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 3

In diesem Schritt soll die grundlegende Validierungsfunktion hinzugefügt werden. Maßnahmen, die als abgeschlossen markiert sind, dürfen keine leeren Eingabefelder enthalten und der Maßnahmentitel darf nicht doppelt belegt sein. `Flutter` stellt das Widget `Form` für die Validierung von Eingabefeldern bereit.

0.2 Einfügen des Form-Widgets

Das Widget `Form` ist ein Container, welcher die Validierung für alle Kinderelemente des Typs `FormField` ausführt. Damit es alle Eingabefelder im Formular umgibt, wird es oberhalb des `Stack` eingefügt (Listing. 1, Z. 161). Das `Form`-Widget muss über einen `key` registriert werden (Z. 162), damit auf die Validierungsfunktionen zurückgegriffen werden kann.

```
161 child: Form(
162   key: formKey,
163   child: Stack(
164     children: [
165       SingleChildScrollView(
166         child: Center(
```

Listing 1: Die Maßnahmencharakteristika Selektionskarten werden ergänzt, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-3/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

Die Erstellung des `formKey` findet zu Beginn der `build`-Methode des Eingabeformulars statt (Listing. 2, Z. 20). Der `GlobalKey` identifiziert ein Element, welches durch ein Widget gebaut wurde, über die gesamte Applikation hinweg. Es erlaubt darüber hinaus auf das `State`-Objekt zuzugreifen, welches mit dem `StatefulWidget` verknüpft ist. Ohne Angabe eines Typparameters kann nur Zugriff auf Funktionen des Typs `State` gewährt werden. Doch die gewünschte Methode `validate` ist nur Teil des Typs `FormState`. Damit das Element, welches über den `GlobalKey` registriert wurde, auch den `FormState` liefert, kann der entsprechende Typparameter `<FormState>` bei der Erstellung des GlobalKey übergeben werden.

```
17 Widget build(BuildContext context) {
18   final vm = AppState.of(context).viewModel;
19   final model = AppState.of(context).model;
20   final formKey = GlobalKey<FormState>();
```

Listing 2: Die Maßnahmencharakteristika Selektionskarten werden ergänzt, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-3/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

0.3 Validierung des Maßnahmentitels

Das Eingabefeld für den Maßnahmen-Titel ist ein `TextFormField` (Listing. 3, Z. 88). Es erbt vom Typ `FormField` und wird daher mit dem Vaterelement `Form` verknüpft. Es beinhaltet bereits einen Parameter für die Validierungsfunktion namens `validator` (Z. 93). Die übergebene Funktion erhält im ersten Parameter den für das Textfeld eingetragenen Wert. Die Funktion soll `null` zurückgeben, wenn keine Fehler in der Validierung geschehen sind. In jedem anderen Fall soll der Text zurückgegeben werden, der als Fehlermeldung angezeigt werden soll.

Sollte der Parameter `null` sein oder aber ein leerer String (Z. 94), so wird die entsprechende Fehlermeldung `'Bitte Text eingeben'` angezeigt (Z. 96). Damit der Benutzer direkt zu

```dart
83  Widget createMassnahmenTitelTextFormField() {
84    final focusNode = FocusNode();
85    return Card(
86      child: Padding(
87        padding: const EdgeInsets.all(16.0),
88        child: TextFormField(
89          focusNode: focusNode,
90          initialValue: vm.massnahmenTitel.value,
91          decoration: const InputDecoration(
92              hintText: 'Maßnahmentitel', labelText: 'Maßnahmentitel'),
93          validator: (title) {
94            if (title == null || title.isEmpty) {
95              focusNode.requestFocus();
96              return 'Bitte Text eingeben';
97            }
98            var massnahmeTitleDoesAlreadyExists =
99                model.storage.value.massnahmen.any((m) =>
100                   m.guid != vm.guid.value &&
101                   m.identifikatoren.massnahmenTitel ==
102                       vm.massnahmenTitel.value);
103
104           if (massnahmeTitleDoesAlreadyExists) {
105             focusNode.requestFocus();
106             return 'Dieser Maßnahmentitel ist bereits vergeben';
107           }
108           return null;
109         },
110         onChanged: (value) {
111           vm.massnahmenTitel.value = value;
112         },
113       ),
114     ),
115   );
116 }
```

Listing 3: Die Maßnahmencharakteristika Selektionskarten werden ergänzt, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-3/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

dem fehlerhaften Eingabefeld geführt wird, kann ein Objekt der Klasse `FocusNode` verwendet werden. Er wird vor der Konstruktion der Karte erstellt (Z. 84) und dem Parameter `focusNode` des `TextFormField` übergeben (Z. 89). Sollte ein Fehler bei der Validierung gefunden werden, kann mit der Methode `requestFocus` angeordnet werden, den Cursor in das betreffende Feld zu setzen (Z. 95). Das sorgt auch dafür, dass das Eingabefeld in den sichtbaren Bereich gerückt wird.

Sollte das Textfeld nicht leer sein, so soll noch überprüft werden, ob der Maßnahmen-Titel bereits vergeben ist. Über das Model kann die Liste der Maßnahmen angefordert werden (Z. 99). Die Funktion `any` akzeptiert als Argument eine Funktion, die für alle Elemente der Liste ausgeführt wird (Z. 99-102). Wenn die Rückgabe der Funktion auch nur in einem Fall `true` ist, so evaluiert auch `any` mit `true`. Andernfalls ist die Rückgabe `false`. Die anonyme Funktion schließt zunächst den Vergleich mit derselben Maßnahme aus, welche sich gerade in Bearbeitung befindet. Der Vergleich der guid ist dafür ausreichend. Sollte es eine andere Maßnahme geben, welche den gleichen Titel hat (Z. 101-102), so wird Die lokale Variable `massnahmeTitleDoesAlreadyExists` auf `true` gesetzt. Der Benutzer bekommt die entsprechende Fehlermeldung `'Dieser Maßnahmentitel ist bereits vergeben'` zu lesen (Z. 106). Wenn keine der beiden Fallunterscheidungen das `return`-Statement (Z. 96, 106) auslöst, so erfolgt schließlich die Rückgabe von `null`. In dem Kontext der `validator`-Funktion bedeutet die Rückgabe von `null` (Z. 108), dass die Validierung erfolgreich war.

Das `Form`-Widget validiert lediglich Kindelemente vom Typ `FormField`. Dementsprechend

wird das Widget `SelectionCard` nicht in die Validierung miteinbezogen. Es erbt nicht von `FormField`. Es wäre möglich, eine weitere Klasse zu erstellen, die von `FormField` erbt und alle Parameter für die Erstellung einer Selektions-Karte wiederverwendet. Doch das würde bedeuten, dass für alle folgenden Schritte jeder weitere Parameter in beiden Konstruktoren der Klassen gepflegt werden müsste. Um der Arbeit leichter folgen zu können, wurde sich für einen anderen, simpleren Weg entschieden: Die Selektionskarte kann ebenso von einem `FormField` umgeben werden (Listing. 4, Z. 121-148), welches die Selektionskarte in der `builder`-Funktion erstellt und an den Parametern nichts ändert, außer einen weiteren hinzuzufügen: der Text für die Fehlermeldung (Z. 147). Der erste Parameterder `builder`-Funktion ist das `State`-Objekt das `FormField`. Es enthält die Getter-Methode `errorText`, die bei gegebenenfalls fehlgeschlagener Validierung die zurückgegebene Fehlermeldung enthält.

```
118 Widget buildSelectionCard<ChoiceType extends Choice>(
119     {required Choices<ChoiceType> allChoices,
120     required BehaviorSubject<ChoiceType?> selectionViewModel}) {
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         return null;
132       },
133       builder: (field) => SelectionCard<ChoiceType>(
134             title: allChoices.name,
135             allChoices: allChoices,
136             initialValue: {
137               if (selectionViewModel.value != null)
138                 selectionViewModel.value!
139             },
140             onSelect: (selectedChoice) =>
141                 selectionViewModel.value = selectedChoice,
142             onDeselect: (selectedChoice) => selectionViewModel.value = null,
143             errorText: field.errorText,
144           ));
145 }
```

Listing 4: Die Maßnahmencharakteristika Selektionskarten werden ergänzt, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-3/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

Die anonyme Funktion, die als Argument dem Parameter `validator` übergeben wird (Z. 122-132), erstellt eine temporäre Menge, die den Wert des `selectionViewModel` enthält, wenn dieser nicht `null` ist, andernfalls ist sie eine leere Menge (Z. 123-125). Die `validator`-Funktion gibt eine Fehlermeldung zurück, sollte die Menge leer sein (Z. 127-129). Ist die Menge dagegen gefüllt, so gibt sie `null` zurück, um mitzuteilen, dass die Validierung erfolgreich war (Z. 131).

Der `errorText` wird im Konstruktor der Klasse `SelectionCard` übergeben (Listing. 5, Z. 29). Da er `null` sein darf, ist er mit dem Suffix `?` als Typ mit Null-Zulässigkeit gekennzeichnet (Z. 21).

Durch Einfügen einer `Column` zwischen der `Card` (Listing. 6, Z. 53) und dem ListTile (Z. 57) kann die visuelle Repräsentation der Selektionskarte in der Höhe erweitert werden. Sollte der `errorText` gesetzt sein (Z. 65), so erscheint unter dem Titel und dem Untertitel eine entsprechende Fehlermeldung (Z. 66-71).

```
19 final OnSelect<ChoiceType> onSelect;
20 final OnDeselect<ChoiceType> onDeselect;
21 final String? errorText;
22
23 SelectionCard(
24     {required this.title,
25     required Iterable<ChoiceType> initialValue,
26     required this.allChoices,
27     required this.onSelect,
28     required this.onDeselect,
29     this.errorText,
30     Key? key})
```

Listing 5: errorText wird der SelectionCard hinzugefügt, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-3/conditional_form/lib/widgets/selection_card.dart

```
53 return Card(
54   child: Column(
55     crossAxisAlignment: CrossAxisAlignment.start,
56     children: [
57       ListTile(
58         focusNode: focusNode,
59         title: Text(title),
60         subtitle: Text(
61             selectedChoices.map((c) => c.description).join(", ")),
62         trailing: const Icon(Icons.edit),
63         onTap: navigateToSelectionScreen,
64       ),
65       if (errorText != null)
66         Padding(
67           padding: const EdgeInsets.all(8.0),
68           child: Text(errorText!,
69               style:
70                   const TextStyle(fontSize: 12.0, color: Colors.red)),
71         )
72     ],
73   ),
74 );
```

Listing 6: errorText wird ausgegeben, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-3/conditional_form/lib/widgets/selection_card.dart

Oberhalb des vorhandenen `FloatingActionButton` wird nun ein weiterer eingefügt, der zum Speichern des Entwurfs mit der Funktion `saveDraftAndGoBackToOverviewScreen` genutzt werden soll (Listing. 7, Z. 207-216). Der ursprüngliche `FloatingActionButton` speichert nun ausschließlich dann, wenn die Maßnahme als „in Bearbeitung“ markiert ist oder alle Eingabefelder valide sind. Dazu nutzt er die Hilfsfunktion `inputsAreValidOrNotMarkedFinal` (Z. 222). Ist das der Fall, so folgt die Speicherung der Maßnahme mithilfe der bereits implementierten Funktion `saveRecord` (Z. 223). Diese funktioniert wie in den letzten Schritten, nur dass sie keinen Rückgabewert mehr hat (siehe Listing 12 in Anhang A auf Seite 12). Anschließend wird der Navigator erneut aufgefordert, zum Übersichtsbildschirm zurückzukehren (Z. 224). Sollte es allerdings zur Ausführung des `else`-Blocks führen (Z. 225-227), da die Maßnahme doch als „abgeschlossen“ markiert und nicht alle Eingabefelder valide waren, so erhält der Benutzer eine Fehlermeldung. Die neu implementierte Hilfsfunktion `showValidationError` wird dafür verwendet (Z. 226).

Auch der `WillPopScope` erhält die gleiche Fehlerbehandlung (Listing 8). Hier wird ebenfalls überprüft, ob die Maßnahme als „abgeschlossen“ markiert wurde und ob alle Eingabefelder valide sind (Z. 153). Falls ja, wird die Maßnahme direkt gespeichert Und ein Objekt des asynchronen Types `Future` zurückgegeben, welches direkt zu `true` evaluiert (Z. 155). Das

```
206 children: [
207   FloatingActionButton(
208     mini: true,
209     heroTag: 'save_draft_floating_action_button',
210     child: const Icon(Icons.paste, color: Colors.white),
211     backgroundColor: Colors.orange,
212     onPressed: saveDraftAndGoBackToOverviewScreen,
213   ),
214   const SizedBox(
215     height: 10,
216   ),
217   FloatingActionButton(
218     tooltip: saveMassnahmeTooltip,
219     heroTag: 'save_floating_action_button',
220     child: const Icon(Icons.check, color: Colors.white),
221     onPressed: () {
222       if (inputsAreValidOrNotMarkedFinal()) {
223         saveRecord();
224         Navigator.of(context).pop();
225       } else {
226         showValidationError();
227       }
228     },
229   )
230 ],
```

Listing 7: Die Maßnahmencharakteristika Selektionskarten werden ergänzt, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-3/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

führt dazu, dass dem Zurücknavigieren zum Übersichtsbildschirm zugestimmt wird. Sollte allerdings der `else`-Block ausgeführt werden, so erscheint erneut die entsprechende Fehlermeldung (Z. 157) und dieses Mal evaluiert das `Future`-Objekt zu `false`, um die Navigation zu unterbinden `158`.

```
151 body: WillPopScope(
152   onWillPop: () {
153     if (inputsAreValidOrNotMarkedFinal()) {
154       saveRecord();
155       return Future.value(true);
156     } else {
157       showValidationError();
158       return Future.value(false);
159     }
160   },
```

Listing 8: Die Maßnahmencharakteristika Selektionskarten werden ergänzt, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-3/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

Die Funktion `saveDraftAndGoBackToOverviewScreen` funktioniert ähnlich wie die nun ausgetauschte Funktion `saveRecord`. Sie zeigt dem Benutzer an, dass die Maßnahme im Entwurfsmodus gespeichert wird (Z. 23-26), speichert sie im Model ab (Z. 31), und navigiert zur letzten Route zurück (Z. 32), welcher der Übersichtsbildschirm ist. Einer der beiden Unterschiede ist, dass die Maßnahme zuvor umgebaut wird. Unerheblich dessen, welchen letzten Status sie aktuell besitzt, erhält sie den letzten Status `"in Bearbeitung"` (Z. 28-29). Der zweite der beiden Unterschiede ist, dass die Funktion nun keinen Rückgabewert hat, während `saveRecord` einen Wert vom Typ `Future<bool>` zurückgeben musste. Der Grund dafür ist, dass die Funktion nur noch über den Aktionsbutton zum Speichern der Maßnahme im Entwurfsmodus ausgelöst wird. Der FloatingActionButton setzt keinen Rückgabewert der ausgelösten Funktion voraus.

Die Hilfsfunktion `inputsAreValidOrNotMarkedFinal` überprüft zunächst, ob der letzte Status

```
22 void saveDraftAndGoBackToOverviewScreen() {
23   ScaffoldMessenger.of(context)
24     ..hideCurrentSnackBar()
25     ..showSnackBar(
26         const SnackBar(content: Text('Entwurf wird gespeichert ...')));
27
28   var draft = vm.model.rebuild((b) =>
29       b.letzteBearbeitung.letzterStatus = LetzterStatus.bearb.abbreviation);
30
31   model.putMassnahmeIfAbsent(draft);
32   Navigator.of(context).pop();
33 }
```

Listing 9: Die Maßnahmencharakteristika Selektionskarten werden ergänzt, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-3/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

ein anderer ist als „abgeschlossen“ (Listing. 10, Z. 71). Da in diesem Fall keine weiteren Überprüfungen notwendig sind, gibt die Funktion direkt `true` zurück (Z. 73). Andernfalls validiert das Formular die Eingabefelder (Z. 76). Dazu muss das Element vom Typ `Form` in den Vaterelementen gefunden werden. Genauer gesagt wird dessen `State`-Objekt benötigt. Der Zugriff auf das Element ist einfach, da es über einen `GlobalKey` registriert wurde. Über `formKey.currentState` kann das `State`-Objekt des Elements abgerufen werden (Z. 76). Die Funktion `validate()` führt dann alle Funktionen aus, die jeweils als Argument dem Parameter `validator` aller Kindelemente des Typs `FormField` übergeben wurden. Sollten alle `validator`-Funktionen `null` zurückgegeben haben – was bedeutet, dass keine Fehler bei der Validierung geschehen sind – so erfolgt die Rückgabe von `true` (Z. 77). Anderenfalls bleibt nur die Rückgabe von `false` übrig (Z. 80).

```
71 bool inputsAreValidOrNotMarkedFinal() {
72   if (vm.letzterStatus.value != LetzterStatus.fertig) {
73     return true;
74   }
75
76   if (formKey.currentState!.validate()) {
77     return true;
78   }
79
80   return false;
81 }
```

Listing 10: Die Maßnahmencharakteristika Selektionskarten werden ergänzt, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-3/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

Sollte es zu einem Fehler kommen, so zeigt die Hilfsfunktion `showValidationError` dem Benutzer die entsprechende Fehlermeldung an (Listing 11). Sie bietet ihm darüber hinaus an, über einen Button die Maßnahme direkt als Entwurf zu speichern. Das ist möglich, da die `SnackBar` (Z. 45) nicht nur die Anzeige von gewöhnlichem Text erlaubt, sondern von jedem beliebigen Widget. Zunächst kommt dazu das Widget `Row` zum Einsatz (Z. 46). Ähnlich wie das Widget `Column` erlaubt es Kinderelemente in einer Reihe aufzulisten. Im Gegensatz zur `Column` allerdings nun horizontal statt vertikal. Als letztes Element der `Row` wird der `ElevatedButton` verwendet. Genauso wie bereits der `FloatingActionButton` zum Speichern der Maßnahme im Entwurfsmodus verwendet nun auch dieser `ElevatedButton` die Funktion `saveDraftAndGoBackToOverviewScreen` (Z. 52).

```
44 void showValidationError() {
45   ScaffoldMessenger.of(context).showSnackBar(SnackBar(
46       content: Row(
47     children: [
48       Text(
49           'Fehler im Formular trotz Status "${LetzterStatus.fertig.description}"'),
50       const SizedBox(width: 4),
51       ElevatedButton(
52         onPressed: saveDraftAndGoBackToOverviewScreen,
53         child: Padding(
54           padding: const EdgeInsets.fromLTRB(4, 4, 8, 4),
55           child: Row(
56             children: const [
57               Icon(Icons.paste, color: Colors.white),
58               SizedBox(width: 4),
59               Text(
60                 "Entwurf speichern?",
61                 style: TextStyle(fontSize: 18.0, color: Colors.white),
62               ),
63             ],
64           ),
65         ),
66       ),
67     ],
68   )));
69 }
```

Listing 11: Die Maßnahmencharakteristika Selektionskarten werden ergänzt, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-3/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

Teil II

Anhang

A Schritt 3 Anhang

```
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 12: Die Maßnahmencharakteristika Selektionskarten werden ergänzt, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-3/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart