„Klasse“* `SelectionCard`. Abbildung 2 zeigt das UML-Diagramm der konkreten Implementierung des *„Strategie-Entwurfsmusters“* für die *„Strategie“* `ChoiceMatcher`. Da sich bei der konkreten Strategie für das Auswahlfeld der *„Nebenziele“* um eine anonyme Funktion handelt, wurde sie zum besseren Verständnis im UML-Diagramm *„nebenzieleChoiceMatcherStrategy“* genannt.

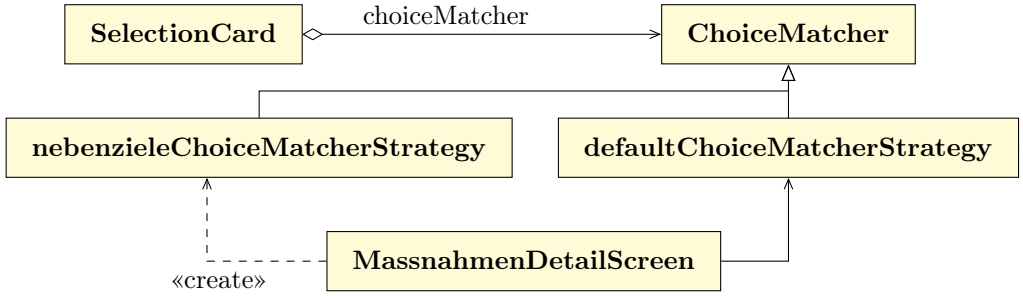

Abbildung 2: UML Diagramme, Quelle: Eigene Abbildung

Im Diagramm ist ebenfalls der *„View“* `MassnahmenDetailScreen` enthalten, denn er verwendet die konkrete Strategie `defaultChoiceMatcherStrategy` für die Validierung (Listing 8).

```
119 Widget buildSelectionCard<ChoiceType extends Choice>(
120     {required Choices<ChoiceType> allChoices,
121     required BehaviorSubject<ChoiceType?> selectionViewModel,
122     ChoiceMatcher<ChoiceType>? choiceMatcher}) {
123   return FormField(
124       validator: (_) => validateChoices(
125           name: allChoices.name,
126           choices: {
127             if (selectionViewModel.value != null) selectionViewModel.value!
128           },
129           priorChoices: vm.priorChoices.value,
130           choiceMatcher: choiceMatcher ?? defaultChoiceMatcherStrategy),
```

Listing 2: XXXX, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-7/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

Sollte nämlich ein Argument für den Parameter `choiceMatcher` übergeben werden (Z. 122), so wird es auch für die Validierung verwendet (Z. 130). Ist das Argument aber nicht gesetzt und damit `null`, so sorgt die *„If-null Expression“* dafür, dass die `defaultChoiceMatcherStrategy` für die Validierung verwendet wird.

Außerdem erstellt `MassnahmenDetailScreen` die konkrete Strategie *„nebenzieleChoiceMatcherStrategy“*, wie in Listing 3 zu sehen ist.

```
221 buildMultiSelectionCard<ZielsetzungLandChoice>(
222   allChoices: nebenzielsetzungLandChoices,
223   selectionViewModel: vm.nebenziele,
224   choiceMatcher: (choice, priorChoices) {
225     if (choice.hasRealValue) {
226       if (vm.hauptzielsetzungLand.value == null ||
227           vm.hauptzielsetzungLand.value!
228               .hasNoRealValue) {
229         return false;
230       } else if (choice ==
231           vm.hauptzielsetzungLand.value) {
232         return false;
233       } else {
234         return true;
235       }
236     }
237     return true;
238   },
239 ),
```

Listing 3: XXXX, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-7/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

Der Aufruf `buildMultiSelectionCard` wird um die Übergabe einer anonymen Funktion für den Parameter `choiceMatcher` erweitert (Z. 224-239). In der ersten Fallunterscheidung wird überprüft, ob die gewählte Option ein tatsächliches Nebenziel ist (Z. 225). Dies kann über die Getter-Methode `hasRealValue` abgefragt werden. Ist dies nicht der Fall, so handelt es sich um die Auswahloptionen *„keine Angabe/Vorgabe“* bzw. *„bitte um Unterstützung“*, weshalb `true` zurückgegeben werden kann (Z. 237), da diese Auswahloptionen immer erlaubt sind. Sollte sich dagegen um ein tatsächliches Nebenziel handeln, so überprüft die nächste Fallunterscheidung, ob das Hauptziel entweder nicht gesetzt ist oder mit einem nicht tatsächlichen Hauptziel belegt ist (Z. 226-228)., Dazu wird die Getter-Methode `hasNoRealValue` benutzt, welche als Gegenteil zu `hasRealValue` fungiert, und dementsprechend `true` zurückgibt wenn die Auswahloption entweder *„keine Angabe/Vorgabe“* oder *„bitte um Unterstützung“* ist (Z. 226-228). Sollte das Hauptziel keinen tatsächlichen Wert einer Zielsetzung enthalten, dann ist die Wahl eines oder mehrerer Nebenziele nicht sinnvoll. Waren beide zuvorigen Bedingungen nicht wahr, so steht bereits fest, dass sowohl das Hauptziel, als auch den Nebenziel gesetzt sind und weder die Option *„keine Angabe/Vorgabe“* oder *„bitte um Unterstützung“* enthalten. Nun soll eine letzte Fallunterscheidung überprüfen, ob das Nebenziel bereits im Hauptziel gesetzt ist (Z. 230-321). Das ist nicht erlaubt, weshalb `false` zurückgegeben werden soll (Z. 232). Anderenfalls sind alle Bedingungen erfüllt und `true` kann zurückgegeben werden.

An diesem Beispiel wird auch offensichtlich, welchen Nutzen die Generalisierung der Klasse

`SelectionCard` hat. Der Typparameter `ZielsetzungLandChoice` wird beim Aufruf der Methode `buildMultiSelectionCard` übergeben (Z. 221). Die Methode übergibt den Typparameter wiederum der Klasse `SelectionCard` (Listing 4).

```
131 builder: (field) => SelectionCard<ChoiceType>(
```

Listing 4: XXXX, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-7/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

Schließlich übergibt die Klasse `SelectionCard` den Typparameter an die Instanzvariable `choiceMatcher` (Listing 5).

```
41 ChoiceMatcher<ChoiceType>? choiceMatcher,
```

Listing 5: XXXX, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-7/conditional_form/lib/widgets/selection_card.dart

Damit handelt es sich also auch bei dem ersten Parameter `choice` der anonymen Funktion, die dem Parameter `choiceMatcher` übergeben wird um den Typ `ZielsetzungLandChoice`. Aus diesem Grund können die Methoden `hasRealValue` (Z. 225) und `hasNoRealValue` (Z. 228) auf dem Objekt `choice` aufrufen werden, obwohl sie nur Teil der Klasse `ZielsetzungLandChoice` aber nicht der Basisklasse `Choice` sind. Ohne Parametrisierung über den Typ müsste das Objekt `choice` in einen anderen Typen umgewandelt werden. Doch nach dieser Typumwandlung könnte ein Laufzeitfehler geschehen, sollte es sich bei dem Objekt tatsächlich nicht um den gewünschten Typ handeln. Durch die Generalisierung der Klassen und die Angabe des Typparameters ist das Vorhandensein des richtigen Typs garantiert und keine Typumwandlung nötig.

Die beiden neuen Methoden sind in Listing 6 zu sehen.

```
185 class ZielsetzungLandChoice extends Choice {
186   static final ka = ZielsetzungLandChoice("ka", "keine Angabe/Vorgabe");
...
198   static final contact =
199       ZielsetzungLandChoice("contact", "bitte um Unterstützung");
200
201   bool get hasRealValue => this != ka && this != contact;
202
203   bool get hasNoRealValue => !hasRealValue;
```

Listing 6: XXX, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-7/conditional_form/lib/choices/choices.dart

`hasRealValue` vergleicht, ob der aktuelle Wert weder *„keine Angabe/Vorgabe“* noch *„bitte um Unterstützung“* ist (Z. 201). `hasNoRealValue` ruft dagegen intern `hasRealValue` auf und negiert Wert (Z. 203).

Überall dort, wo zuvor der Ausdruck `choice.conditionMatches(priorChoices)` verwendet wurde, muss nun der Aufruf des `choiceMatcher` erfolgen. So zum Beispiel der Stream, welcher die Validität der Auswahlfelder prüft (Listing 7).

```
63 final validityChanged = priorChoices
64     .map((choices) =>
65         selectionViewModel.value.any((c) => !choiceMatcher(c, choices)))
66     .distinct();
```

Listing 7: XXXX, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-7/conditional_form/lib/widgets/selection_card.dart

Alle Vorkommnisse, die durch den neuen Ausdruck ersetzt werden, sind im Anhang A auf den Seiten 10 bis 12 zu finden.

Teil II

Anhang

Teil III

Implementierung

A Schritt 7 Anhang

```
Widget buildSelectionCard<ChoiceType extends Choice>(
    {required Choices<ChoiceType> allChoices,
    required BehaviorSubject<ChoiceType?> selectionViewModel,
    ChoiceMatcher<ChoiceType>? choiceMatcher}) {
  return FormField(
      validator: (_) => validateChoices(
          name: allChoices.name,
          choices: {
            if (selectionViewModel.value != null) selectionViewModel.value!
          },
          priorChoices: vm.priorChoices.value,
          choiceMatcher: choiceMatcher ?? defaultChoiceMatcherStrategy),
      builder: (field) => SelectionCard<ChoiceType>(
            title: allChoices.name,
            allChoices: allChoices,
            priorChoices: vm.priorChoices,
            initialValue: {
              if (selectionViewModel.value != null)
                selectionViewModel.value!
            },
            choiceMatcher: choiceMatcher,
            onSelect: (selectedChoice) =>
                selectionViewModel.value = selectedChoice,
            onDeselect: (selectedChoice) => selectionViewModel.value = null,
            errorText: field.errorText,
          ));
}
```

Listing 8: XXXX, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-7/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

```
147 Widget buildMultiSelectionCard<ChoiceType extends Choice>(
148     {required Choices<ChoiceType> allChoices,
149     required BehaviorSubject<BuiltSet<ChoiceType>> selectionViewModel,
150     ChoiceMatcher<ChoiceType>? choiceMatcher}) {
151   return FormField(
152       validator: (_) => validateChoices(
153           name: allChoices.name,
154           choices: selectionViewModel.value,
155           priorChoices: vm.priorChoices.value,
156           choiceMatcher: choiceMatcher ?? defaultChoiceMatcherStrategy),
157       builder: (field) => SelectionCard<ChoiceType>(
158             title: allChoices.name,
159             multiSelection: true,
160             allChoices: allChoices,
161             priorChoices: vm.priorChoices,
162             initialValue: selectionViewModel.value,
163             choiceMatcher: choiceMatcher,
164             onSelect: (selectedChoice) => selectionViewModel.value =
165                 selectionViewModel.value
166                     .rebuild((b) => b.add(selectedChoice)),
167             onDeselect: (selectedChoice) => selectionViewModel.value =
168                 selectionViewModel.value
169                     .rebuild((b) => b.remove(selectedChoice)),
170             errorText: field.errorText,
171           ));
172 }
```

Listing 9: XXXX, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-7/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

```dart
Widget build(BuildContext context) {
  final focusNode = FocusNode();

  navigateToSelectionScreen() async {
    focusNode.requestFocus();

    Navigator.push(
        context,
        MaterialPageRoute(
            builder: (context) =>
                createMultipleChoiceSelectionScreen(context)));
  }

  final validityChanged = priorChoices
      .map((choices) =>
          selectionViewModel.value.any((c) => !choiceMatcher(c, choices)))
      .distinct();

  final needsRepaint = BehaviorSubject.seeded(true);
  validityChanged.listen((value) => needsRepaint.add(true));
  selectionViewModel.listen((value) => needsRepaint.add(true));

  return StreamBuilder(
      stream: needsRepaint,
      builder: (context, snapshot) {
        final selectedChoices = selectionViewModel.value;
        final bool wrongSelection =
            selectedChoices.any((c) => !choiceMatcher(c, priorChoices.value));

        return Card(
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              ListTile(
                focusNode: focusNode,
                title: Text(title),
                subtitle: Text(
                    selectedChoices.map((c) => c.description).join(", ")),
                trailing: const Icon(Icons.edit),
                onTap: navigateToSelectionScreen,
                tileColor:
                    wrongSelection || errorText != null ? Colors.red : null,
              ),
              if (errorText != null)
                Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: Text(errorText!,
                      style:
                          const TextStyle(fontSize: 12.0, color: Colors.red)),
                )
            ],
          ),
        );
```

Listing 10: XXXX, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-7/conditional_form/lib/widgets/selection_card.dart

```
Widget createMultipleChoiceSelectionScreen(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(title),
    ),
    body: StreamBuilder(
        stream: selectionViewModel,
        builder: (context, snapshot) {
          final selectedChoices = selectionViewModel.value;

          Set<ChoiceType> selectedAndSelectableChoices = {};
          Set<ChoiceType> unselectableChoices = {};

          for (ChoiceType c in allChoices) {
            if (selectedChoices.contains(c) ||
                choiceMatcher(c, priorChoices.value)) {
              selectedAndSelectableChoices.add(c);
            } else {
              unselectableChoices.add(c);
            }
          }

          return ListView(children: [
            ...selectedAndSelectableChoices.map((ChoiceType c) {
              bool isSelected = selectedChoices.contains(c);
              bool selectedButDoesNotMatch =
                  !choiceMatcher(c, priorChoices.value);

              return CheckboxListTile(
                  key: Key(
                      "valid choice ${allChoices.name} - ${c.abbreviation}"),
                  controlAffinity: ListTileControlAffinity.leading,
                  title: Text(c.description),
                  tileColor: selectedButDoesNotMatch ? Colors.red : null,
                  value: isSelected,
                  onChanged: (selected) {
                    if (selected != null) {
                      if (multiSelection) {
                        selectionViewModel.value =
                            selectionViewModel.value.rebuild((b) {
                          if (selectionViewModel.value.contains(c)) {
                            b.remove(c);
                          } else {
                            b.add(c);
                          }
                        });
                      } else {
                        selectionViewModel.value =
                            selectionViewModel.value.rebuild((b) {
                          b.replace(isSelected ? [] : [c]);
                        });
                      }
                      if (selected) {
                        onSelect(c);
                      } else {
                        onDeselect(c);
                      }
                    }
                  });
```

Listing 11: XXXX, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-7/conditional_form/lib/widgets/selection_card.dart

```
298 String? validateChoices<ChoiceType extends Choice>(
299     {required String name,
300     required Iterable<ChoiceType> choices,
301     required Set<Choice> priorChoices,
302     required ChoiceMatcher<ChoiceType> choiceMatcher}) {
303   if (choices.isEmpty) {
304     return "Feld ${name} enthält keinen Wert!";
305   }
306
307   bool atLeastOneValueInvalid =
308       choices.any((c) => !choiceMatcher(c, priorChoices));
309
310   if (atLeastOneValueInvalid) {
311     return "Wenigstens ein Wert im Feld ${name} ist fehlerhaft!";
312   }
313
314   return null;
315 }
```

Listing 12: XXXX, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-7/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart