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

Hochschule Harz
Fachbereich Automatisierung und Informatik

Thema und Aufgabenstellung der Masterarbeit
MA AI 29/2021

für Herrn Alexander Johr

Entwicklung einer Formularanwendung mit Kompatibilitätsvalidierung der Einfach- und Mehrfachauswahl-Eingabefelder

Das Thünen-Institut für Ländliche Räume wertet Daten zu Maßnahmen auf landwirtschaftlich genutzten Flächen aus. Dafür müssen entsprechende Maßnahmen bundesweit mit Zeitbezug auswertbar sein und mit Attributen versehen werden. Um die Eingabe für die Wissenschaftler des Instituts zu beschleunigen und um fehlerhafte Eingaben zu minimieren, soll eine spezielle Formularanwendung entwickelt werden. Neben herkömmlichen Freitextfeldern beinhaltet das gewünschte Formular zum Großteil Eingabefelder für Einfach- und Mehrfachauswahl. Je nach Feld kann die Anzahl der Auswahloptionen mitunter zahlreich sein. Dem Nutzer sollen daher nur solche Auswahloptionen angeboten werden, die zusammen mit der zuvor getroffenen Auswahl sinnvoll sind.

Im Wesentlichen ergibt sich die Kompatibilität der Auswahloptionen aus der Bedingung, dass für dasselbe oder ein anderes Eingabefeld eine Auswahlmöglichkeit gewählt bzw. nicht gewählt wurde. Diese Bedingungen müssen durch Konjunktion und Disjunktion verknüpft werden können. In Sonderfällen muss ein Formularfeld jedoch auch die Konfiguration einer vom Standard abweichenden Bedingung ermöglichen. Wird dennoch versucht, eine deaktivierte Option zu selektieren, wäre eine Anzeige der inkompatiblen sowie der stattdessen notwendigen Auswahl ideal.

Die primäre Zielplattform der Anwendung ist das Desktop-Betriebssystem Microsoft Windows 10. Idealerweise ist die Formularanwendung auch auf weiteren Desktop-Plattformen sowie mobilen Endgeräten wie Android- und iOS-Smartphones und -Tablets lauffähig. Die Serialisierung der eingegebenen Daten genügt dem Institut zunächst als Ablage einer lokalen Datei im JSON-Format.

Die Masterarbeit umfasst folgende Teilaufgaben:

- Analyse der Anforderungen an die Formularanwendung
- Evaluation der angemessenen Technologie für die Implementierung
- Entwurf und Umsetzung der Übersichts- und Eingabeoberfläche
- Konzeption und Implementierung der Validierung der Eingabefelder
- Entwicklung von automatisierten Testfällen zur Qualitätskontrolle
- Bewertung der Implementierung und Vergleich mit den Wunschkriterien

Digital unterschrieben von Juergen K. Singer
o= Hochschule Harz, Hochschule fuer angewandte Wissenschaften, l= Wernigerode
Datum: 2021.03.23 12:30:26 MEZ

Prof. Jürgen Singer Ph.D.
1. Prüfer

Prof. Daniel Ackermann
2. Prüfer

Inhaltsverzeichnis

Abbildungsverzeichnis

Listingsverzeichnis

1 Anforderungen

Dieses Kapitel behandelt die Anforderungen

- Performance: Hohe Anzahl Eingabefelder
- nested formulars

Wunsch

- Alle Komponenten wiederverwendbar wie etwa Selection Caard nicht nur für Choices
-
- Strategy Entwurfsmuster für compute choice von viewModelType weil viewmodel nötig ist für nested formular
-

Teil I

Einleitung

2 Problemstellung

Das primäre Problem der Formular-Anwendung ist, dass sich die Auswahlfelder untereinander beeinflussen. Wird eine Option in einem Auswahlfeld selektiert, so werden die möglichen Auswahlfelder von potenziell jedem anderen Auswahlfeld dadurch manipuliert. Es muss eine Möglichkeit gefunden werden, die Abhängigkeiten in einer einfachen Art und Weise für jede Auswahloption zu hinterlegen und bei Bedarf abzurufen.

Das sekundäre Problem, welches sich vom primären Problem ableiten lässt, ist die Laufzeit-Geschwindigkeit. Wenn die Auswahl in einem Auswahlfeld die Auswahlmöglichkeiten in potenziell allen anderen Auswahlfeldern manipuliert, so könnte dies zu einer hohen Last beim erneuten Zeichnen der Oberfläche zur Folge haben. Wann immer der Nutzer eine Selektion tätigt, müsste das gesamte Formular neu gezeichnet werden, um sicherzustellen, das invalide Auswahloptionen gekennzeichnet werden. Bei einem Formular mit wenigen Auswahlfeldern wäre das kein Problem. Doch die nötigen Auswahlfelder für das eintragen von Maßnahmen des Europäische Landwirtschaftsfonds für die Entwicklung des ländlichen Raums (ELER) sind zahlreich. Ein automatisierter Integrationstest, welcher im Formular Daten einer beispielhaften Maßnahme einträgt, zählt zum Zeitpunkt der Erstellung dieser Arbeit bereits 58 aufgerufene Auswahlfelder und 107 darin selektierte Auswahloptionen. Das bedeutet, dass bei jedem dieser 107 Selektionen die 58 Auswahlfelder und all ihre Kinder neu gezeichnet werden müssten. Es könnten also Wartezeiten nach jedem Auswählen einer Option entstehen. Das Formular soll in Zukunft zudem noch erweitert und auch für die Eingabe ganz anderer Datensätze mit potenziell noch mehr Auswahlfeldern eingesetzt werden. Es ist also erforderlich, dass ein Mechanismus gefunden wird, der nur die Elemente neu zeichnet, die sich wirklich ändern.

Für Form wurde Flutter gewählt

3 Grundlagen

Für die Formular Anwendung wurde die Programmiersprache Dart und das Oberflächen Framework Flutter gewählt. Kapitel Kapitel einfügen erläutert die Entscheidungs-Grundlage dafür.

Nachfolgend soll auf Grundlagen der beiden Technologien eingegangen werden.

3.1 Flutter

Flutter ist ein Framework zur Entwicklung von Oberflächen von Google. Es unterstützt eine breite Anzahl an Ziel-System. Dazu gehören:

- Desktop:[1]
  - Windows:

[1] Vgl. Google LLC, *Desktop support for Flutter*.

    - Win32,
    - Universal Windows Platform,
  - macOS,
  - Linux,
- Mobile Endgeräte[2]:
  - Android,
  - iOS,
- und das Web[3].

Flutter ist inspiriert durch das Web-Framework react und deren Oberflächenelemente, die Components genannt werden[4]. Die visuellen Oberflächen-Elemente in Flutter werden dagegen Widgets genannt. „react“ „Components“ verfügen über einen Zustand – „State“ genannt – der bei Veränderung das neu zeichnen der visuellen Repräsentation erwirkt. Flutter unterscheidet allerdings zwischen zwei Arten von Widgets: denen, die einen Zustand pflegen – den „Stateful Widgets“ – und solchen, die keinen Zustand haben – den „Stateless Widgets“.

„Stateful Widgets“ pflegen einen Zustand, der mittels der Methode `setstate` gesetzt werden kann. Bei Aufrufen der Methode wird das gesamte Widget neu gezeichnet. Der Zustand selbst ist dabei im visuellen Baum als Vater der visuellen Elemente des Widgets verankert und bleibt erhalten, während die dazugehörigen Oberflächenelemente ausgetauscht werden.

„Stateless Widgets“ haben dagegen keinen solchen Mechanismus. Wie alle Widgets werden sie neu gezeichnet, wenn es durch das Framework angeordnet wurde. Das kann unter anderen der Fall sein, wenn das Widget zum ersten Mal in der Oberfläche auftaucht, oder das Vater-Element und damit alle Kinder Elemente neu gezeichnet werden.

„StatefulWidgets“ sind nur eine von vielen Möglichkeiten den Zustand des Programms zu verwalten. Die Formular-Anwendung verwendet ausschließlich `StatelessWidget`s, da die Verwaltung des Zustands über das sogenannte BloC Pattern umgesetzt ist. Mehr dazu im Kapitel `Kapitel einfügen`.

3.2 Dart

Flutter-Anwendungen werden in der Programmiersprache Dart geschrieben. Nachfolgend soll auf eine Reihe von Besonderheiten von Dart im Vergleich zu anderen objektorientierten Programmiersprachen eingegangen werden.

Dart ist eine Hochsprache, die hauptsächlich für die Entwicklung von Oberflächen entwickelt wurde, sich jedoch ebenso dazu eignet, Programme für das Back-End zu entwickeln.

Ein Hauptaspekt bei dem Design der Sprache ist die Produktivität des Entwicklers. Mechanismen wie das „hot reload“ verkürzen die Entwicklungszyklen erheblich. Das „hot reload“ ermöglicht es, während eine Anwendung im Debug-Modus ausgeführt wird, Änderungen an dessen Quellcode vorzunehmen. Daraufhin werden nur die Teile der laufenden Applikation aktualisiert, die tatsächlich verändert wurden. Währenddessen bleibt die Anwendung in der gleichen Ansicht, anstatt zum Hauptbildschirm zurückgesetzt werden, von der aus der Entwickler erneut zur gewünschten Ansicht zurücknavigieren müsste.

[2] Vgl. Google LLC, *Flutter - Beautiful native apps in record time*.
[3] Vgl. Google LLC, *Web support for Flutter*.
[4] Vgl. Google LLC, *Flutter - Introduction to widgets*.

3.2.1 AOT und JIT

Nicht nur für die reibungslose Entwicklung sondern auch für das Laufzeitverhalten der finalen Applikation wurde die Sprache optimiert. Für die Ziel-Architektouren ARM32, ARM64 und x86_64 wird Dart in Maschinencode kompiliert. \url {https://dart.dev/overview##native-platform}

Dementsprechend kommt während der Entwicklung eine virtuelle Maschine - die Dart VM - geschiet über Just-in-time-Kompilierung (JIT) zum Einsatz. Für die Kompilierung in Maschinencode wird dagegen Ahead-of-time-Kompilierung (AOT) eingesetz.

tree shaking Für die Minimierung der Dateigröße des resultierenden Kompilats wird das sogenannte „tree shaking“ eingesetzt. Das Hauptprogramm importiert über das Schlüsselwort `import` Funktionalitäten aus weiteren .dart-Dateien oder sogar ganzen Bibliotheken. Diese Dateien importieren wieder Weitere. Dadurch wird ein Baum aufgespannt. Das „tree shaking“ identifiziert, welche Funktionalitäten tatsächlich vom Programm verwendet werden und welche nicht. Dies bringt aber einer wichtige Einschränkung mit sich. Die Metaprogrammierung (der Zugriff auf sprachinterne Eigenschaften, wie etwa Klassen und ihre Attribute) ist damit stark eingeschränkt.

Meta-Programmierung Bei der Kompilierung werden die Original-Bezeichner durch Symbole ersetzt, welche minimalen Speicher-Bedarf haben. Aber nicht nur das, denn durch das „tree shaking“ werden auch etwaige Eigenschaften und Funktionalitäten entfernt, die nicht verwendet werden. Die sogenannte „Reflexion“ oder „Introspektion“ versucht auf solche Meta-Informationen während der Laufzeit zuzugreifen. Da die Eigenschaften aber nicht mehr verfügbar sind, ist „Reflexion“ nicht anwendbar. Dart greift daher auf eine andere Variante der Meta-Programmierung zurück: die Quellcode Generierung.

Quellcode-Generierung Das Package „source_gen“ erlaubt das Auslesen der Meta-Informationen und ermöglicht das Generieren von Quellcode, der von diesen Eigenschaften abgeleitet werden kann. So verwendet beispielsweise das Package „built_value“ die Quellcode-Generierung. Zunächst werden Eigenschaften wie Klassennamen, Instanzvariablen mit ihren Bezeichnern und Datentypen gelesen. Die Eigenschaften können dann genutzt werden, um unveränderliche Werte-Typen und dazugehörige sogenannte „Builder“-Objekte des Erbauer-Entwurfsmusters, sowie Funktionen zum Serialisieren und Deserialisieren von Objekten zu generieren. Referenzen

3.2.2 Set und Map Literale

Dart erlaubt es Listen (`List`), Mengen (`Set`) und Hashtabellen (`Map`) als sogenannte Literale zu deklarieren. Ein Literal ist die textuelle Repräsentation eines Wertes eines speziellen Datentyps. Beispielsweise ist `"Text"` ein String-Literal für eine Zeichenkette mit den Elementen „T“, „e“, „x“, „t“. So ist auch \{"Text"\} ein Literal für eine Menge (`Set`). Eine Menge mit den gleichen Werten könnte genauso auch wie in Listing 1 erstellt werden.

```
1 var menge = Set();
2 menge.add("Text");
```

Listing 1: Ein Set, Quelle: Eigenes Listing

Es entfällt also die Instanziierung einer Liste, einer Menge oder einer Hashtabelle über den Klassennamen und der darauffolgenden Zuweisung der einzelnen Werte. Stattdessen startet das `Set` und `Map` Literal mit einer öffnenden geschweiften Klammer und endet mit einer schließenden geschweiften Klammer. Innerhalb der Klammern werden die Werte im Fall eines Sets mit `,` getrennt nacheinander aufgeführt (`\{1,2\}`). Im Fall einer Map werden der Schlüssel und der Wert durch einen `:` voneinander getrennt und die Schlüsse-Wertepaare wiederum durch `,` getrennt nacheinander aufgelistet (`\{1: "erster Wert",2:"zweiter Wert"\}`). Eine Liste wiederum wird mit eckigen Klammern geöffnet und geschlossen. Die Werte werden erneut mit, getrennt voneinander angegeben (`[1,2]`).

Collection for Dart erlaubt es Schleifen innerhalb von Listen-, Mengen- und Hashtabellen-Literalen zu verwenden. Dabei darf die Schleife jedoch keinen Schleifen-Körper besitzen. Ledig der Schleifen-Kopf wird dazu im Literal geschrieben. Darauf folgt der Wert, der bei jedem Schleifendurchlauf hinzugefügt werden soll. Dabei kann der Wert von der Schleifenvariable genutzt oder davon abgeleitet werden. Listing 2 geht beispielsweise durch die Liste von der Temperatur-Angaben 97.7,105.8, die in Fahrenheit gelistet sind. Für jeden Schleifendurchlauf wird die Schleifen-Variable f mit der entsprechenden Formel in Grad Celsius umgewandelt. Das Ergebnis ist somit äquivalent mit dem `Set`-Literal `{36.5, 38.5, 41}`.

```
1 var gradCelsiusTemperaturen = {
2     for (var f in [97.7, 101.3, 105.8])
3         (f - 32) * 5 / 9
4 };
```

Listing 2: Collection-for in einer Menge, Quelle: Eigenes Listing

Gleiches gilt für Hashtabellen. Hierbei wird ein Schlüssel-Werte-Paar übergeben. Links vom einem `:` ist der Schlüssel und rechts davon der Wert. In Listing 3 wird durch die gleiche Liste von Temperaturen in Fahrenheit iteriert. Für jede Schleifen variable f wird für das resultierende Schlüsselwörter paar das Ergebnis in Grad Celsius als Schlüssel und das Ergebnis als Wert eingetragen. Das Ergebnis von `celsiusUndFahrenheit` ist dementsprechend eine `Map` mit dem Wert: `{36.5: 97.7, 38.5: 101.3, 41: 105.8}`

```
1 var celsiusUndFahrenheit = {
2     for (var f in [97.7, 101.3, 105.8])
3         (f - 32) * 5 / 9 :  f
4 };
```

Listing 3: Collection-for in einer Hashtabelle, Quelle: Eigenes Listing

Collection-if Neben dem Collection-for ist auch die Nutzung von Fallunterscheidungen in Kollektionen erlaubt. Vor dem Wert, der in die Kollektion aufgenommen werden soll oder nicht, kann das Schlüsselwort `if` mit einer darauffolgenden Bedingung in Klammern gesetzt werden. Listing 4 iteriert durch eine Anzahl von Temperaturen in Grad Celsius. Nur in dem Fall, dass die Temperatur der Schleifen-Variable `c` größer oder gleich 38,5 ist, wird die Temperatur der Liste zugefügt. Das Ergebnis der Liste `fieberTemperaturen` Ergibt also `[38.5, 41]`.

```dart
var fieberTemperaturen = [
    for (var c in [36.5, 38.5, 41])
        if (c >= 38.5) c
];
```

Listing 4: Collection-if in einer Liste, Quelle: Eigenes Listing

3.2.3 Typen ohne NULL-Zulässigkeit

Im Vergleich zu vielen anderen Programmiersprachen wie beispielsweise in Java wird in Dart zwischen gewöhnlichen Typen und nullable Typen unterschieden. In Sprachen wie Beispielsweise Java ist es nur bei atomaren Datentypen wie `int` und `float` vorgeschrieben einen Wert anzugeben. `null` ist bei diesen primitiven Datentypen nicht als Wert erlaubt. Doch nicht atomare Datentypen erlauben immer die Angabe von `null` als Wert. `null` drückt dabei immer das Nichtvorhandensein von Daten aus. Ab Dart 2.12 kann allen Datentypen standardmäßig kein Null-Wert zugewiesen werden. Das hat den Vorteil, dass der Compiler sich darauf verlassen kann, das eine Variable niemals den Wert `null` haben kann. Das ist besonders dann nützlich, wenn auf Einem Objekt eine Methode aufgerufen wird. Ist die Referenz das Objekt ist in Wahrheit `null` so gibt es erst zur Laufzeit einen Fehler, da die Methode auf der Referenz `null` nicht aufgerufen werden kann. Damit ein Laufzeitfehler geworfen werden kann, muss vor jedem Aufruf einer Methode auf einer Referenz überprüft werden, ob die Referenzen nicht `null` sind. Würde diese Überprüfung nicht stattfinden, so könnte kein Laufzeitfehler geworfen werden und das Programm würde ohne Fehlermeldung abstürzen. Handelt es sich allerdings um eine Referenz, die niemals den Wert `null` annehmen kann, so kann der Compiler die Überprüfung auf Null-Werte für diese Referenzen überspringen. Damit erhört sich zusätzlich die Ausführungsgeschwindigkeit, da die Überprüfung Zeit in Anspruch nimmt. Vor allem aber ist es vorteilhaft für den Entwickler, da der Compiler Fehlermeldungen und Warnungen mitteilen kann, wenn Operationen auf Variablen mit potenziellen Null-Werten verwendet werden. Die Abwesenheit von Daten ist jedoch bei der Entwicklung sehr wichtig. Nicht alle Variablen können immer einen Wert haben. Aus diesem Grund gibt es in Dart auch die Typen, die auch Null-Werte zulassen. Allerdings gelten besondere Regeln für diese Typen.

3.2.4 Typen mit Null-Zulässigkeit

Wird in Dart hinter einem Typen ein `?` angegeben, so kann die Variable nicht nur Werte annehmen, die dieser Datentyp zulässt sondern zusätzlich auch noch den Wert `null`. Methoden auf Objekten mit Null-Zulässigkeit aufzurufen ist nicht ohne weiteres möglich.

Im Listing 5 wird versucht die auf die Variable `fahrenheitTemperature` den Operator `-` anzuwenden um sie mit `32` zu subtrahieren. Der Compiler liefert jedoch einen Fehler, da der Wert der Variable `null` sein kann, wie die Notation `int?` anzeigt. Solange nicht feststeht, dass die Variable zur Laufzeit tatsächlich nicht `null` ist, kann das Programm nicht kompiliert werden.

```dart
void printTemperatureInCelsius(int? fahrenheitTemperature) {
    print((fahrenheitTemperature - 32) * 5 / 9);
}
```

Listing 5: Collection-if in einer Liste, Quelle: Eigenes Listing

Zu diesem Zweck macht Dart von der sogenannten Type Promotion - deutsch Typ Be-

förderung - gebrauch. Mithilfe einer Fallunterscheidung kann vor Anwenden der Operation nachgesehen werden, ob der Wert der Variable nicht `null` ist. Innerhalb des Körpers der Fallunterscheidung wird der Typ der Variable automatisch in einen Typ ohne Null-Zulässigkeit befördert. Der Code in Listing 6 lässt sich daher wieder kompilieren.

```
1 void printTemperatureInCelsius(int? temperature) {
2   if (temperature != null) {
3     print((temperature - 32) * 5 / 9);
4   }
5 }
```

Listing 6: Collection-if in einer Liste, Quelle: Eigenes Listing

Eine Besonderheit stellen dabei allerdings Instanzvariablen dar. In Dart wird syntaktisch nicht zwischen dem Aufruf einer Getter-Methode oder einer Instanzvariable unterschieden. In Listing könnte sich hinter den Aufrufen von `temperature` in den Zeilen 6 und 7 die Instanzvariable verbergen, die in Zeile 2 deklariert ist.

```
1  class Patient {
2    num? temperature;
3    Patient({this.temperature});
4
5    void printTemperatureInCelsius() {
6      if (temperature != null) {
7        print((temperature - 32) * 5 / 9);
8      }
9    }
10 }
```

Listing 7: Collection-if in einer Liste, Quelle: Eigenes Listing

Genauso könnte es aber auch sein, das eine Klasse von Patient erbt und das Feld `temperature` mit einer gleichnamigen Getter-Methode überschreibt. Auch wenn es sehr unwahrscheinlich ist, könnte es trotzdem vorkommen, dass der Aufruf von `temperature` in Zeile 6 einen Wert zurückgibt, der nicht `null` ist und der darauffolgende Aufruf in Zeile 7 `null` liefert. So provoziert es die Klasse `UnusualPatient` im Listing 8. Beim ersten Aufruf von `temperature` wird die Zähl-Variable `counter` von `0` auf `1` erhöht. Die Abfrage, ob es sich bei dem Wert von Counter um eine ungerade Zahl handelt ist erfolgreich (Z. 6), weshalb mit `97,7` ein valider Wert zurückgegeben wird. Beim zweiten Aufruf erhöht sich `counter` allerdings auf `2`. Die gleiche Abfrage schlägt dieses Mal fehl. Deshalb liefert die Getter-Methode nun `null` (Z. 9). Ein solches Szenario ist schon sehr unwahrscheinlich, doch die Typ-Überprüfung des Compilers arbeitet mit Beweisen. Im Fall von Instanzvariablen kann nicht bewiesen werden, das zur Laufzeit ein solcher Fall ausgeschlossen werden kann.

Sollte sich der Entwickler sicher sein, dass die Variable nicht `null` sein kann, so kann er mit einem nachgestellten `!` erzwingen, dass die Variable als nicht `null` angesehen wird (Listing. Z. 3). Sollte es dann dennoch passieren, dass die Variable `null` ist, so wird eine Fehlermeldung beim Aufruf der Variable geworfen.

Eine noch sicherere Variante ist es, die Instanzvariable zuvor in eine lokale Variable zu speichern (Listing. 10 Z. 2). Die lokale Variable hat keine Möglichkeit zwischen den zwei Aufrufen einen unterschiedlichen Wert anzunehmen. Somit kann auch das Suffix `!` weggelassen werden (Z. 4).

```dart
class UnusualPatient extends Patient {
  int counter = 0;

  num? get temperature {
    counter++;
    if (counter.isOdd) {
      return 97.7;
    } else {
      return null;
    }
  }
}
```

Listing 8: Collection-if in einer Liste, Quelle: Eigenes Listing

```dart
void printTemperatureInCelsius() {
  if (temperature != null) {
    print((temperature! - 32) * 5 / 9);
  }
}
```

Listing 9: Collection-if in einer Liste, Quelle: Eigenes Listing

3.2.5 Asynchrone Programmierung

Wird auf eine externe Ressource zugegriffen - wie zum Beispiel das Abrufen einer Information von einem Webserver, oder das Lesen einer Datei im lokalen Dateisystem - so handelt es sich um asynchrone Operationen.

Im Sprachkern stellt Dart Schlüsselwörter und Datentypen für die asynchrone Programmierung bereit. Das sind unter anderem die Datentypen `Future` und `Stream` sowie die Schlüsselwörter `async` und `await`.

Future Ein `Future`-Objekt repräsentiert einen potenziellen einmaligen Wert, der in der erst in der Zukunft bereit steht. Er gleicht damit dem sogenannten `Promise` - deutsch Versprechen – in JavaScript. \url {https://developer.mozilla.org/de/docs/orphaned/Web/JavaScript/Reference

Das Listing 11 zeigt mit dem Lesen einer Datei ein Beispiel für den Aufruf einer asynchronen Operation.

Anders als erwartet, befindet sich in der Variable `fileContent` in Wahrheit kein Text mit dem Inhalt der Datei. Stattdessen hat die Variable den Datentyp `Future<String>` und ist lediglich ein sogenannter „Handle“ - deutsch Referenzwert - für das potenzielle und zukünftige Ergebnis der Operation.

Mit der Übergabe einer Funktion, die bei Vollendung der Operation aufgerufen wird, kann der Wert ausgewertet werden. Man nennt diese Operation auch „Callback-Funktion“ - deutsch Rückruffunktion. Listing 12 zeigt, wie auf den Dateiinhalt zugegriffen werden kann. Über die Methode `then` wird eine Funktion übergeben, die genau einen Parameter hat. In diesem Parameter wird der Text der gelesenen Datei bei Vollendung der Operation übergeben.

Der Einsatz von „Callback-Funktionen“ kann den Quellcode stark verkomplizieren. Man spricht von der sogenannten „callback hell“ - deutsch Rückruffunktionen-Hölle -, wenn

```
void printTemperatureInCelsius() {
  num? temperature = this.temperature;
  if (temperature != null) {
    print((temperature - 32) * 5 / 9);
  }
}
```

Listing 10: Collection-if in einer Liste, Quelle: Eigenes Listing

```
var fileContent = file.readAsString();
```

Listing 11: Collection-if in einer Liste, Quelle: Eigenes Listing

solche „Callback-Funktionen“ über etliche Level hinweg ineinander verschachtelt sind.

Um genau das zu verhindern, existieren in Dart die Schlüsselwörter `async` und `await`. Genauso heißen sie auch in anderen Sprachen wie etwa C# ab Version 4.5 und JavaScript ab Version ES8. Referenz

Listing 13 zeigt, dass durch das Anwenden des Schlüsselwortes `await` vor der Operation `file.readAsString` dafür sorgt, dass der zukünftige Wert direkt in `fileContent` gespeichert wird. Ganz ohne „Callback-Funktion“ kann der Dateiinhalt in der darauffolgenden Zeile ausgegeben werden.

Doch jede Funktion, die auf andere Funktionsaufrufe wartet, muss selbst als asynchron gekennzeichnet werden. Dazu dient das `async` Schlüsselwort vor Beginn des Methoden-Körpers.

Streams „Streams“ liefern nicht nur einen Wert – wie im Fall eines `Future` – sondern eine Serie von Werten, die in der Zukunft geliefert werden. Listing 14 zeigt wie auf einen solchen Stream gehorcht werden kann. `countStream` liefert jede Sekunde einen neuen Wert, nämlich die aktuelle Sekunde - von 0 beginnent. Mit `countStream.listen` kann eine Funktion übergeben werden, die immer dann ausgeführt wird, wenn dem `countStream` ein neuer Wert hinzugefügt wurde. Der erste Parameter ist dabei der hinzugefügte Wert.

Es wird zwischen zwei Arten von Streams unterschieden. Solche, die genau einen Empfänger haben - „singe subscription streams“ - und solche, die beliebig viele Empfänger haben können - „broadcast streams“.

Für die Formular Anwendung sind ausschließlich „broadcast streams“ zu berücksichtigen. Die Streams sollen verwendet werden, um Änderungen in der Eingabemaske zu behandeln. Die Oberflächenelemente horchen auf diese Änderungen. Teile der Oberfläche und damit die Oberflächenelemente, welche auf die Streams horchen, werden immer wieder neu gezeichnet. Dabei werden die Elemente entfernt und durch neu konstruierte ersetzt. Damit melden sich immer wieder Zuhörer vom „Stream“ ab und neue Elemente melden sich an. Daher kommen nur „broadcast streams“ infrage.

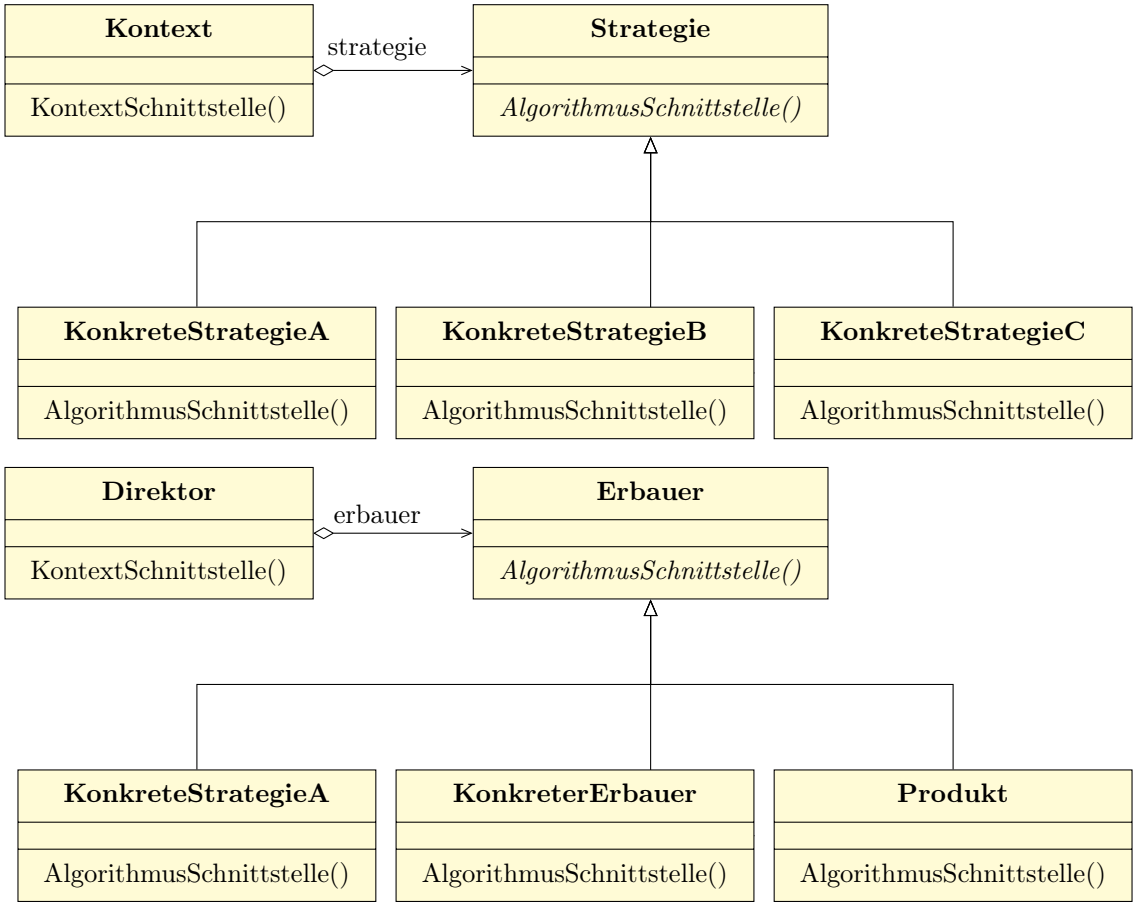
```
fileContent.then((text) {
  print("Der Datei-Inhalt ist: $text");
});
```

Listing 12: Collection-if in einer Liste, Quelle: Eigenes Listing

```
printFileContent() async {
  var fileContent = await file.readAsString();
  print("Der Datei-Inhalt ist: $fileContent");
}
```

Listing 13: Collection-if in einer Liste, Quelle: Eigenes Listing

3.3 Entwurfsmuster



```
var countStream = Stream<num>.periodic(const Duration(seconds: 1), (count) {
    return count;
  });

  countStream.listen((count) {
    print("Gezählte Sekunden: $count");
  });
```

Listing 14: Collection-if in einer Liste, Quelle: Eigenes Listing

4 Technologie Auswahl

Dieses Kapitel behandelt die Auswahl der Frontend-Technologie für die Umsetzung der Formular-Anwendung. Dazu werden im ersten Schritt die dafür in Frage kommenden Technologien identifiziert. Anschließend wird der Trend der Popularität dieser Technologien miteinander verglichen. Die daraus resultierenden Kandidaten sollen dann detaillierter untersucht werden. In Hinblick auf die Anforderungen an die Formular-Anwendung soll dabei die angemessenste Frontend-Technologie ausgewählt werden.

4.1 Trendanalyse

Zwei Quellen wurden für die Analyse der Technologie-Trends ausgewählt: die Ergebnisse der jährlichen Stack Overflow Umfragen und das Such-Interesse von Google Trends.

Stack Overflow Umfrage Die Internet-Plattform Stack Overflow richtet sich an Softwareentwickler und bietet ihren Nutzern die Möglichkeiten, Fragen zu stellen, Antworten einzustellen und Antworten anderer Nutzer auf- und abzuwerten. Besonders für Fehlermeldungen, die häufig während der Softwareentwicklung auftreten, findet man auf dieser Plattform rasch die Erklärung und den Lösungsvorschlag gleich mit. Dadurch lässt sich auch die Herkunft des Domain-Namens herleiten:

> We named it Stack Overflow, after a common type of bug that causes software to crash – plus, the domain name stackoverflow.com happened to be available.
> - Joel Spolsky, Mitgründer von Stack Overflow [5]

Aufgrund des Erfolgsrezepts von Stack Overflow ist die Plattform kaum einem Softwareentwickler unbekannt. Dementsprechend nehmen auch jährlich tausende Entwickler an den von Stack Overflow herausgegebenen Umfragen teil. Seit 2013 beinhalten die Umfragen auch die Angabe der aktuell genutzten und in Zukunft gewünschten Frontend-Technologien. Stackoverflow erstellt aus diesen gesammelten Daten Auswertungen und Übersichten. Doch gleichzeitig werden die zugrundeliegenden Daten veröffentlicht. [6]

Um den Trend der Beliebtheit der Frontend-Technologien aufzuzeigen, wurde ein Jupyter Notebook erstellt. Es transformiert die Daten in ein einheitliches Format, da die Umfrageergebnisse von Jahr zu Jahr in einer unterschiedlichen Struktur abgelegt wurden. Anschließend erstellt es Diagramme, die im Folgenden analysiert werden. Das Jupyter Notebook ist im Anhang zu finden.

Google Trends Suchanfragen die an die Suchmaschine Google abgesetzt werden, lassen sich über den Dienst Google Trends als Trenddiagramm visualisieren. Um das relative Such-Interesse abzubilden, werden die Ergebnisse normalisiert, um die Ergebnisse auf einer Skala von 0 bis 100 darstellen zu können. [7]

> Google Trends ist keine wissenschaftliche Umfrage und sollte nicht mit Umfragedaten verwechselt werden. Es spiegelt lediglich das Suchinteresse an bestimmten Themen wider. [8]

[5] Spolsky, „How Hard Could It Be?: The Unproven Path“
[6] Stack Exchange, Inc., *Stack Overflow Insights - Developer Hiring, Marketing, and User Research*
[7] Vgl. Google LLC, *Häufig gestellte Fragen zu Google Trends-Daten - Google Trends-Hilfe*
[8] Google LLC, *Häufig gestellte Fragen zu Google Trends-Daten - Google Trends-Hilfe*

Genau aus diesem Grund wird Google Trends im Folgenden lediglich zum Abgleich der Ergebnisse der Stack Overflow Umfrage eingesetzt.

4.1.1 Frameworks mit geringer Relevanz

NativeScript, Sencha (bzw. Sencha Touch) und Appcelerator spielen in den Umfrageergebnissen eine untergeordnete Rolle. Dies ist in den aufsummierten Stimmen von 2013 bis 2020 für alle in der Umfrage auftauchenden Frontend-Technologien zu sehen (Abb. **??**).

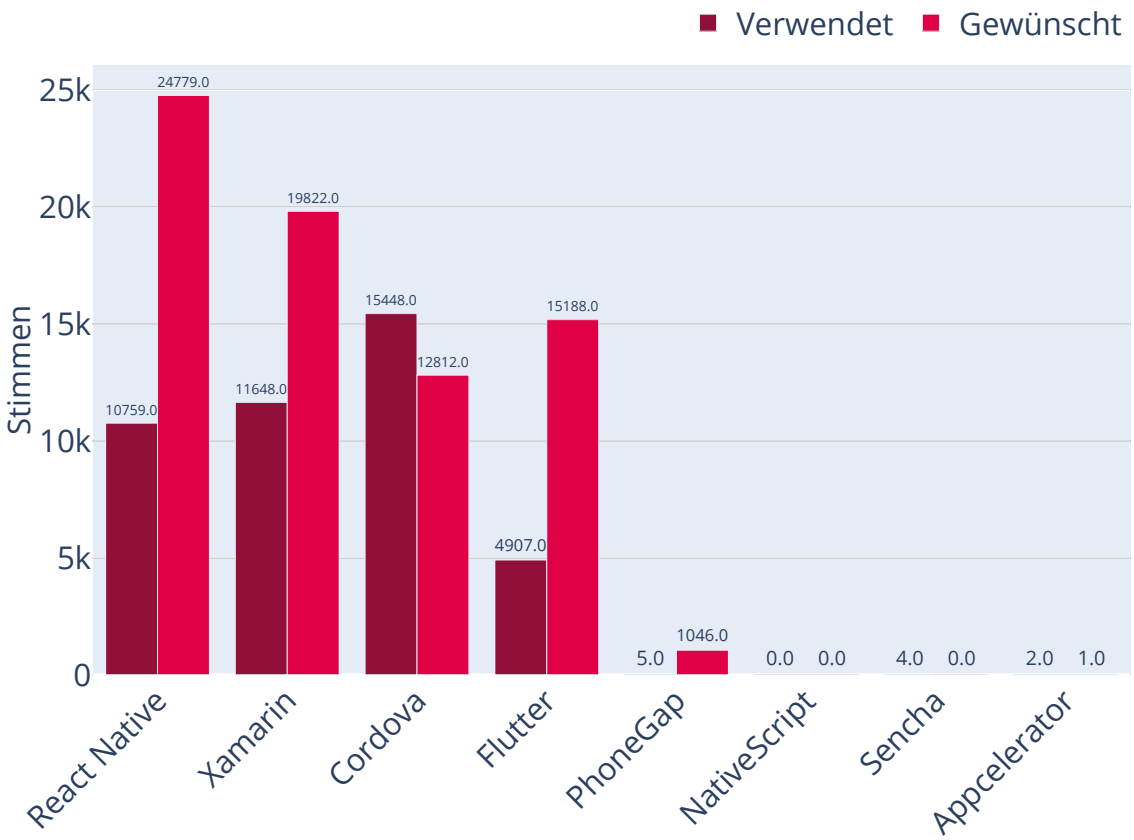


Abbildung 1: Summe der Stimmen der Stack Overflow Umfrage von 2013 bis 2020, Quelle: Eigene Abbildung, Notebook: `Charts/StackOverflowUmfrage/StackOverflowUmfrage.ipynb`, Daten-Quelle: FEHLT!

Auch das Suchinteresse auf Google ist für diese Frameworks äußerst gering. In Abbildung 2 werden NativeScript, Sencha, Appcelerator und auch Adobe PhoneGap mit Apache Cordova für das relative Suchinteresse verglichen.

Verwandte Technologien zu Apache Cordova Das Ionic Framework taucht in den Ergebnissen der Stack Overflow Umfragen nicht auf. Ein Grund dafür könnte sein, dass es auf Apache Cordova aufbaut[10], welches bereits in den Ergebnissen vorkommt. Adobe PhoneGap taucht zwar in den Ergebnissen von 2013 mit 1043 Stimmen auf (Siehe Abbildung 3), verliert jedoch in den Folgejahren mit weniger als 10 Stimmen abrubt an Relevanz. Das stimmt nicht mit dem Suchinteresse auf Google überein, da es dort ab 2013 sogar steigt, wie in Abbildung 2 zu sehen ist. 2013 existierte PhoneGap noch als extra Mehrfachauswahlfeld in den Daten, während es ab 2014 nur noch in dem Feld für die sonstigen Freitext Angaben auftaucht [11]. Auch Adobe PhoneGap baut auf Apache Cordova auf[12]. Für diese

[10] Lynch, *The Last Word on Cordova and PhoneGap*
[11] Vgl. Stack Exchange, Inc., *Stack Overflow Insights - Developer Hiring, Marketing, and User Research*
[12] Vgl. Adobe Inc., *FAQ | PhoneGap Docs*

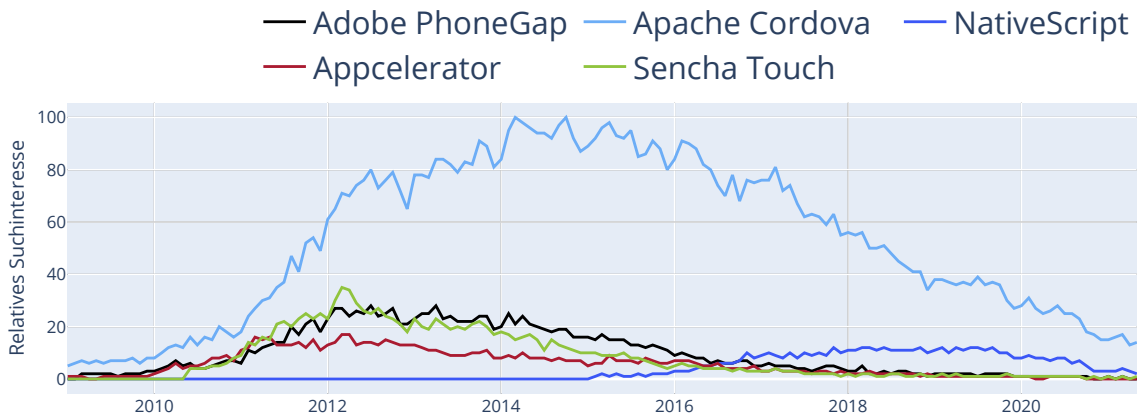


Abbildung 2: Suchinteresse der Frameworks mit geringer Relevanz, Quelle: Eigene Abbildung, Notebook: `Charts/GoogleTrends/GoogleTrends.ipynb`, Daten-Quelle: Google Trends[9]

Auswertung spielen diese verwandten Technologien eine untergeordnete Rolle, da sie auch in den Google Trends weit hinter Apache Cordova zurückbleiben (Abb. 2).

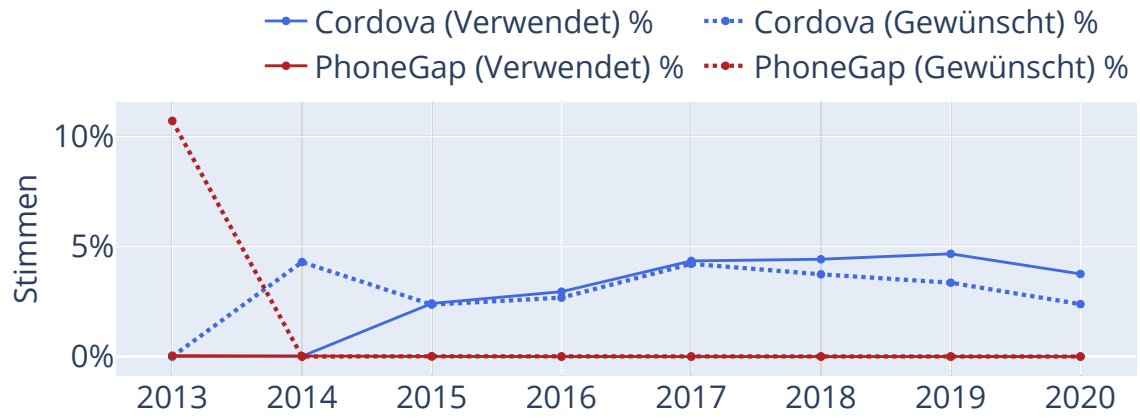


Abbildung 3: Stimmen für Cordova und PhoneGap 2013 bis 2020, Quelle: Eigene Abbildung, Notebook: `Charts/StackOverflowUmfrage/StackOverflowUmfrage.ipynb`, Daten-Quelle: FEHLT!

Am Beispiel von Adobe PhoneGap wird deutlich, wie wichtig es ist, auf eine Technologie zu setzen, die weit verbreitet ist. Im schlimmsten Fall wird die Technologie sogar vom Betreiber aufgrund zu geringer Nutzung komplett eingestellt, wie es bereits bei PhoneGap geschehen ist. Adobe gab am 11. August 2020 bekannt, dass die Entwicklung an PhoneGap eingestellt wird und empfiehlt die Migration hin zu Apache Cordova.[13]

4.1.2 Frameworks mit sinkender Relevanz

Die Technologien Xamarin und Cordova zeigen bereits einen abfallenden Trend, wie in Abbildung 4 ersichtlich ist. Im Fall von Xamarin gibt es immerhin mehr Entwickler, die sich wünschen, mit dem Framework zu arbeiten, als Entwickler, die tatsächlich mit Xamarin arbeiten. Cordova scheint in diesem Hinblick dagegen eher unbeliebt: Es gibt mehr Entwickler, die mit Cordova arbeiten, als tatsächlich damit arbeiten wollen.

In Abbildung 5 ist noch einmal zu sehen, dass Google Trends die Erkenntnisse aus der Stack Overflow Umfrage reflektiert und es wird auch sichtbar, welche beiden Technologien möglicherweise der Grund für den Rückgang von Xamarin und Cordova sind.

[13] Vgl. Adobe Inc., *Update for Customers Using PhoneGap and PhoneGap Build*

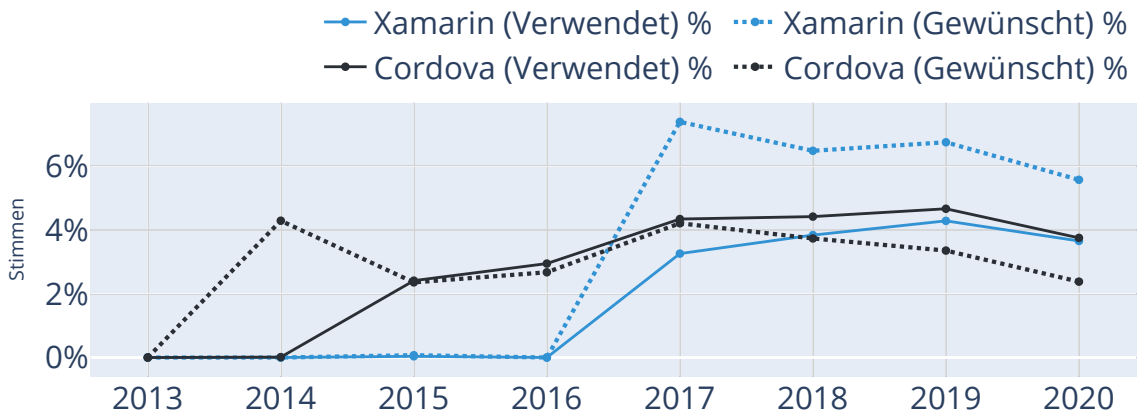


Abbildung 4: Stimmen für Xamarin und Cordova 2013 bis 2020, Quelle: Eigene Abbildung, Notebook: `Charts/StackOverflowUmfrage/StackOverflowUmfrage.ipynb`, Daten-Quelle: FEHLT!

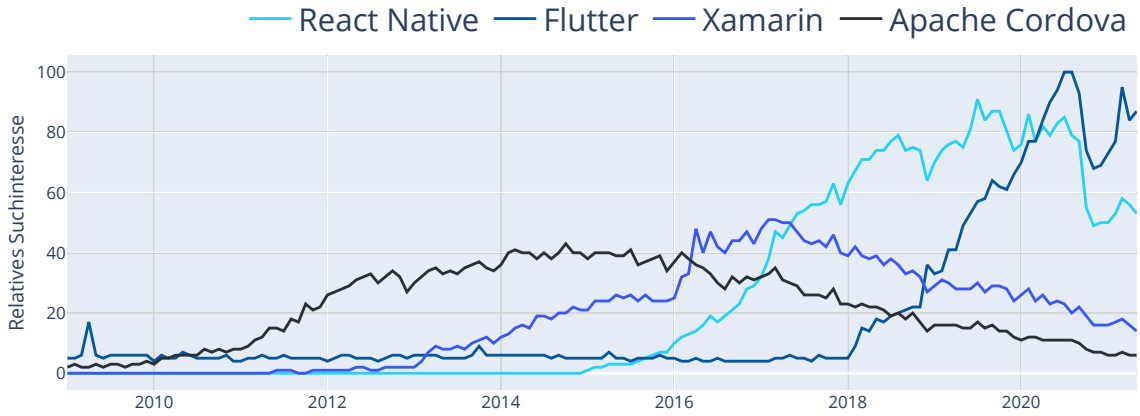


Abbildung 5: Suchinteresse sinkende und steigende Relevanz, Quelle: Eigene Abbildung, Notebook: `Charts/StackOverflowUmfrage/StackOverflowUmfrage.ipynb`, Daten-Quelle: FEHLT!

4.1.3 Frameworks mit steigender Relevanz

Besser ist es, auf Technologien zu setzen, die noch einen steigenden Trend der Verbreitung und Beliebtheit zeigen. In Abbildung 6 wird sichtbar, dass es sich dabei um Flutter und immerhin im Hinblick auf die Verbreitung auch für React Native handelt. Ungünstigerweise wird React Native in der Stack Overflow Umfrage erst seit 2018 als tatsächliches Framework abgefragt. Vorher erschien lediglich das Framework React, welches sich nicht für den Vergleich der Cross-PlatformFrameworks eignet, da es sich um ein reines Web-Framework handelt. Doch auch die Ergebnisse von Google Trends zeigen einen ähnlichen Verlauf für die Jahre 2019 und 2020 (Abb. 5).

Im Vergleich von dem Jahr 2019 mit 2020 wird sichbar, dass die Zahl der Entwickler, die sich wünschen, mit React Native zu arbeiten, gesunken ist. Dennoch ist die Anzahl der Entwickler, die mit React Native arbeiten möchten noch weit höher, als die der Entwickler, die tatsächlich mit React Native arbeiten.

Es ist möglich, dass der abfallende Trend daran liegt, dass die Zahl der Entwickler, die mit Flutter arbeiten möchten im selben Jahr gestiegen ist. React Native hat im Vergleich zu Flutter jedoch noch immer mehr aktive Entwickler und die Tendenz ist steigend. Doch die Anzahl der aktiven Flutter Entwickler zeigt einen noch stärker steigenden Trend. So könnte es sein, dass die Zahl der Flutter Entwickler die der React Native Entwickler in einem der nächsten Jahre überholt. Im Such-Interesse hat sich diese Entwicklung bereits vollzogen (Abb. 5).

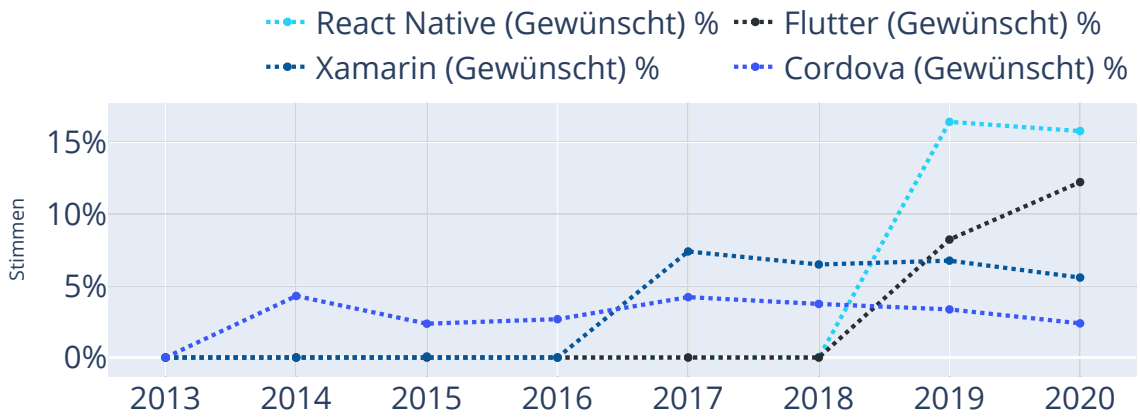


Abbildung 6: Stimmen für React Native und Flutter 2013 bis 2020, Quelle: Eigene Abbildung, Notebook: `Charts/StackOverflowUmfrage/StackOverflowUmfrage.ipynb`, Daten-Quelle: FEHLT!

Nichtsdestotrotz scheinen beide Technologien als Kandidaten für einen detaillierteren Vergleich für dieses Projekt in Frage zu kommen. Im nächsten Kapitel soll evaluiert werden, welches Framework für die Entwicklung der Formular-Anwendung angemessener ist.

4.2 Vergleich React Native und Flutter

4.2.1 Vergleich zweier minimaler Beispiele für Formulare und Validierung

Es soll eine Formularanwendung mit komplexer Validierung im Rahmen dieser These erstellt werden. Es ist durchaus sinnvoll, die beiden Technologien anhand von Beispielanwendungen, welche Formulare und die Validierung dieser beinhalten, zu vergleichen. Deshalb soll nachfolgend jeweils eine solche Beispielanwendung der jeweiligen Technologie gefunden werden. Die Anwendungen werden sich stark voneinander unterscheiden, weshalb sie im nächsten Schritt vereinfacht und aneinander angeglichen werden. Anschließend wird ersichtlich werden, nach welchen Kriterien sich die Technologien im Hinblick auf die Entwicklung der Formularanwendung vergleichen lassen.

React Native React native stellt nur eine vergleichsweise geringe Anzahl von eigenen Komponenten zur Verfügung und zu diesen gehören keine, welche die Validierung von Formularen ermöglichen. Doch die im react.js Raum sehr bekannten Bibliotheken Formic, Redux Forms und React Hook Form sind alle drei kompatibel mit React Native.[14,15,16]

Für die Formular-Anwendung ist die Validierung komplexer Bedingungen nötig. Die Formular-Validierungs-Bibliotheken bieten in der Regel Funktionen an, welche überprüfen, ob ein Feld gefüllt ist oder der Inhalt einem speziellen Muster entspricht – wie etwa einem regulären Ausdruck. Doch solche mitgelieferten Validierungs-Funktionen reichen nicht aus, um die Komplexität der Bedingungen abzubilden. Stattdessen müssen benutzerdefinierte Funktionen zum Einsatz kommen.

Keiner der drei oben genannten Validierungs-Bibliotheken ist in dieser Hinsicht limitiert. Sie alle bieten die Möglichkeit, eine JavaScript Funktion für die Validierung zu übergeben. Diese Funktion gibt einen Wahrheitswert zurück – wahr, wenn das Feld oder die Felder valide sind, falsch, falls nicht. In React Hook Form ist es die Funktion register,

[14] Vgl. *React Native | Formik Docs*
[15] Vgl. *Does redux-form work with React Native?*
[16] Vgl. *React Native | React Hook Form - Get Started*

die ein Parameter-Objekt namens Register Options erhält, dessen Eigenschaft validate die JavaScript Funktion zugewiesen werden kann.[17] In Redux Form ist es die Initialisierungs-Funktion reduxForm, die ein Konfigurations-Objekt mit dem Namen config erhält, in welchem die Eigenschaft ebenfalls validate heißt.[18] Auch in Formic ist der Bezeichner validate, und ist als Attribut in der Formic Komponente zu finden.[19]

Es ist also absehbar, dass die Formular-Anwendung in React Native entwickelt werden kann. Die nötigen Funktionen werden von den Bibliotheken bereitgestellt. Einziger Nachteil hierbei ist, dass es sich um Drittanbieter Bibliotheken handelt, welche im Verlauf der Zeit an Beliebtheit gewinnen und verlieren können. Möglicherweise geht die Beliebtheit einer der Bibliotheken mit der Zeit zurück, weshalb es weniger Kontributionen wie etwa neue Funktionalitäten oder Fehlerbehebungen, sowie Fragen und Antworten und Anleitungen zu diesen Bibliotheken geben wird, da die Entwickler sich für andere Bibliotheken entscheiden. Die Wahl der Bibliothek kann also schwerwiegende Folgen wie Mangel an Dokumentation oder Limitationen im Vergleich zu anderen Bibliotheken mit sich bringen. Eine Migration von der einen Bibliothek zu einer anderen könnte in Zukunft notwendig werden, wenn diese Limitationen während der Entwicklung auffallen. Aus dem Grund ist es in der Regel von Vorteil, wenn solche Funktionalitäten bereits im Kern der Frontend-Technologie integriert sind. Der Fall, dass die Kern-Komponenten an Relevanz verlieren und empfohlen wird, auf externe Bibliotheken zuzugreifen, ist zwar nicht ausgeschlossen, geschieht aber im Wesentlichen seltener.

Flutter Die Flutter Dokumentation stellt in ihrer cookbook Sektion ein Beispiel einer minimalistischen Formularanwendung mit Validierung bereit.[20] Das Rezept ist Teil einer Serie von insgesamt fünf Anleitungen, welche Formulare in Flutter behandeln.[21]

4.2.2 Automatisiertes Testen

Automatisierte Tests in React Native Die React Native Dokumentation führt genau eine Seite mit einem Überblick über die unterschiedlichen Testarten. Dabei wird das Konzept von Unit Tests, Mocking, Integrations Tests, Komponenten Tests und Snapshot Tests kurz erläutert, jedoch ohne ein Beispiel zu geben oder zu verlinken. Vier Quellcodeschnipsel sind auf der Seite zu finden: Ein Schnipsel zeigt den minimalen Aufbau eines Tests, zwei weitere Schnipsel veranschaulichen beispielhaft, wie Nutzerinteraktionen getestet werden können und Letzteres zeigt die textuelle Repräsentation der Ausgabe einer Komponente, die für einen Snapshottest verwendet wird. Weiterhin wird auf die Jest API Dokumentation verwiesen, sowie auf ein Beispiel für einen Snapshot Test in der Jest Dokumentation.[I]

Um die notwendigen Anleitungen für das Erstellen der jeweiligen Tests ausfindig zu machen, ist es notwendig, die Dokumentation von React Native zu verlassen.

Die Dokumentation von Jest enthält mehr Details zum Einsatz der Testbibliothek, welches für mehrere auf Javascript basierende Frontend Frameworks kompatibel ist[II]. Somit muss zum Erstellen der Unit-Tests immerhin nur dieses Framework studiert werden.

Zum Entwickeln von Tests von React Native Komponenten wird unter anderem auf die

[17] Vgl. *register | React Hook Form - API*
[18] Vgl. *reduxForm | Redux Form - API*
[19] Vgl. *<Formik /> | Formik Docs API*
[20] Vgl. Google LLC, *Build a form with validation*
[21] Vgl. Google LLC, *Forms | Flutter Docs Cookbook*

[I] https://jestjs.io/docs/snapshot-testing
[II] https://jestjs.io/docs/getting-started

Bibliothek React Native Testing Library verwiesen. Anders als der Name vermuten lässt, handelt es sich nicht um eine von React Native bereitgestellte Bibliothek. Im Unterschied zur React Testing Library, von der sie inspiriert ist, läuft sie ebenso wie React Native selbst nicht in einer Browser-Umgebung.[22] Herausgegeben wird die React Native Testing Library vom Drittanbieter Callstack - ein Partner im React Native Ökosystem.[23]

Sie verwendet im Hintergrund den React Test Renderer[III], welcher wiederum vom React Team angeboten wird und auch zum Testen von react.js Anwendungen geeignet ist. Der React Test Renderer wird ebenfalls empfohlen, um Komponententests zu kreieren, die keine React Native spezifischen Funktionalitäten nutzen.

Um Integrationstests zu entwickeln - welche die Applikation auf einem physischen Gerät oder auf einem Emulator testen - wird auf zwei weitere Drittanbieter-Bibliotheken verlinkt: Appium[IV] und Detox[V]. Es wird darauf hingewiesen, dass Detox speziell für die Entwicklung von React Native Integrationstests entwickelt wurde. Appium wird lediglich als ein weiteres bekanntes Werkzeug erwähnt.

Es lässt sich damit zusammenfassen, dass der Aufwand der Einarbeitung für automatisiertes Testen in React Native vergleichsweise hoch ist. Die Dokumentation ist auf die Seiten der jeweiligen Anbieter verteilt. Der Entwickler muss sich den Überblick selbst verschaffen und zusätzlich die für das Framework React Native relevanten Inhalte identifizieren. Notwendig ist auch das Erlernen von mehreren APIs um alle Testarten abzudecken. Für einen Anfänger kommt erschwerend hinzu, dass eine Entscheidung für die eine oder andere Bibliothek notwendig wird. Um diese Entscheidung treffen zu können, ist eine Auseinandersetzung mit den Vor- und Nachteilen der Technologien im Vorfeld vom Entwickler zu leisten.

Automatisierte Tests in Flutter Die Flutter Dokumentation erklärt sehr umfangreich auf 11 Unterseiten die unterschiedlichen Testarten mit Quellcodebeispielen und verlinkt für jede Testart eine bis mehrere detaillierte Schritt-für-Schritt-Anleitungen, wie ein solcher Test erstellt wird.

Eine Seite erklärt den Unterschied zwischen Unit Test, Widget Test und Integrationstest[VI]. Eine weitere Seite erklärt Integrationstests in mehr Detail[VII].

Ein sogenanntes Codelab führt durch die Erstellung einer minimalistischen App und zwei Unit-, fünf Widget- und zwei Integrationstests für diese App[VIII]

Im sogenannten Kochbuch tauchen folgende Rezepte auf:

- 2 Rezepte für Unit Tests
  - eine grundlegende Anleitung zum Erstellen von Unit-Tests [IX]
  - Eine weitere Anleitung zum Nutzen von Mocks in Unit Test mithilfe der Bibliothek mockito [X]

[22] Vgl. Borenkraout, *Native Testing Library Introduction | Testing Library Docs*
[23] Vgl. Facebook Inc., *The React Native Ecosystem*

[III] https://reactjs.org/docs/test-renderer.html
[IV] http://appium.io/
[V] https://github.com/wix/detox/
[VI] https://flutter.dev/docs/testing
[VII] https://flutter.dev/docs/testing/integration-tests
[VIII] https://codelabs.developers.google.com/codelabs/flutter-app-testing
[IX] https://flutter.dev/docs/cookbook/testing/unit/introduction
[X] https://flutter.dev/docs/cookbook/testing/unit/mocking

- 3 Rezepte für Widget Tests
  - Eine grundlegende Anleitung zum Erstellen von Widget Tests [XI]
  - Ein Rezept mit detaillierteren Beispielen zum Finden von Widgets zur Laufzeit eines Widget Tests [XII]
  - Ein Rezept zum Testen von Nutzerverhalten wie dem Tab, dem Drag und dem Eingeben von Text [XIII]
- 3 Rezepte für Integrationstests
  - Eine grundlegende Anleitung zum Erstellen eines Integrationstests [XIV]
  - eine Anleitung zum Simulieren von dem Scrollen in der Anwendung während der Laufzeit eines Integrationstests [XV]
  - eine Anleitung zum Performance Profiling [XVI]

Zusammengefasst: Der Aufwand der Einarbeitung in das Testen in Flutter ist gering. Alle Werkzeuge werden vom Dart- und Flutter-Team bereitgestellt. Die Dokumentation ist umfangreich, folgt jedoch einem roten Faden. Eine Übersichtsseite fasst die Kerninformationen zusammen und verweist auf die jeweiligen Seiten für detailliertere Informationen und Übungen.

[XI] https://flutter.dev/docs/cookbook/testing/widget/introduction
[XII] https://flutter.dev/docs/cookbook/testing/widget/finders
[XIII] https://flutter.dev/docs/cookbook/testing/widget/tap-drag
[XIV] https://flutter.dev/docs/cookbook/testing/integration/introduction
[XV] https://flutter.dev/docs/cookbook/testing/integration/scrolling
[XVI] https://flutter.dev/docs/cookbook/testing/integration/profiling

Teil II

Implementierung

5 Schritt 1 - Formular in Grundstruktur erstellen

Im ersten Schritt soll die Formular-Anwendung in ihrer Grundstruktur entwickelt werden. Das beinhaltet alle drei Oberflächen, welche in den darauf folgenden Schritten lediglich erweitert werden. Das Formular erhält noch keine Validierung. Somit sind alle Eingaben oder nicht kompatible Selektionen erlaubt.Die erste Ansicht, welche der Benutzer sieht, soll die Übersicht der bereits eingetragenen Maßnahmen sein (Abb. 7).

Abbildung 7: Der Übersicht-Bildschirm zeigt in Schritt 1 zunächst nur die Maßnahmen mit ihrem Titel und Bearbeitungsdatum in den Kategorien „Abgeschlossen“ und „In Bearbeitung“, Quelle: Eigene Abbildung

Die Auflistung der Maßnahmen erfolgt in den Kategorien „In Bearbeitung“ und „Abgeschlossen“. Innerhalb dieser Rubriken werden die Maßnahmen in einer Tabelle angezeigt. Mit einem Klick auf den Button unten rechts im Bild wird der Benutzer auf die zweite Ansicht weitergeleitet: die Eingabemaske (Abb. 8).

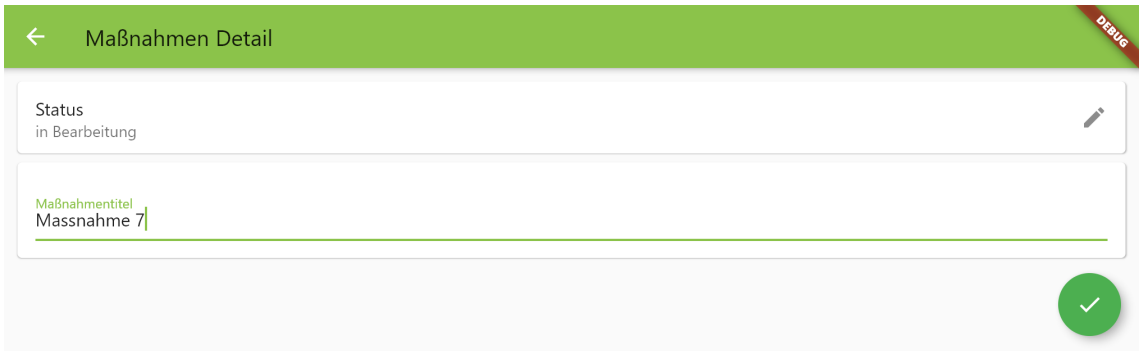


Abbildung 8: Die Eingabemaske zeigt im Schritt 1 eine Karte zum Selektieren des Status und ein Eingabefeld für den Titel, Quelle: Eigene Abbildung

Sie ermöglicht die Eingabe des Maßnahmen-Titels über ein simples Eingabefeld. Darüber hinaus ist die Selektions-Karte für den Status zu sehen. Mit einem Klick auf diese Karte öffnet sich der Selektions-Bildschirm. Er ermöglicht die Auswahl der Auswahloptionen, in

diesem Fall die Optionen „in Bearbeitung“ und „abgeschlossen“ (Abb. 9).

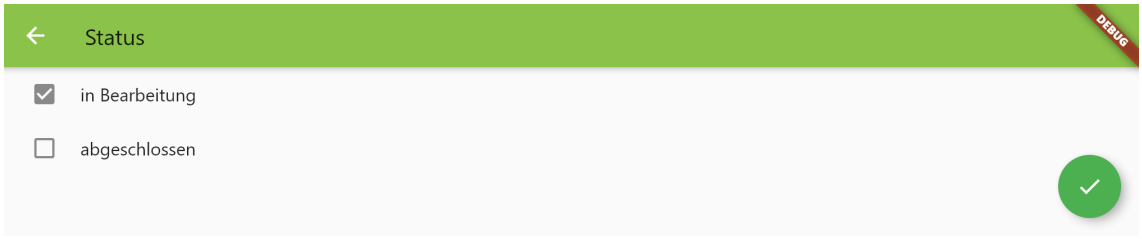


Abbildung 9: Der Selektions-Bildschirm für das Feld Status erlaubt die Auswahl der Optionen „in Bearbeitung“ und „abgeschlossen“, Quelle: Eigene Abbildung

5.1 Auswahloptionen hinzufügen

Dart verfügt – anders als beispielsweise Java[24] – nicht über Aufzählungstypen mit zusätzlichen Eigenschaften. Das Schlüsselwort `enum` in Dart erlaubt lediglich die Auflistung konstanter Symbole[25]. Für die Auswahl Optionen ist es jedoch notwendig, dass es zwei Eigenschaften gibt:

- die Abkürzung, die in der resultierendem Datei gespeichert werden soll
- und der Beschreibungstext, welcher in der Oberfläche angezeigt wird.

Das hat den Hintergrund, dass die Abkürzungen weniger Speicherplatz einnehmen und die Beschreibung sich in Zukunft auch ändern darf. Würde anstatt der Abkürzung die Beschreibung als Schlüssel verwendet werden, so würde eine Datei, die mit einer älteren Version des Formulars erstellt wurde, nicht mehr von neueren Versionen der Applikation eingelesen werden können. Der alte Beschreibungstext würde nicht mehr mit dem Text übereinstimmen, der als Schlüssel in der Anwendung verwendet wird.

Die beiden Zustände „in Bearbeitung“ und „abgeschlossen“ werden daher in Listing **??** als statische Klassenvariablen deklariert (Z. 6-7). Die beiden Konstruktor-Aufrufe übergeben dabei als erstes Argument die Abkürzung und als zweites Argument die Beschreibung. Der Konstruktor selbst (Z. 9-10) deklariert die beiden Parameter als positionale Parameter.

```
5  class LetzterStatus extends Choice {
6    static final bearb = LetzterStatus("bearb", "in Bearbeitung");
7    static final fertig = LetzterStatus("fertig", "abgeschlossen");
8
9    LetzterStatus(String abbreviation, String description)
10       : super(abbreviation, description);
11 }
```

Listing 15: Die Klasse LetzterStatus, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/choices/choices.dart

Positionale Parameter Im Vergleich zu den benannten Parametern ist bei den positionalen Parametern nur ihre Reihenfolge in der Parameterliste ausschlaggebend. Das Argument für die `abbreviation` steht dabei also immer an erster Stelle und das Argument für `description` immer an der zweiten (Z. 6-7). Positionale Parameter sind vorgeschrieben. Werden sie ausgelassen, so gibt es einen Compilerfehler.[26]

[24] Vgl. Gosling u. a., *The Java® Language Specification Java SE 16 Edition*, S. 321.
[25] Vgl. Google LLC, *Dart Programming Language Specification 5th edition*, S. 74f.
[26] Vgl. Google LLC, *Dart Programming Language Specification 5th edition*, S. 74f.

Die Klasse `LetzterStatus` erbt von der Basisklasse `Choice` (Z. 5). Der Konstruktor der Klasse (Z. 9) übergibt beide Parameter als Argumente an den Konstruktor der Klasse `Choice`. Genau wie in Java wird mithilfe des Schlüsselwortes `super` (Z. 10) der Konstruktor der Basisklasse aufgerufen. Doch anders als in Java erfolgt der Aufruf des super Konstruktors nicht in der ersten Zeile des Konstruktor-Körpers[27]. Weil das Aufrufen des Konstruktors der Basisklasse zum statischen Teil der Objekt-Instanziierung gehört, muss der Aufruf von `super` in der Initialisierungsliste erfolgen. Die Initialisierungsliste wird mit dem `:` nach der Parameterliste eingeleitet (Z. 10)[28].

Die Basisklasse `Choice` (Listing 16) deklariert lediglich die beiden Felder `description` und `abbreviation` jeweils als `String` (Z. 4-5). Beide sind mit `final` gekennzeichnet, was sie zu unveränderlichen Instanzvariablen macht. Nach der Initialisierung, können sie keine anderen Werte annehmen.[29] Die Initialisierung der beiden Variablen muss im statischen Kontext der Instanziierung erfolgen. Mit der abgekürzten Schreibweise `this.abbreviation` und `this.description` im Konstruktor (Z. 7) werden die Parameter den Feldern zugewiesen.

```
3 class Choice {
4   final String description;
5   final String abbreviation;
6
7   const Choice(this.abbreviation, this.description);
```

Listing 16: Die Klasse Choice, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/choices/base/choice.dart

Dies erübrigt sowohl die Angabe des Parametertypes mittels `(String abbreviation, String description)`, denn der Typ des Parameters kann bereits durch Angabe des Typs in der Instanzvariablen-Deklaration(Z. 4-5) abgeleitet werden. Außerdem entfällt auch die Zuweisung, die man ansonstenin der Form `this.abbreviation = abbreviation` und `this. description = description` in der Initialisierungsliste erreichen würde.[30]

Die Variable `letzterStatusChoices` (Listing. 17 Z. 13) fasst die beiden statischen Klassenvariablen als eine Kollektion zusammen. Da es sich um eine solche Kollektion handelt, in der jedes Element nur ein einziges Mal vorkommen darf, ist hier von einer Menge zu sprechen. Auffällig hier ist, dass das Schlüsselwort new fehlt. In Dart ist das Schlüsselwort für die Konstruktion von Instanzen optional. Die Klasse, die zur Konstruktion dieser Menge verwendet wird, ist die selbst erstellte Klasse choices. Über das Typargument LetzterStatus wird erreicht, das ausschließlich Variablen dieses Typs in der Menge eingefügt werden dürfen. Wird stattdessen eine Variable eingefügt, die weder vom selben Typ, noch von einem Typ, der von letzter Status erbt, so gibt es einen Compilerfehler. Dies dient einzig und allein dem Zweck, dem Fehler vorzubeugen, dass aus Versehen falsche Optionen in der Menge eingetragen werden. Über den Parameter name ist es möglich dieser Menge die Beschriftung "Status" hinzuzufügen. Es handelt sich hier um einen benannte Parameter.

```
13 final letzterStatusChoices = Choices<LetzterStatus>(
14     {LetzterStatus.bearb, LetzterStatus.fertig},
15     name: "Status");
```

Listing 17: Die Menge letzterStatusChoices, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/choices/choices.dart

[27] Vgl. Gosling u. a., *The Java® Language Specification Java SE 16 Edition*, S. 310.
[28] Vgl. Google LLC, *Dart Programming Language Specification 5th edition*, S. 42.
[29] Vgl. Google LLC, *Dart Programming Language Specification 5th edition*, S. S16.
[30] Vgl. Google LLC, *Dart Programming Language Specification 5th edition*, S. 40f.

Listing 18 zeigt die Klasse `Choices`. Sie erbt von `UnmodifiableSetView` und erlaubt damit die Erstellung einer eigenen Menge - auch `Set` genannt Referenz. Methoden, die man von einem `Set` erwartet, lassen sich somit direkt auf Instanzen der Klasse `Choices` aufrufen. Darunter unter anderem die `contains` Methode, welche erlaubt, das Vorhandensein eines Objektes im `Set` zu überprüfen Referenz.

```
10 class Choices<T extends Choice> extends UnmodifiableSetView<T> {
11   final String name;
12   final Map<String, T> choiceByAbbreviation;
13
14   T? fromAbbreviation(String? abbreviation) => choiceByAbbreviation[abbreviation];
15
16   Choices(Set<T> choices, {required this.name})
17       : choiceByAbbreviation = {
18           for (var choice in choices) choice.abbreviation: choice,
19         },
20         super(choices);
21 }
```

Listing 18: Die Klasse Choices, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/choices/base/choice.dart

Instanzvariable `name` (Z. 11) wird im Konstruktor `16` zugewiesen. Auffällig hierbei ist, dass der Parameter in geschweiften Klammern geschrieben steht und das Schlüsselwort `required` vorangestellt ist. Das macht den Parameter zu einem vorgeschriebenen benannten Parameter.

Vorgeschriebene benannte Parameter Gewöhnlicher benannte Parameter sind optional. Wird ihnen das Schlüsselwort `required` vorangestellt, so müssen sie gesetz werden, denn sonst gibt es einen Compilerfehler. An dieser Stelle ist das `required` Schlüsselwort sinnvoll, denn es handelt sich um den Datentyp String der nicht den wert `null` annehmender. Würde der Parameter aber optional sein, so wäre es möglich, das programm zu kompilieren, auch wenn bei Aufrufen des Konstruktors kein Argument für den Parameter übergeben wurde. Doch in diesem Fall gäbe es keinen Initialwert für Name und somit müsste der Instanzvariable null zugewiesen werden. in der statischen Analyse wird daher sichergestellt, das Instanzvariablen durch benannte Parameter nur dann initialisiert werden dürfen, wenn dieser durch required als vorgeschrieben gekennzeichnet sind und damit unter keinem Umstand ausgelassen werden können. Dürfte name den Wert Null annehmen, So würde es sich um den nullable Datentyp String mit der Notation String? Handeln.

Neben `name` wird mit `choiceByAbbreviation` eine weitere Instanzvariable deklariert (Z. 12). Es handelt sich um den Datentyp `Map` - eine Kollektion die Daten mittels Schlüssel Wertepaaren ablegen kann. Als Schlüssel wird die Abkürzung mit dem Datentyp `String` verwendet. Als Wert ist der generische Typ-Parameter `T` angegeben. Er ist in Zeile 10 deklariert und muss mindestens von der Klasse `Choice` erben. In `choiceByAbbreviation` werden also die Auswahlmöglichkeiten über ihre Abkürzung abgelegt und können über dieselbe wieder referenziert werden. Da es sich auch hier um eine unveränderliche Instanzvariable handelt, muss sie schon in der Initialisierungsliste initialisiert werden (Z. 17-19). Dabei wird zunächst mit der öffnenden geschweiften Klammer (Z. 17) ein sogenanntes Literal einer Map begonnen, welches mit schließenden geschweiften Klammer (Z. 19) endet. Mehr zu `Map` Literalen in dem Grundlagenkapitel `Kapitel einfügen`.

Auffällig ist jedoch, dass In Zeile 18 dem Set lateral keine einfache Auflistung von Werten übergeben wird. Stattdessen wird das mit dem sogenannte collection for eine wiederholung verwendet.

In Zeile 18 wird durch die Menge aller Auswahloptionen `choices` iteriert und dabei in jedem

Schleifendurchlauf die Auswahloption in die Variablen `choice` gespeichert. Während des Schleifendurchlauf wird dann ein Schlüssel-Wertepaar gebildet wobei `choice.abbreviation` der Schlüssel ist und das Objekt `choice` selbst der Wert.

Die Map `choiceByAbbreviation` erlaubt es nach der Initialisierung mit Hilfe der Methode `fromAbbreviation` (Z. 14) über die Abkürzung das dazugehörige `Choice`-Objekt abzurufen. Beispielsweise gibt der Befehl `letzterStatusChoices.fromAbbreviation('fertig')` das Objekt `LetzterStatus("fertig", "abgeschlossen")` zurück. Auffällig dabei ist das der Parameter abbreviation Mit dem Typen `String?` und der generische Rückgabetyp mit `T?` gekennzeichnet ist. Der Suffix `?` macht beide zu Typen mit Null-Zulässigkeit.

Die Methode `fromAbbreviation` soll für die Deserialisierung genutzt werden. Sollten im Formular Auswahlfelder leer gelassen worden sein, so haben entsprechenden Variablen den Wert `null`. Wenn nun das Formular abgespeichert wird, so tauchen auch in der abgespeicherten Json-Datei keine Werte für das Feld auf. Wird die Datei gelesen wird die Methode `fromAbbreviation` genutzt um aus der in der Json-Datei gespeicherten Abkürzung wieder die entsprechende Auswahl Option abzurufen. Sollte jedoch kein Wert hinterlegt sein, so wird `letzterStatusChoices.fromAbbreviation(null)` aufgerufen werden. Dadurch wird klar, dass der Parameter `null` zulassen muss. Es impliziert auch, dass potenziell `null` zurückgeben werden kann, da für den Schlüssel `null` kein Wert in der Map hinterlegt sein kann. Deshalb erlaubt auch der Rückgabetyp `T?` Null-Werte.

5.2 Serialisierung einer Maßnahme

Damit die Daten angezeigt und verändert werden können, müssen sie zunächst serialisierbar sein, sodass sie auf einen Datenträger geschrieben und von dort auch wieder gelesen werden können. Die zwei bekanntesten Bibliotheken zum Serialisieren in Dart heißen json_serializable und built_value. Beide haben gemeinsam, dass sie Quellcode generieren, welcher die Umwandlung der Objekte in JSON übernimmt. built_value bietet im Gegensatz zu JSON Serializable jedoch die Möglichkeit unveränderbare Werte-Typen - sogenannte immutable value types - zu erstellen. Da diese unveränderbaren Werte noch bei der Erstellung des sogenannten ViewModels - Mehr dazu im Kapitel `Einfügen` - hilfreich werden, wurde sich für diese Bibliothek entschieden.

Ein Werte-Typ für built_value erfordert etwas Boilerplate-Code, um den generierten Quellcode mit der selbstgeschriebenen Klasse zu verknüpfen. Entwicklungsumgebungen wie Visual Studio Code und Android Studio erlauben solchen Boilerplate Code generieren zu lassen und dabei nur die erforderlichen Platzhalter einzugeben. In Visual Studio Code werden diese Templates „Snippets“ genannt, in Android Studio heißen sie „Live Templates“. Listing 59 zeigt, wie das live Template für das Generieren eines Wertetyps für built_value aussieht. Templates für built_value wie dieses und weitere müssen nicht vom Nutzer eingegeben werden, sondern existieren bereits als Plugin für die beiden Entwicklungsumgebungen[XVII], [XVIII].

```
6  part '$file_name$.g.dart';
7
8  abstract class $ClassName$ implements Built<$ClassName$, $ClassName$Builder> {
9      $todo$
10
11     $ClassName$._();
12     factory $ClassName$([void Function($ClassName$Builder) updates]) = _$$ClassName$;
13 }
```

Listing 19: Live Template für die Erstellung von built_value Boilerplate-Code in Android Studio, Quelle: Jetbrains Marketplace Built Value Snippets Plugin

`$ClassName$` Wird dabei jeweils durch den gewünschten Klassennamen ersetzt. Android Studio erlaubt, dass bei Einfügen des live templates der Klassenname einmalig eingegeben werden muss. Anschließend wird mithilfe des live templates der Boilerplate Code generiert.

In Listing 20 ist der Werte-Typ `Massnahme` zu sehen. Die Zeilen 11 bis 13, sowie 23 bis 28 wurden dabei automatisch erstellt. Die Zeilen 14 bis 21 wurden hinzugefügt. Zunächst soll die Maßnahme über die `guid` eindeutig identifiziert werden können.

Globally Unique Identifier Ein GUID – Kurzform von Globally Unique IDentifier – ist eine Folge von 128 Bits, die zur Identifikation genutzt werden kann. Eine solche GUID hat einer textuelle Repräsentation wie beispielsweise die folgende: `'f81d4fae-7dec-11d0-a765-00a0c91e6bf6'`

Die Attribute `letzteBearbeitung` und `identifikatoren` sind im Gegensatz zu dem String-Attribut guid zusammengesetzte Datentypen, die im Folgenden weiter beleuchtet werden.

Auffällig ist, dass es sich hier um eine abstrakte Klasse handelt und die drei Attribute jeweils Getter-Methoden ohne Implementierung sind. Eine solche Getter-Methode speichert keinen wert, sondern gibt lediglich den Wert eines Feldes zurück. Die dazugehörigen

[XVII] https://plugins.jetbrains.com/plugin/13786-built-value-snippets
[XVIII] https://marketplace.visualstudio.com/items?itemName=GiancarloCode.built-value-snippets

```dart
part 'massnahme.g.dart';

abstract class Massnahme implements Built<Massnahme, MassnahmeBuilder> {
  String get guid;

  LetzteBearbeitung get letzteBearbeitung;

  Identifikatoren get identifikatoren;

  static void _initializeBuilder(MassnahmeBuilder b) =>
      b..guid = const Uuid().v4();

  Massnahme._();

  factory Massnahme([void Function(MassnahmeBuilder) updates]) = _$Massnahme;

  static Serializer<Massnahme> get serializer => _$massnahmeSerializer;
}
```

Listing 20: Der Werte-Typ Massnahme, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/data_model/massnahme.dart

Felder, Setter-Methoden, die konkrete Klasse und der restliche generierte Code ist in der gleichnamigen Datei mit der Endung `.g.dart` (Zeile 11) zu finden.

Die Klassen-Methode `_initializeBuilder` kann in jedem Werte-Typ hinterlegt werden, um Standardwerte für Felder festzulegen. Die Methode wird intern von „built_value“ aufgerufen. Bei dem Feld „guid“ handelt es sich um einen String, der keine Null-Werte zulässt. Könnte das Feld auch Null-Werte annehmen, so wäre die Notation in Dart dafür stattdessen `String? get guid;` . „built_value“ erwartet also immer einen Wert für dieses Feld. Sollte die Datei gelesen werden, welche die Maßnahmen enthält, so enthält jede Maßnahme bei der Deserialisierung den abgespeicherten Wert für die `guid` und somit wird das Feld gefüllt. Doch sollte eine leere Maßnahme über einen Konstruktor erstellt werden, so wäre das Feld „guid“ leer und „built_value“ würde einen Fehler auslösen. Aus diesem Grund wird in der Zeile 21 für das Feld `guid` ein Standardwert festgelegt, nämlich eine zufällige generierte ID die dem Standard Uuid der Version 4 entspricht. Zu diesem Zweck wird das Builder-Objekt verwendet. Die Klasse `MassnahmeBuilder` gehört dabei zu dem von „built_value“ generierten Quellcode. Der Parametername wird hier – wie so häufig im builder pattern – mit einem b für Builder abgekürzt. Die Syntax `=>` leitet einen sogenannten „arrow function body“ ein. Dabei handelt es sich schlicht um einen Funktions-Körper, der genau eine Anweisung ausführt und deshalb nicht in öffnenden und schließenden geschweiften Klammern gesetzt werden muss.[31] Auf dem Bilder-Objekt können dann die Eigenschaften so gesetzt werden, als wären sie die Eigenschaften von dem Objekt `Massnahme` . In Wahrheit werden sie aber nur auf den Builder-Objekt angewendet. Ebenfalls auffällig ist die Syntax `b..guid` . Statt dem Punkt zum Zugriff auf Attribute des Objektes wird hier der sogenannte Kaskadierungs-Operator benutzt.

Der Kaskadierungs-Operator Durch Eingabe von zwei aufeinanderfolgende Punkten `..` statt nur einem `.` können mehrere Operationen an einem Objekt ausgeführt werden, ohne das Objekt zuvor einer Variable Zuzuweisen oder die Operationen über dessen Namen wiederholt aufzurufen.[32] Beispiel: die Aufrufe `objekt.tueEtwas();` `objekt.tueEtwasAnderes();` und `objekt..tueEtwas()..tueEtwasAnderes();` sind äquivalent.

Da der Kaskadierung-Operator jedoch dazu verwendet wird, mehrere Operationen auf ei-

[31] Vgl. Google LLC, *Dart Programming Language Specification 5th edition*, S. 18f., 234.
[32] Vgl. Google LLC, *Dart Programming Language Specification 5th edition*, S. 149f.

nem Objekt auszuführen, hat er in Zeile 16 keine Funktion. Doch bei Änderung eines Objektes über das builder pattern werden für gewöhnlich mehrere Operationen am gleichen Builder Objekt ausgeführt, weshalb - der Stringenz halber - der Kaskadierung-Operator immer im Zusammenhang mit dem Bilder Objekt verwendet wird.

Die Attribute `letzteBearbeitung` und `identifikatoren` (Z. 11, 13) erhalten dagegen ganz automatisch Standardwerte in Form von Instanzen der dazugehörigen Klassen. Diese wiederum konfigurieren ihre eigenen Felder und deren initialen Werte.

Der Werte-Typ `Identifikatoren` ist in Listing 21 zu sehen. Er enthält das Attribut `massnahmenTitel`, welcher im Eingabeformular durch das Texteingabefeld gefüllt werden wird.

```
25 abstract class Identifikatoren
26     implements Built<Identifikatoren, IdentifikatorenBuilder> {
27   String get massnahmenTitel;
28
29   static void _initializeBuilder(IdentifikatorenBuilder b) =>
30       b..massnahmenTitel = "";
```

Listing 21: Der Werte-Typ Identifikatoren, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/data_model/massnahme.dart

Schließlich enthält der Werte-Typ `LetzteBearbeitung` in Listing 22 noch die Attribute `letztesBearbeitungsDatum` in Zeile 43 und `letzterStatus` in Zeile 50. Im Eingabeformular wird der Selektions-Bildschirm den Inhalt des Feldes `letzterStatus` Bestimmen. Der initiale Wert auf wird in Zeile 54 auf einen konstanten Wert gesetzt, der dem Zustand `'in Bearbeitung'` entspricht - mehr dazu im Kapitel `Kapitel einfügen`.

```
41 abstract class LetzteBearbeitung
42     implements Built<LetzteBearbeitung, LetzteBearbeitungBuilder> {
43   DateTime get letztesBearbeitungsDatum;
44
45   String get formattedDate {
46     final date = letztesBearbeitungsDatum;
47     return "${date.year}-${date.month}-${date.day} ${date.hour}:${date.minute}";
48   }
49
50   String get letzterStatus;
51
52   static void _initializeBuilder(LetzteBearbeitungBuilder b) => b
53     ..letztesBearbeitungsDatum = DateTime.now().toUtc()
54     ..letzterStatus = LetzterStatus.bearb.abbreviation;
```

Listing 22: Der Werte-Typ LetzteBearbeitung, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/data_model/massnahme.dart

Das Attribut `letztesBearbeitungsDatum` ist dagegen nicht im Formular änderbar, sondern wird einmalig in Zeile 53 auf den aktuellen Zeitstempel gesetzt. Zugehörig zu diesem Attribut gibt es noch eine abgeleitete Eigenschaft namens `formattedDate` (Z. 45-48). Es ist eine Hilfsmethode, die das letzte Bearbeitungsdatum in ein für Menschen lesbares Datumsformat umwandelt. In dem Übersichts-Bildschirm Abbildung 7 ist das Datumsformat sichtbar.

Da diese Getter-Methode eine Implementierung besitzt, wird für sie von „built_value“ kein Quellcode für die Serialisierung generiert.

Bevor die Werte-Typen serialisiert werden können, muss built_value jedoch noch mitgeteilt werden, für welche Werte-Typen Serialisierungs-Funktionen generiert werden sollen. Dazu werden über die Annotation `@SerializersFor` die gewünschten Klassen aufgelistet (Listing. 23 Z. 10). Die Zeilen 11 und 12 sind dabei immer gleich, es sei denn, es ist ein anderer Se-

rialisierung Algorithmus gewünscht. In diesem Fall wird das `StandardJsonPlugin` verwendet.

```
10 @SerializersFor([Massnahme, Storage])
11 final Serializers serializers =
12     (_$serializers.toBuilder()..addPlugin(StandardJsonPlugin())).build();
```

Listing 23: Der Serialisierer für Massnahme und Storage, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/data_model/serializers.dart

Wird nun der Befehl `flutter pub run build_runner build` ausgeführt, so wird der Quellcode generiert und die Werte-Typen können für die Serialisierung genutzt werden.

5.3 Test der Serialisierung einer Maßnahme

Das Ergebnis der Serialisierung wird im dazugehörigen Unit-Test ersichtlich (Listing 24). In Zeile 8 wird ein Objekt der Klasse `Massnahme` instanzieiert. Anders als bei gewöhnlichen Datentypen lassen sich bei diesem unveränderlichen Datentyp keine Attribute nach der Erstellung anpassen. Die einzige Möglichkeit besteht darin, ein neues Objekt mit dem gewünschten Attributwert zu erstellen und die restlichen Werte des alten Objektes zu übernehmen. Dies ist mit Hilfe des sogenannten Builder-Entwurfsmuster es möglich, welches in built_value Anwendung findet.

```
6  test('Massnahme serialises without error', () {
7    var massnahme = Massnahme();
8    massnahme = massnahme
9        .rebuild((b) => b.identifikatoren.massnahmenTitel = "Massnahme 1");
10
11   var actualJson = serializers.serializeWith(Massnahme.serializer, massnahme);
12
13   var expectedJson = {
14     'guid': massnahme.guid,
15     'letzteBearbeitung': {
16       'letztesBearbeitungsDatum': massnahme
17           .letzteBearbeitung.letztesBearbeitungsDatum.microsecondsSinceEpoch,
18       'letzterStatus': 'bearb'
19     },
20     'identifikatoren': {'massnahmenTitel': 'Massnahme 1'}
21   };
22
23   expect(actualJson, equals(expectedJson));
```

Listing 24: Serialisierung einer Maßnahme Unittest, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/test/data_model/massnahme_test.dart

Erbauer-Entwurfsmuster Das Erbauer-Entwurfsmuster - englisch builder pattern - ist ein Erzeugungsmuster, welches die Konstruktion komplexer Objekte von ihrer Repräsentation trennt. Es gehört zu der Serie von Entwurfsmustern der Gang of Four.[33]. Im Fall von built_value trennt es die unveränderlichen Objekte von ihrer Konstruktion. Über den Builder lassen sich Änderungen an diesen unveränderlichen Objekten vornehmen, wodurch eine Kopie dieses unveränderlichen Objekt mit der gewünschten Änderung, zurückgegeben wird.

In den Zeilen 9 bis 10 wird so ein neues Objekt von der Klasse Maßnahme mit Hilfe der Methode `rebuild` erzeugt und anschließend der Referenz `massnahme` zugewiesen, wodurch sie ihren alten Wert verliert. Über die generiertet Methode `serializers .serializeWith` kann das Objekt in Json übersetzt werden. Der erste Parameter `Massnahme.serializer` gibt dabei

[33] Vgl. Gamma u. a., *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*, S. 119.

an, wie diese Serialisierung erfolgen soll und auch dieses Objekt wurde von „built_value“ generiert. Der zweite Parameter ist die tatsächliche `massnahme`, die in Json umgewandelt werden soll. Die Zeilen 13 bis 21 erstellen das Json Dokument, mit dem das serialisierte Ergebnis am Ende verglichen werden soll. Dabei werden die gleichen Eigenschaften eingetragen. So etwa die `guid` (Z. 14), welcher bei der Initialisierung der Maßnahme automatisch und zufällig erstellt wurde. Außerdem das letzte Bearbeitungsdatum, welches den Zeitstempel erhält, zudem die Maßnahme generiert wurde. Da built_value bei der Serialisierung die Datumswerte in Mikrosekunden umwandelt, muss für das erwartete Json-Dokument das gleiche passieren (Z. 16-17). Der `'letzterStatus'` (Z. 18) wird hierbei auf den Standardwert `'bearb'` gesetzt und der `'massnahmenTitel'` (Z. 20) auf den gleichen Wert, der in Zeile 9 übergeben wurde. Schließlich vergleicht die Methode `expect` das tatsächlich serialisierter Json-Dokument mit dem, welches zuvor zum Vergleich aufgebaut wurde. Der zweite Parameter ist ein sogenannter `Matcher` und die Variante mit dem Namen `equals` überprüft auf absolute Gleichheit.

Analog zur Serialisierung testet der Unit-Test in Listing 25 auch die Deserialisierung. Das Json-Dokument ist dabei sehr ähnlich und unterscheidet sich lediglich in zwei Details. Der `'guid'` wird auf einen festen Wert festgelegt (Z. 38), statt - wie zuvor - durch den in dem Initialisierungs-Prozess der Maßnahme zufällig generiert zu werden. Außerdem wird auch das `letztesBearbeitungsDatum` festgesetzt, nämlich auf die Microsekunde `0` (Z. 40).

```
36 test('Massnahme deserialises without error', () {
37   var json = {
38     'guid': "test massnahme id",
39     'letzteBearbeitung': {
40       'letztesBearbeitungsDatum': 0,
41       'letzterStatus': 'bearb'
42     },
43     'identifikatoren': {'massnahmenTitel': 'Massnahme 1'}
44   };
45
46   var expectedMassnahme = Massnahme((b) => b
47     ..guid = "test massnahme id"
48     ..identifikatoren.massnahmenTitel = "Massnahme 1"
49     ..letzteBearbeitung.update((b) {
50       b.letztesBearbeitungsDatum =
51           DateTime.fromMicrosecondsSinceEpoch(0, isUtc: true);
52     }));
53   var actualMassnahme =
54       serializers.deserializeWith(Massnahme.serializer, json);
55
56   expect(actualMassnahme, equals(expectedMassnahme));
```

Listing 25: Deserialisierung einer Maßnahme Unittest, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/test/data_model/massnahme_test.dart

Zum Vergleich wird in den Zeilen 46 bis 52 eine Maßnahme über das Builder-Entwurfsmuster generiert und die gleichen feste Werte werden für die Eigenschaften übergeben. Dabei ist darauf zu achten, dass die Instanzvariable `letzteBearbeitung` keinen Wert über den Zuweisungs-Operator `=` erhält, sondern stattdessen die Methode `update` darauf aufgerufen wird.

Da es sich bei der Instanzvariable `letzteBearbeitung` genauso um ein Object eines Wertetypen - handelt, ist sie ebenso unveränderlich. Deshalb kann sie nur über einen Builder manipuliert werden. Ein Blick in den generierten Quellcode offenbart, dass es sich in Wahrheit um einen Bilder handelt (Listing. 26 Z. 224-225).

Außerdem müssen die Microsekunden für das Datum zunächst in ein Objekt von `DateTime` umgewandelt werden muss. Dafür wird der benannte Konstruktor `fromMillisecondsSinceEpoch`

```
216 class MassnahmeBuilder implements Builder<Massnahme, MassnahmeBuilder> {
217   _$Massnahme? _$v;
218
219   String? _guid;
220   String? get guid => _$this._guid;
221   set guid(String? guid) => _$this._guid = guid;
222
223   LetzteBearbeitungBuilder? _letzteBearbeitung;
224   LetzteBearbeitungBuilder get letzteBearbeitung =>
225       _$this._letzteBearbeitung ??= new LetzteBearbeitungBuilder();
226   set letzteBearbeitung(LetzteBearbeitungBuilder? letzteBearbeitung) =>
227       _$this._letzteBearbeitung = letzteBearbeitung;
```

Listing 26: Instanzvariable `letzteBearbeitung` gibt einen `LetzteBearbeitungBuilder` zurück, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/data_model/massnahme.g.dart

von `DateTime` (Z. 51) aufgerufen.

Benannte Konstruktoren In Programmiersprachen wie beispielsweise Java können Methoden überladen werden, indem ihre Methodensignatur geändert wird. Beim Aufruf der Methode kann über die Anzahl und die Typen der übergebenen Argumente die gewünschte Methode gewählt werden. Das gleiche gilt für Konstruktoren. Wird ein weiterer Konstruktor für eine Klasse in Java benötigt, so besteht einzig und allein die Möglichkeit darin, den Konstruktor zu überladen. Sowohl überladene Methoden als auch überladene Konstruktoren existieren in Dart nicht. Wird also in dartein Alternative constructorgewünscht Komma so muss er einen Namen bekommen. Beim Aufruf des Konstruktors wird dieser Name dann mit einem `.` nach dem Klassennamen angegeben, um den gewünschten Konstruktor zu benennen.

Ganz ähnlich wie bei der Serialisierung wird nun mit dem Befehl `serializers . deserializeWith` unter Angabe des Objektes, welches die Deserialisierung übernehmen soll - nämlich wiederum `Massnahme.serializer` - das Json-Dokument in ein Objekt das Werte-Typs `Massnahme` deserialisiert (Z. 53-54). Schließlichh wird in Zeile 56 das Ergebnis der Deserialisierung mit dem gewünschten Ergebnis verglichen.

Gibt man in der Kommandozeile den Befehl `flutter test test /data_model /massnahme _test.dart` ein, so werden die Tests in der Testdatei ausgeführt. Die Ausgabe `00:01 +2: All tests passed!` teilt mit, dass beide Tests ausgeführt und beide Ergebnisse mit den verglichenen Werten übereinstimmten.

5.4 Serialisierung der Maßnahmenliste

Damit alle Maßnahmen - statt nur einer einzigen - in einer Datei zusammengefasst werden können, müssen die Maßnahmen zunächst zu einer Menge zusammengefasst werden, die ebenfalls serialisierbar ist. Der Werte-Typ Storage ist dafür vorgesehen (Listing 27). Er deklariert allein das `BuiltSet massnahmen` (Z. 10). Ein BuiltSet ist die Abwandlung eines gewöhnlichen Sets, jedoch unter anderem mit der Möglichkeit, es mit einem Builder zu erstellen und das Set zu serialisieren. Die Übergabe des Typarguments `<Massnahme>` gewährleistet, dass keine anderen Objekte Eingefügt werden können, die weder eine Instanz der Klasse `Massnahme` sind, oder einer Klasse, die von `Massnahmen` erbt.

Der Befehl `flutter pub run build_runner build` stößt erneut die Quellcodegenerierung für den Werte-Typen `Storage` an.

```
9  abstract class Storage implements Built<Storage, StorageBuilder> {
10   BuiltSet<Massnahme> get massnahmen;
```

Listing 27: Der Werte-Typ Storage, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/data_model/storage.dart

5.5 Test der Serialisierung der Maßnahmenliste

Nun soll noch überprüft werden, ob die Menge von Maßnahmen mit genau einer eingetragenen Maßnahme korrekt serialisiert. Auch das wird von einem Unit Test überprüft (Listing 28). In Zeile 8 wird das leere Objekt `storage` erstellt. In Zeile 9 wird es dann wiederverwendet, um aufbauend auf der Kopie Änderungen mithilfe der `rebuild`-Methode durchzuführen.

```
7  test('Storage with one Massnahme serialises without error', () {
8    var storage = Storage();
9    storage = storage.rebuild((b) => b.massnahmen.add(
10       Massnahme((b) => b.identifikatoren.massnahmenTitel = "Massnahme 1")));
11
12   var actualJson = serializers.serializeWith(Storage.serializer, storage);
13
14   var expectedJson = {
15     "massnahmen": [
16       {
17         "guid": storage.massnahmen.first.guid,
18         "letzteBearbeitung": {
19           "letztesBearbeitungsDatum": storage
20               .massnahmen
21               .first
22               .letzteBearbeitung
23               .letztesBearbeitungsDatum
24               .microsecondsSinceEpoch,
25           "letzterStatus": "bearb"
26         },
27         "identifikatoren": {"massnahmenTitel": "Massnahme 1"}
28       }
29     ]
30   };
31   expect(actualJson, equals(expectedJson));
```

Listing 28: Ein automatisierter Testfall überprüft, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/test/data_model/storage_test.dart

Bei der Instanzvariable `massnahmen` der Klasse `Storage` handelt es sich um ein `BuiltSet`. Der Aufruf von `b.massnahmen` gibt allerdings nicht dieses `BuiltSet` zurück. Wäre es so, so könnte die Operation `add` nicht darauf angewendet werden. Ein `BuiltSet` stellt keine Methoden zur Manipulation des Sets zur Verfügung. In Wahrheit gibt der Ausdruck `b.massnahmen` einen `SetBuilder` zurück. Das kann im generierten Quellcode nachgesehen werden (Listing. 29 Z. .)

```
91 class StorageBuilder implements Builder<Storage, StorageBuilder> {
92   _$Storage? _$v;
93
94   SetBuilder<Massnahme>? _massnahmen;
95   SetBuilder<Massnahme> get massnahmen =>
96       _$this._massnahmen ??= new SetBuilder<Massnahme>();
```

Listing 29: Instanzvariable `massnahmen` gibt einen `SetBuilder` zurück, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/data_model/storage.g.dart

Der `SetBuilder` wiederum erlaubt es, Änderungen am Set vorzunehmen und stellt dafür die

- für ein Set übliche - Methode `add` bereit. Im Aufruf von `add` wird dann ein Objekt des Werte-Typs Maßnahme konstruiert (Z. 10). Dazu wird dieses Mal die anonyme Funktion zum Konstruieren der Maßnahme gleich im Konstruktor übergeben.

Diesmal konstruiert die Methode `serializers.serializeWith` mit dem Serialisierer `Storage.serializer` ein weiteres JSON-Object (Z. 12). Genau wie zuvor wird ein Json-Dokument vorbereitet (Z. 14-30), welches der `Matcher` `equals` gegen das serialisierte Dokument des soeben konstruierten Objects `storage` vergleicht (Z. 31). Das Json-Dokument unterscheidet sich nur darin, dass es einen Knoten namens `'massnahmen'` enthält, der als Wert eine Liste hat. Die Liste hat nur ein Element. Weil dieses Mal das Objeckt des Typs `Massnahme` nicht direkt zugreifbar ist, muss es zunächst über die Liste der Maßnahmen aus dem `storage`-Object abgerufen werden. Das ist mit dem Befehl `first` möglich, der das erste Objekt - und in diesem Fall einzige Objekt - der Kollektion zurückgibt (Z. 17, 21). Darüber kann erneut die `guid` und das `letztesBearbeitungsDatum` abgerufen werden.

Ein weiterer Unit-Test überprüft, ob auch die Deserialisierung eines `storage`-Objects erfolgreich ist. Er ist in Listing 67 im Anhang D zu finden. Auch dieser Test ist der Deserialisierung des Objektes des Typs `Massnahme` sehr ähnlich. Er unterscheidet sich nur darin, dass das `Massnahme`-Objekt in der Liste `massnahmen` des `storage`-Objektes enthalten ist.

5.6 Der Haupteinstiegspunkt

Das Listing 30 zeigt den Haupteinstiegspunkt des Programms. Darin ist erkennbar, dass sich die Applikation in drei Rubriken einteilen lässt:

- das Model (Z. 27-30)
- der View (Z. 41-44)
- das ViewModel. (Z. 25-26)

Model View ViewModel Das ModelViewViewmodel Entwurfsmuster wurde zunächst von John Gossman für die Windows Presentation Foundation beschrieben. Das Model beschreibt die Datenzugriffs-Komponente welche die Daten in relationalen Datenbanken oder hierarchischen Datenstrukturen wie XML oder JSON ablegt. Der View beschreibt die Oberflächenelemente wie Texteingabefelder und Buttons. Diese beiden Komponenten sind auch aus dem ModelViewController Entwurfsmuster bekannt. Das ModelViewViewmodel Entwurfsmuster ist eine Weiterentwicklung davon und integriert das sogenannte ViewModel. Es ist dafür zuständig als Schnittstelle zwischen View und Model zu fungieren. Die Daten des Models lassen sich in der Regel nicht direkt mit Oberflächen Elementen verknüpfen. Denn es kann es notwendig sein, dass die Oberfläche weitere temporäre Daten benötigt, die aber nicht mit den Daten des Models gespeichert werden sollen. Das ViewModel übernimmt diese Arbeit, indem es die Daten des Models abruft und sie in veränderter Form den Oberflächen-Elementen zur Verfügung steht. Andersherum formt es die Eingaben in der Nutzeroberfläche so um, dass sie im strikten Datenmodell des Models Platz finden.[34]

`MassnahmenModel` (Z. 29) verwaltet die eingegebenen Daten der Maßnahmen und nutzt die Abhängigkeit `MassnahmenJsonFile` (Z. 27) um die Daten auf einem Datenträger als eine JSON-Datei zu speichern. Somit gehören diese beiden Klassen dem Model an.

`MassnahmenFormViewModel` (Z. 25) greift die Daten des Models ab und formt diese um, sodass sie von dem View `MassnahmenDetailScreen` (Z. 43) verändert werden können. Sollen die Daten gespeichert werden, so stellt `MassnahmenFormViewModel` ebenfalls Methoden zur Verfügung um die Daten wieder in das Format des Models einpflegen zu können.

`MassnahmenMasterScreen` (Z. 41) stellt eine Ausnahme dar, denn dieser View präsentiert die Daten aus dem Model ohne eine Schnittstelle über ein ViewModel. Das ist möglich, weil die Daten nicht manipuliert, sondern nur angezeigt werden müssen.

Damit sowohl ViewModel als auch Model von jedem View heraus abrufbar sind, werden sie in eine Art Service eingefügt (Z. 23). Das Widget AppState ist dieser Service. Er erhält das Model (Z. 24) und das ViewModel (Z. 25) im Konstruktor. Die Abhängigkeit zum Schreiben des Models in eine JSON Datei `MassnahmenJsonFile` bekommt das Model ebenfalls im Konstruktor übergeben (Z. 24). `AppState` ist das erste Element, welches im Widget-Baum auftaucht. Die gesamte restliche Applikation ist als Kind-Element hinterlegt (Z. 26). Damit können alle Widgets auf den Service zugreifen.

Service Locator und Dependency Inection Das Service Locator Entwurfsmuster folgt dem Umsetzungsparadigma Inversion of Control – deutsch Umkehrung der Steuerung. Frameworks folgen diesem Muster, indem sie als erweiterbare Skelett-Applikationen fungieren. Anstatt, dass die Applikation den Programmfluss steuert und dabei selbst Funktionen aufruft, wird die Programmflusssteuerung an das Framework abgegeben und mit Hilfe von

[34] Vgl. Gossman, *Introduction to Model/View/ViewModel pattern for building WPF apps*.

Ereignissen ermöglicht, dass das Framework Funktionen des Nutzers aufruft.[35] Im Service Locator Entwurfsmuster werden Komponenten darüber hinaus zentral registriert und über dieses Register anderen Komponenten zur Interaktion zur Verfügung gestellt.[36] Anstatt die Komponenten direkt miteinander zu verknüpfen, werden Sie für den Zugriff von praktisch überall vorbereitet. Vor allem für automatisierte Tests ist dies von Vorteil, da solche Abhängigkeiten ausgetauscht werden können, um ganz spezielle Teil-Funktionalitäten eines Programms zu testen. Mehr dazu im Kapitel `Kapitel einfügen`.

Anders als der Name vermuten lässt, steuert `MaterialApp` nicht nur das Aussehen der Applikation im Material Design Look. Darüber hinaus stellt das Widget auch Grundfunktionalitäten einer App, wie etwa den Navigator bereit. Damit hat die Applikation die Möglichkeit – ähnlich wie bei einer Webside – auf Unterseiten zu navigieren. Hat der Benutzer die Arbeit in der Unterseite vollendet, so kann der Navigator gebeten werden, zur vorherigen Ansicht zurückzukehren. Mit dem Parameter `routes` (Z. 34-39) erfolgt die Angabe der Unterseiten, die besucht werden können. Über `initialRoute` (Z. 39) kann die Startseite angegeben werden.

```
18 class MassnahmenFormApp extends StatelessWidget {
19   const MassnahmenFormApp({Key? key}) : super(key: key);
20
21   @override
22   Widget build(BuildContext context) {
23     return AppState(
24         model: MassnahmenModel(MassnahmenJsonFile()),
25         viewModel: MassnahmenFormViewModel(),
26     child: MaterialApp(
27       title: 'Maßnahmen',
28       theme: ThemeData(
29         primarySwatch: Colors.lightGreen,
30         accentColor: Colors.green,
31         primaryIconTheme: const IconThemeData(color: Colors.white),
32       ),
33       initialRoute: MassnahmenMasterScreen.routeName,
34       routes: {
35         MassnahmenMasterScreen.routeName: (context) =>
36         const MassnahmenMasterScreen(),
37         MassnahmenDetailScreen.routeName: (context) =>
38         const MassnahmenDetailScreen()
39       },
40     ));
41   }
42 }
```

Listing 30: Der Haupteinstiegspunkt, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/main.dart

[35] Vgl. Johnson und Foote, „Designing reusable classes“.
[36] Vgl. Fowler, *Inversion of Control Containers and the Dependency Injection pattern*.

5.7 Der Service für den Applikations übergreifenden Zustand

10

Um Daten an alle Kind Elementen im Widgets mitzugeben, finden die sogenannten „InheritedWidgets“ Anwendung. Der Service `AppState` (Listing 31) ist genauso ein solches.

Im Konstruktor erhält er zunächst bei den Parameter des Typs `Key` (Z. 7). Es ist gängige Praxis in Flutter, jedem Widget im Konstruktor zu ermöglichen, einen solchen Schlüssel zu übergeben. Es ist jedoch optional. Ein solcher Schlüssel kann genutzt werden, um das Widget eindeutig zu identifizieren und es unter anderem über den Schlüssel wiederzufinden. In den Zeilen 8 und 9 werden das Model und das ViewModel dem Objekt im Konstruktor übergeben. In den Zeilen 14 und 15 sind sie deklariert. Das letzte Element im Konstruktor ist das `child`. Ihm muss der Widget-Baum übergeben werden, dem der Zustand verfügbar gemacht werden soll.

Der Aufruf des Basis-Konstruktors mit den Argumenten `key` und `child` ist in Zeile 11 zu sehen. Die Basisklasse von `InheritedWidget` ist `ProxyWidget` und erhält exakt dieselben Argumente. Das `ProxyWidget` verwendet das Kindelement, um es im WidgetBaum unterhalb von sich selbst zu zeichnen. Eine eigene Methode zum Zeichnen musst also nicht für das `InheritedWidget` implementiert werden. Die einzige Methode, welche implementiert werden muss, ist `updateShouldNotify` (Z. 24). Immer dann, wenn das `InheritedWidget` selbst aktualisiert wird, kann es alle Widgets, die davon abhängig sind, benachrichtigen. In dem Fall werden diese Widgets ebenfalls neu gezeichnet. Für die Formular-Applikation ist das allerdings nicht gewünscht. Die Aktualisierung der Oberfläche soll in den nachfolgenden Schritten selbst kontrolliert werden. Deshalb erfolgt die Rückgabe `false`, da in Zukunft nicht gewünscht ist, den Applikations-Zustand komplett auszutauschen. Um die Aktualisierung der Oberfläche kümmern sich sowohl Model als auch ViewModel.

Damit ein Widget Abhängigkeit von dem `AppState` anmelden kann, verwendet es in seiner eigenen `build`-Methode die Methode `dependOnInheritedWidgetOfExactType<AppState>()`. Der Aufruf der Methode erfolgt auf dem Objekt vom Typ `BuildContext`. Weil dieser Kontext bei jedem Zeichnen allen Kindern übergeben wird, kann jedes Kind darüber die Vater-Elemente wiederfinden.

Damit der Aufruf leichter lesbar und kürzer ist, empfiehlt das Flutter-Team eine eigene Klassenmethode zu erstellen, welche die Methode für den Benutzer aufruft (Z. 16-17). Auch eine Fehlermeldung kann bei dieser Auslagerung geworfen werden, sollte im Kontext kein Objekt des gewünschten Typs vorhanden sein (Z. 18). Das Widget, welches auf den `AppState` zugreifen möchte, kann es mit der einfachen Schreibweise `AppState.of(context)` abrufen.

Abbildung zeigt die Beziehung zwischen den Bildschirmen und dem AppState auf. Sowohl MassnahmenMasterScreen und MassnahmenDetailScreen müssen auf MassnahmenModel und MassnahmenFormViewModel zugreifen können. Zu diesem Zweck erstellt MassnahmenFormApp den AppState. Er enthält sowohl ViewModel als auch Model. Über ihn können beide Bildschirme auf Model und ViewModel zugreifen.

```dart
class AppState extends InheritedWidget {
  const AppState({
    Key? key,
    required this.model,
    required this.viewModel,
    required Widget child
  }) : super(key: key, child: child);

  final MassnahmenFormViewModel viewModel;
  final MassnahmenModel model;

  static AppState of(BuildContext context) {
    final AppState? result = context.dependOnInheritedWidgetOfExactType<AppState>();
    assert(result != null, "Kein AppState im 'context' gefunden");
    return result!;
  }

  @override
  bool updateShouldNotify(covariant AppState oldWidget) => false;
}
```

Listing 31: Der Service AppState, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/widgets/app_state.dart

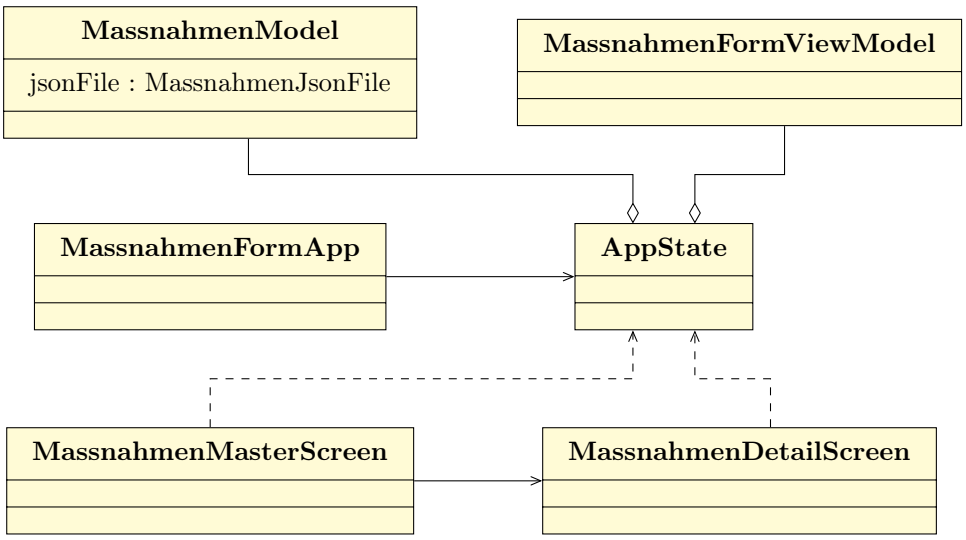


Abbildung 10: UML Diagramme, Quelle: Eigene Abbildung

5.8 Speichern der Maßnahmen in eine Json-Datei

Das Model wird durch die Klasse `MassnahmenJsonFile` in eine JSON-Datei gespeichert (Listing 32). Der Dateipfad wird dabei durch die Methode `_localMassnahmenJsonFile` (Z. 8-11) abgerufen. Die Hilfsmethode `getApplicationSupportDirectory` (Z. 9) gibt aus dem Nutzerverzeichnis des aktuellen Nutzers den zur Applikation zugeordneten Datei-Ordner zurück. Auf Windows-Betriebssystemen wäre das beispielsweise `C:\Users\AktuellerNutzer\AppData\Roaming\com.example\conditional_form`.

Dadurch, dass dem Methoden-Bezeichner `_localMassnahmenJsonFile` ein Unterstrich vorangestellt ist, ist die Methode privat und kann nur innerhalb der Klasse aufgerufen werden. Dart hat damit eine Konvention zum Standard werden lassen. In Programmiersprachen wie beispielsweise C++ wurde der Unterstrich zusätzlich den Bezeichnern von Instanz Attributen vorangestellt, die mit dem `private` Schlüsselwort gekennzeichnet sind, damit sie überall im Quellcode als private Attribute identifizierbar sind, ohne dazu die Klassendefinition ansehen zu müssen. In Dart gibt es dagegen das `private` Schlüsselwort nicht. Stattdessen wird der Unterstrich vor dem Bezeichner verwendet, um ein Instanzattribut privat zu deklarieren.

Die Getter-Methode `_localMassnahmenJsonFile` hat den Rückgabetyp `Future<File>` und ist zudem mit dem Schlüsselwort `async` gekennzeichnet. Asynchron muss die Methode deshalb sein, weil sie auf den Aufruf `getApplicationSupportDirectory` warten musst, der ebenfalls asynchron abläuft.

Der Funktion `saveMassnahmen` (Z. 13-16) wird ein JSON Objekt in Form einer Hashtabelle übergeben. Sie ruft die Hilfs-Getter-Methode `_localMassnahmenJsonFile` (Z. 14) auf und schreib den Dateiinhalt in die Datei des abgefragten Pfades (Z. 15). Zuvor wird dazu das JSON-Object in eine textuelle Repräsentation überführt. Dazu dient die Funktion `jsonEncode`.

Das Äquivalent dazu stellt die Methode `readMassnahmen` (Z. 18-30) dar. Auch sie ruft den Dateipfad ab (Z. 19), überprüft allerdings im nächsten Schritt, ob die Datei bereits existiert (Z. 21). Sollte das der Fall sein, so wird die Datei eingelesen (Z. 23). Die textuelle Repräsentation aus der Datei wird mittels Methode `jsonDecode` in ein Json-Objekt in der Form einer Hashtabelle gespeichert (Z. 24) und schließlich zurückgegeben (Z. 26). Sollte die Dateien nicht existieren, führt das zu einer Ausnahme (Z. 28), welche von der aufrufenden Funktion behandelt werden kann.

```dart
class MassnahmenJsonFile {
  Future<File> get _localMassnahmenJsonFile async {
    var directory = await getApplicationSupportDirectory();
    return File("${directory.path}/Maßnahmen.json");
  }

  Future<void> saveMassnahmen(Map<String, dynamic> massnahmenAsJson) async {
    var file = await _localMassnahmenJsonFile;
    await file.writeAsString(jsonEncode(massnahmenAsJson));
  }

  Future<Map<String, dynamic>> readMassnahmen() async {
    var file = await _localMassnahmenJsonFile;

    var fileExists = await file.exists();
    if (fileExists) {
      final fileContent = await file.readAsString();
      final jsonObject = jsonDecode(fileContent) as Map<String, dynamic>;

      return jsonObject;
    } else {
      throw MassnahmenFileDoesNotExistException("$file was not found");
    }
  }
}
```

Listing 32: Die Klasse MassnahmenJsonFile, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/persistence/massnahmen_json_file.dart

5.9 Abhängigkeit zum Verwalten der Maßnahmen

Die Art und Weise, wie die Maßnahmen abgerufen werden, sollte nach Möglichkeit abstrahiert werden. Das erlaubt, den Mechanismus in Zukunft auszutauschen, ohne dabei den Rest der Applikation verändern zu müssen. So wäre es beispielsweise denkbar, statt einer Json-Datei eine direkte Verbindung zu einer relationalen Datenbank herzustellen. Auch das Austauschen der Abhängigkeit mit einem Platzhalter, der lediglich die Aufrufe der Methoden zählt, ist damit möglich. Ein solches Platzhalterobjekt wird „Mock“ genannt und für automatisiertes Testen eingesetzt (siehe Kapitel `Kapitel einfügen`). Ebenso abstrahiert werden soll der Umgang mit Ausnahmen. Sollte die Datei nicht verfügbar sein, so muss die Oberfläche davon nicht zwingend betroffen sein. Stattdessen kann der Service sich entscheiden, eine leere Liste von Maßnahmen zurückzugeben. Sobald die Liste manipuliert wird, kann eine neue Datei angelegt werden und sie mit den eingegebenen Daten beschreiben. Die Klasse MassnahmenModel (Listing 33) tut genau das.

Sie bekommt `MassnahmenJsonFile` im Konstruktor übergeben (Z. 11). Daraufhin ruft der Konstrukteur gleich die `init` auf (Z. 12), welche in den Zeilen 15-22 deklariert ist. Darin wird der Stream `storage` (Z. 19) initialisiert. Es handelt sich um eine Erweiterung eines „broadcast streams“ mit dem Namen `BehaviorSubject` (Z. 9). Es entstammt dem Paket rx.dart, welches die Streams in Dart um eine Reihe von weiteren Funktionalitäten erweitert. Ein `BehaviorSubject` hat die Besonderheit, dass es den Wert des letzten Ereignisses zwischenspeichern. Die „broadcast streams“ haben für gewöhnlich den Nachteil, das neue Zuhörer des Streams nur die neuen Ereignisse erhalten. Alle in der Vergangenheit erfolgten Ereignisse sind nicht mehr verfügbar. Vor allem dann, wenn in der Oberfläche der letzte Wert eines Streams verwendet werden soll, um Elemente zu zeichnen, ist das von einem besonderen Nachteil. Denn wenn der Stream zuvor initialisiert wurde, so gibt es keine Daten zu dem Zeitpunkt, wenn die Oberfläche gezeichnet wird. Sollte die Oberfläche jedoch gezeichnet werden, bevor der Stream initialisiert wurde, so existieren ebenfalls keine Daten. Hier kommt das `BehaviorSubject` ins Spiel. Sobald die Oberfläche gezeichnet wird und der Stream bereits initialisiert ist, kann dennnoch auf den zuletzt übertragenen Wert zurückgegriffen werden. Anschließend überträgt der Stream die folgenden Aktualisierungen für die Oberfläche mit jedem neuen Ereignis, so wie es für Streams üblich ist.

Der Stream kann nicht bereits in der Initialisierungsliste des Konstruktors mit den Daten aus der Json-Datei gefüllt werden. Das liegt daran, dass die Json-Daten dazu zunächst gelesen werden müssen, was nur durch eine Reihe von asynchronen Operation möglich ist. In einer Initialisierungsliste können allerdings keine asynchronen Operationen ausgeführt werden. Deshalb wird `init` erst im Konstruktor-Körper aufgerufen (Z. 7).

Damit der Stream anfangs nicht leer ist, füllt ihn der benannte Konstruktor `seeded` mit einem leeren Objekt des Typs `Storage` (Z. 9). Sobald die Datei gelesen (Z. 17) und anschließend deserialisiert wurde (Z. 20), erhält der Stream über die Setter-Methode `value` ein neues Ereignis mit dem gelesenen Wert (Z. 19).

Die Initialisierung ist von einem `try`-Block umgeben. Sollte die Initialisierung verschlagen, weil die Json-Datei nicht existiert, wird die entsprechende Fehlerbehandlung ausgeführt (Z. 21). Diese ist leer, da sich im Stream bereits ein leeres `Storage`-Objekt befindet. Mit diesem leeren Objekt kann die Oberfläche weiterarbeiten. In Zukunft könnte es sinnvoll sein, innerhalb der Fehlerbehandlung eine Meldung an den Benutzer zu geben, um darüber zu informieren, dass eine neue Datei angelegt wurde.

Mit `putMassnahmeIfAbsent` (Z. 24-33) steht eine Methode bereit, um gleichzeitig sowohl die Oberfläche, als auch die Json-Datei zu aktualisieren. Sollte die eigetragene Maßnahme schon existieren, wird sie zunächst gelöscht (Z. 26). In jedem Fall wird die neue Maßnahme dem Stream hinzugefügt (Z. 27). Durch Austauschen des gesamten Objektes mit der Zu-

weisung von `storage.value` (Z. 25) erhält der Stream erneut ein neues Ereignis, womit er die Oberfläche benachrichtigen kann, sich neu zu zeichnen. Außerdem wird die Serialisierung des `Storage`-Objektes und angestoßen (Z. 29-30) und die neue Liste von Maßnahmen im darauffolgenden Schritt zurück in die Json-Datei gespeichert (Z. 32).

```
7  class MassnahmenModel {
8    final MassnahmenJsonFile jsonFile;
9    final storage = BehaviorSubject<Storage>.seeded(Storage());
10
11   MassnahmenModel(this.jsonFile) {
12     init();
13   }
14
15   init() async {
16     try {
17       final massnahmenAsJson = await jsonFile.readMassnahmen();
18
19       storage.value =
20           serializers.deserializeWith(Storage.serializer, massnahmenAsJson)!;
21     } on MassnahmenFileDoesNotExistException {}
22   }
23
24   putMassnahmeIfAbsent(Massnahme massnahme) async {
25     storage.value = storage.value.rebuild((b) => b.massnahmen
26       ..removeWhere((m) => m.guid == massnahme.guid)
27       ..add(massnahme));
28
29     var serializedMassnahmen =
30         serializers.serializeWith(Storage.serializer, storage.value);
31
32     await jsonFile.saveMassnahmen(serializedMassnahmen as Map<String, dynamic>);
33   }
34 }
```

Listing 33: Die Klasse MassnahmenModel, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/data_access/massnahmen_model.dart

5.10 Übersichtsbildschirm der Maßnahmen

Der erste Bildschirm - die Übersicht der Maßnahmen - kann auf das im letzten Schritt erstellten Model zugreifen. In Listing 34 ist die Struktur des Übersicht-Bildschirms zu sehen. Über die Route `/massnahmen_master` ist der Bildschirm erreichbar (Z. 16). Die `build`-Methode zeichnet die Oberfläche `21-111`. Da ein Objekt des Typs MassnahmenPool im zentralen Register der Provider hinterlegt wurde, kann mit der Methode Provider.of darauf zugegriffen werden.

Mittels `AppState.of(context)` ist nun der Zugriff auf sowohl Model als auch ViewModel möglich. Zur einfacheren Verwendung sind sie als lokale Variablen zwischengespeichert (Z. 20-21).

Das Widget `Scaffold` - deutsch Gerüst - stell ein grundlegendes Layout mit einer Überschrift und einem Bereich für den Inhalt bereit (Z. 23). Das `Scaffold` kann auch Mitteilungen an den Benutzer am unteren Bildschirmrand einblenden.

Die Überschrift wird in der sogenannten `AppBar` hinterlegt (Z. 24). Sie unterstützt weitere Funktionalitäten. Sollte es sich bei der aktuell besuchten Route um eine Unterseite handeln, taucht links von der Titel-Überschrift einen Button zum Zurücknavigieren auf. Weiterhin können rechts von der Titelleiste Aktionsbuttons hinzugefügt werden, welches für die Formular Anwendung allerdings nicht nötig ist.

Zusätzlich kann dem `Scaffold` ein Button für die primäre Aktion auf diesem Bildschirm hinzugefügt werden: der sogenannte `FloatingActionButton` (Z. 88-97). Bei Aktivierung des Buttons navigiert die Applikation zur Eingabemaske, um eine neue Maßnahme anzulegen (Z. 98).

Das Eingabeformular sollte den Benutzer auffordern, tatsächlich leere Eingabefelder zu füllen. Deshalb muss die Aktivierung des Buttons auch das ViewModel neu initialisieren. Dies geschieht durch Zuweisung einer leeren Maßnahme zu zur Setter-Methode `vm.model` (Z. 95). Ohne die Neuinitialisierung würde die Eingabemaske immer die zuletzt eingetragene Maßnahme enthalten. Dies würde große Verwirrung beim Benutzer stiften.

The `FloatingActionButton` erhält den Schlüssel `createNewMassnahmeButtonKey` (Z. 89). Er ist als `GlobalKey` deklariert (Z. 11). Er findet beim Integrationstest Anwendung, um den Button zu finden (Siehe Kapitel `Kapitel einfügen`).

Der Inhaltsbereich des `Scaffold` beinhaltet das Widget `StreamBuilder` (Z. 27). Er kann auf Streams horchen, die Ereignisse des Typs `Storage` übermitteln. Er horcht auf Änderungen im Model, um genau zu sein auf Änderungen des Streams `model.storage` (Z. 28). Sobald der `StreamBuilder` ein Ereignis erhält, so führt er die Methode aus, die als Argument des Parameters `builder` hinterlegt ist. Alle Widgets außerhalb davon, wie etwa das `Scaffold`, erhalten dabei keine Aufforderung zum Neuzeichnen, sobald eine Maßnahme hinzugefügt wird. Das wirkt sich positiv auf die Laufzeit-Geschwindigkeit aus.

```dart
final createNewMassnahmeButtonKey = GlobalKey();

class MassnahmenMasterScreen extends StatelessWidget {
  static const routeName = '/massnahmen_master';

  const MassnahmenMasterScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final model = AppState.of(context).model;
    final vm = AppState.of(context).viewModel;

    return Scaffold(
      appBar: AppBar(
        title: const Text('Maßnahmen Master'),
      ),
      body: StreamBuilder<Storage>(
          stream: model.storage,
          builder: (context, _) {
            return SingleChildScrollView(
              // ...
            );
          }),
      floatingActionButton: FloatingActionButton(
          key: createNewMassnahmeButtonKey,
          child: const Icon(
            Icons.post_add_outlined,
            color: Colors.white,
          ),
          onPressed: () {
            vm.model = Massnahme();
            Navigator.of(context).pushNamed(MassnahmenDetailScreen.routeName);
          }),
    );
  }
}
```

Listing 34: Die Struktur der Klasse MassnahmenMasterScreen, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/screens/massnahmen_master.dart

5.10.1 Auflistung der Maßnahmen im Übersichtsbildschirm

Der Inhalt der `builder`-Methode ist in Listing 35 dargestellt. Das erste Widget ist ein `SingleChildScrollView` (Z. 30). Das Argument `scrollDirection` ist nicht gefüllt, weshalb die Standardoption - die vertikale Scrollrichtung - gewählt wird. Sollte die Liste der Maßnahmen die Höhe des Fensters überschreiten, so kann der Benutzer vertikal über die Liste scrollen.

Das Kind das Scrollbereich ist ein `Column`-Widget (Z. 31). Sie zeichnet Widgets, die als Argument des Parameters `children` gesetzt sind, von oben nach unten. Der Parameter `crossAxisAlignment` gibt an, wie die Kindelemente ausgerichtet sein sollen. `crossAxis` bedeutet dabei die zur Anzeige-Richtung entgegengesetzten Richtung. Da die `Column` vertikal zeichnet, ist mit `crossAxis` die horizontale Achse gemeint. `CrossAxisAlignment.start` beschreibt, dass Elemente entlang der horizontalen Achse an dessen Startpunkt auszurichten sind. Dadurch sind alle Elemente der Liste linksbündig.

Zuerst kommt die Auflistung der abgeschlossenen Maßnahmen. Die Überschrift `"Abgeschlossen"` (Z. 37), soll einen Abstand von jeweils 16 Pixel in alle Richtungen haben. Das ermöglicht das Widget `Padding` (Z. 35-40) und das Argument `EdgeInsets.all(16.0)`. Nach der Überschrift erscheint als zweites Element in der `Column` ein weiterer `SingleChildScrollView` (Z. 41-57), allerdings dieses Mal mit horizontaler Scroll-Richtung (Z. 42). Sollten die Informationen der Maßnahmen die Breite des Fensters überschreiten, kann der Nutzer von links nach rechts scrollen.

Die Informationen der Maßnahmen werden in einer Tabelle angezeigt. Dies übernimmt das selbstgeschriebener „Widget“ `MassnahmenTable` (Z. 45). Als erstes Argument erfolgt die Übergabe der anzuzeigenden Maßnahmen aus dem Model. `storage.value.massnahmen` gibt den aktuellen Wert des Streams des `storage`-Objektes zurück und greift auf die Liste der Maßnahmen zu. Mit der Methode `where` (Z. 47) kann ein Filter auf die Liste angewendet werden. Die übergebene anonyme Funktion (Z. 47-49) überprüft, ob der letzte Status auf fertig gesetz ist. Dazu reicht der Vergleich der Abkürzung. Nur wenn die Bedingung erfüllt ist, bleibt die Maßnahme in der gefilterten Kollektion zurück. Ein solcher Filter gibt ein sogenanntes „lazy“ `Iterable` zurück. Erst beim Zugriff auf das Ergebnis findet der Filter Anwendung. Doch es gibt keinen Zwischenspeicher für die gefilterten Elemente. Jeder Zugriff filtert die Elemente also neu. Der Aufruf `toSet` bewirkt allerdings das Speichern der Ergebnisse in einer Menge (Z. 50). Das Resultat erhält das Widget `MassnahmenTable` zur Anzeige.

Ein weiterer Parameter ist `onSelect` (Z. 50). Als Argument kann eine Funktion mit genau einem Parameter gesetzt werden. Sollte der Benutzer in der Tabelle eine Maßnahme auswählen, so löst er damit die Funktion aus. Der erste Parameter enthält dann die ausgewählte Maßnahme. Daraufhin soll sich wieder die Eingabemaske öffnen (Z. 55-56). Dann beinhalten die Eingabefehler jedoch die Werte der ausgewählten Maßnahme. Um das zu erreichen, reicht eine Zuweisung der Maßnahme an das ViewModel (Z. 51). Allerdings soll die Maßnahme zuvor ein neues letztes Bearbeitungsdatum mit dem aktuellen Zeitstempel erhalten (Z. 51-53).

Unterhalb der Rubrik der finalen Maßnahmen, listed die Übersicht die Maßnahmen, welche sich noch im Entwurf befinden (Z. 59-83). Daher ist das dritte Element der `Column` wiederum eine Überschrift: `"In Bearbeitung"` (Z. 62) gefolgt von einem weiteren horizontalen Scrollbereich (Z. 66-83) mit einer Tabelle von Maßnahmen (Z. 70-82). Der einzige Unterschied hier: die Bedingung der Filterfunktion. Dieses Mal filtert die Kollektion auf Maßnahmen in Bearbeitung (Z. 73-74).

```
return SingleChildScrollView(
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      const Padding(
        padding: EdgeInsets.all(16.0),
        child: Text(
          "Abgeschlossen",
          style: TextStyle(fontSize: 20),
        ),
      ),
      SingleChildScrollView(
          scrollDirection: Axis.horizontal,
          child: Padding(
            padding: const EdgeInsets.all(16.0),
            child: MassnahmenTable(
                model.storage.value.massnahmen
                    .where((m) =>
                        m.letzteBearbeitung.letzterStatus ==
                        LetzterStatus.fertig.abbreviation)
                    .toSet(), onSelect: (selectedMassnahme) {
              vm.model = selectedMassnahme.rebuild((m) => m
                ..letzteBearbeitung.letztesBearbeitungsDatum =
                    DateTime.now().toUtc());

              Navigator.of(context)
                  .pushNamed(MassnahmenDetailScreen.routeName);
            }),
          )),
      const Padding(
        padding: EdgeInsets.all(16.0),
        child: Text(
          "In Bearbeitung",
          style: TextStyle(fontSize: 20),
        ),
      ),
      SingleChildScrollView(
          scrollDirection: Axis.horizontal,
          child: Padding(
            padding: const EdgeInsets.all(16.0),
            child: MassnahmenTable(
                model.storage.value.massnahmen
                    .where((m) =>
                        m.letzteBearbeitung.letzterStatus ==
                        LetzterStatus.bearb.abbreviation)
                    .toSet(), onSelect: (selectedMassnahme) {
              vm.model = selectedMassnahme.rebuild((m) => m
                ..letzteBearbeitung.letztesBearbeitungsDatum =
                    DateTime.now().toUtc());

              Navigator.of(context)
                  .pushNamed(MassnahmenDetailScreen.routeName);
            }),
          )),
    ],
  ),
);
```

Listing 35: Die Ausgabe der Maßnahmen, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/screens/massnahmen_master.dart

5.11 Widget MassnahmenTable

Die `MassnahmenTable` ist ein `StatelessWidget` (Listing. 36 Z. 6). Zur Anzeige eignet sich das Widget `Table` (Z. 15-31).

Im Verlauf der Erstellung der Arbeit, wurde versucht das Widget `DataTable` zu verwenden. Doch im Gegensatz zur `DataTable` erlaubt es die Table, unterschiedlich hohe Zeilen zu zeichnen. Die Höhe der Zeile wird dazu in Abhängigkeit von dem benötigten Inhalt der Zellen berechnet. Die Breite und Ausrichtung der Spalten kann konfiguriert werden. Die Eigenschaft `IntrinsicColumnWidth` Sorgt dafür, dass die Spalten immer genau so groß sind, wie der Inhalt es benötigt (Z. 17). Zeilenumbrüche für die Texte in den Spalten sind somit nicht notwendig. `TableCellVerticalAlignment.middle` lässt die Tabelle die Inhalte zentriert darstellen (Z. 18).

Der Parameter `children` erhält als Argument eine Liste von `TableRow` Elementen (Z. 20-30). Die erste Tabellenzeile `20-23` beinhaltet die Spalten-Bezeichnungen. Jede `TableRow` hat wiederum den Parameter `children`. Das Argument bezieht sich hier auf die Zellen in der Zeile. Dabei ist wichtig, dass jede `TableRow` die gleiche Anzahl von Zellen hat. Weicht nur eine Zeile davon ab, zeichnet sich die gesamte Tabelle nicht und eine Ausnahme wird ausgelöst. Für die Spaltenbezeichnungen wurde eine Hilfsmethode kreiert: `_buildColumnHeader` (Z. 34-37). Sie zeichnet die Spalten mit einem Abstand von 8 Pixel in alle Richtungen.

Nach den Spaltenbezeichnungen folgen die Zeilen für die Daten der Maßnahmen (Z. 24-29). Die Methode `map` (Z. 24) ermöglicht es dazu durch die Liste der Maßnahmen zu iterieren und für jede Maßnahme ein Element eines völlig anderen Typs - in diesem Fall `TableRow` - zurückzugeben. Bei den vorangestellten Punkten `...` in Zeile 24 handelt sich um den spread operator. Die Filtermethode `map` und die darauffolgende Methode `toList` liefert eine Liste von `TableRow` Elementen. Die umgebende Liste der Zeilen `children` (Z. 19-30) erwartet jedoch Elemente des Typs `TableRow` und keine Elemente des Typs `List`. Der spread operator ermöglicht alle Elemente der inneren Liste in die äußere Liste einzufügen.[37]

Eine weitere Hilfsmethode `_buildSelectableCell` erstellt Zellen, die anklickbar sind (Z. 39-51). Das Widget `TableRowInkWell` (Z. 41-51) kann in Tabellen verwendet werden, um einen anklickbaren Bereich zu erstellen. Beim Anklicken breitet sich ausgehend von der Position des Klicks ein Tintenklecks aus. Dabei überschreitet der Tintenklecks nicht den Bereich, der von der umgebenden Zeile begrenzt ist. Bei auslösen des Ereignisses `onTap` erfolgt die Ausführung des Callbacks `onSelect` (Z. 44) mit der ausgewählten Maßnahme. Doch zuvor muss überprüft werden, ob der Callback auch initialisiert wurde (Z. 43). Wie hier zu sehen ist, reicht es nicht aus, abzufragen, ob `onSelect` gesetzt ist. Trotzdem erfolgt keine Typ-Beförderung zu einem Typen ohne Null-Zulässigkeit, denn es handelt sich um eine Instanzvariable. Deshalb muss der Suffix `!` gesetz sein (Siehe Grundlagenkapitel 3.2.4 Typen mit Null-Zulässigkeit).

Bei `onSelect` handelt es sich um einen Callback. An diesem Beispiel kann das Inversion of Control Entwurfsmuster visualisiert werden. Abbildung 11 zeigt wie die Akteure zusammenarbeiten. Der `MassnahmenMasterScreen` verwendet die `MassnahmenTable`. Die Tabelle enthält ein Objekt namens `onSelect`. Dabei handelt es sich um einen Funktions-Objekt. Anstatt eine neue Klasse mit einer beinhaltenden Funktion zu deklarieren, kann das gleiche über eine Abkürzung erreicht werden: dem Schlüsselwort `typedef` (Z. 4). Hier erlaubt es eine Funktionssignatur als eigenen Typ zu deklarieren. Der `MassnahmenMasterScreen` wiederum instanziiert genauso so ein Funktions-Objekt als anonyme Funktion (Listing. 35 Z. 75-82). Weil es der Signatur der Typdefinition von `OnSelectCallback` entspricht, kann es der Tabelle als Argument für den Parameter `onSelect` übergeben werden.

[37] Vgl. Google LLC, *Dart - Language tour - spread operator*.

```dart
typedef OnSelectCallback = void Function(Massnahme selectedMassnahme);

class MassnahmenTable extends StatelessWidget {
  final Set<Massnahme> _massnahmenToDisplay;
  final OnSelectCallback? onSelect;

  const MassnahmenTable(this._massnahmenToDisplay, {this.onSelect, Key? key})
      : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Table(
      border: TableBorder.all(width: 3),
      defaultColumnWidth: const IntrinsicColumnWidth(),
      defaultVerticalAlignment: TableCellVerticalAlignment.middle,
      children: [
        TableRow(children: [
          _buildColumnHeader(const Text("Zuletzt bearbeitet am")),
          _buildColumnHeader(const Text("Maßnahmentitel"))
        ]),
        ..._massnahmenToDisplay.map((m) {
          return TableRow(children: [
            _buildSelectableCell(m, Text(m.letzteBearbeitung.formattedDate)),
            _buildSelectableCell(m, Text(m.identifikatoren.massnahmenTitel)),
          ]);
        }).toList(),
      ],
    );
  }

  Widget _buildColumnHeader(Widget child) => Padding(
        padding: const EdgeInsets.all(8.0),
        child: child,
      );

  Widget _buildSelectableCell(Massnahme m, Widget child,
          {double padding = 8.0}) =>
      TableRowInkWell(
        onTap: () {
          if (onSelect != null) {
            onSelect!(m);
          }
        },
        child: Padding(
          padding: EdgeInsets.all(padding),
          child: child,
        ),
      );
}
```

Listing 36: Die Klasse MassnahmenTable, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/widgets/massnahmen_table.dart

Das Inversion of Control Entwurfsmuster ist auch unter dem Namen „hollywood pattern“ bekannt , da es ähnlich wie die typische Antwort auf eine Bewerbung für einen Hollywood Film - don’t call us, we’ll call you - funktioniert.[38]

Und genauso arbeiten der Übersichts-Bildschirm und die Tabelle zusammen. Der Übersichts-Bildschirm verwendet die Tabelle, welche nicht wissen musst, wofür sie eingesetzt wird. Sobald die Tabelle eine Selektion des Benutzers bemerkt, kommuniziert sie wieder mit dem Übersicht Bildschirm. Nun greift der Übersicht-Bildschirm über den Service Locator auf das ViewModel zu, um die selektierte Maßnahme zu übergeben.

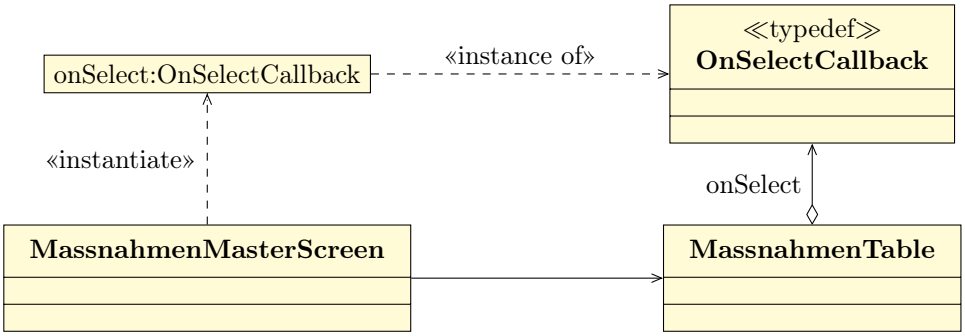


Abbildung 11: UML Diagramm, Quelle: Eigene Abbildung

5.12 Das View Model

Listing 37 Zeig das ViewModel. Im ersten Schritt enthält es nur drei Streams vom Typ `BehaviorSubject` . Eines für den letzten Status (Z. 6), eines für die „guid“ (Z. 8) und eines für den Titel der Maßnahme (Z. 10). Anhand dessen wird offensichtlich, warum ein ViewModel nötig ist. Die Daten, die in der Oberfläche angezeigt werden, sind Streams, die neue Werte annehmen können. Wann immer sich ein Wert ändert, löst der Stream ein neues Ereignis aus. Auf dieses Ereignis kann der View reagieren. Das Model bietet die Eigenschaften der Maßnahmen dagegen nicht als Stream an.

Weil sich ich Model und ViewModel in ihrer Struktur unterscheiden, gibt es zwei Methoden, die die Konvertierung in beide Richtungen vornehmen. Die Setter-Methode `model` (Z. 12-18) erhält ein Objekt des Wert des Typs `Massnahme` - das Format des Models. Die einzelnen Eigenschaften werden dann in das Format des ViewModels umgewandelt: in Streams. Darüber wird der Setter-Methode `value` von jedem `BehaviorSubject` der entsprechende Wert aus dem Model zugewiesen. Besonders das auch, wie die Auswahloptionen sich im Model und ViewModel unterscheiden. Im ViewModel sind es abgeleitete Objekte der Basisklasse Choice, wie z.b. LetzterStatus. Im Gegensatz dazu speichert das Modell die Optionen lediglich über die Abkürzung als String ab. Mit Hilfe der Methode `fromAbbreviation` kann Anhand der Abkürzung wieder das entsprechende Objekt wiedergefunden werden (Z. 16).

Die Getter-Methode dagegen konvertiert in das exakte Gegenteil. Die aktuellen Werte von jedem `BehaviorSubject` werden über die Getter-Methode `value` ausgelesen und anschließend der entsprechenden Eigenschaft des Objektes vom Werte-Typ `Massnahme` gespeichert. Die Auswahloption, die für den letzten Status hinterlegt wurde, wird dabei wiederum nur als Abkürzung eingetragen. Dementsprechend ist bloß die Eigenschaft `abbreviation` abzufragen (Z. 22).

Allerdings kann bei Auswahlfeldern auch keine Option gewählt sein. Die Getter-Methode `value` kann daher also auch `null` zurück geben. Der Compiler gibt einen Fehler aus, wenn versucht wird, auf `value` eine Operation auszuführen, sollte es sich um einen Typ mit

[38] Vgl. Fowler, *InversionOfControl*.

Null-Zulässigkeit handeln. So ist es bei dem Aufruf von `abbreviation` der Fall (Z. 22). Der Fehler kann nur damit behoben werden, indem das Prefix `?` der Operation vorangestellt wird. In diesem Fall wird die Methode aufgerufen, sollte `value` nicht `null` sein. Ist `value` dagegen `null`, so wird die Operation nicht ausgeführt und der gesamte Ausdruck gibt direkt `null` zurück.

```
5  class MassnahmenFormViewModel {
6    final letzterStatus = BehaviorSubject<LetzterStatus?>.seeded(null);
7
8    final guid = BehaviorSubject<String?>.seeded(null);
9
10   final massnahmenTitel = BehaviorSubject<String>.seeded("");
11
12   set model(Massnahme model) {
13     guid.value = model.guid;
14
15     letzterStatus.value = letzterStatusChoices
16         .fromAbbreviation(model.letzteBearbeitung.letzterStatus);
17     massnahmenTitel.value = model.identifikatoren.massnahmenTitel;
18   }
19
20   Massnahme get model => Massnahme((b) => b
21     ..guid = guid.value
22     ..letzteBearbeitung.letzterStatus = letzterStatus.value?.abbreviation
23     ..letzteBearbeitung.letztesBearbeitungsDatum = DateTime.now().toUtc()
24     ..identifikatoren
25         .update((b) => b..massnahmenTitel = massnahmenTitel.value));
26 }
```

Listing 37: Die Klasse MassnahmenFormViewModel, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/screens/massnahmen_detail/massnahmen_form_view_model.dart

5.13 Eingabeformular

Das soeben erstellte ViewModel kann nun für die Eingabemaske verwendet werden. Listing 38 zeig die grundlegende Struktur der Klasse `MassnahmenDetailScreen`.

Wiederum werden das ViewModel und das Model über das „InheritedWidget“ `AppState` abgerufen und in die jeweiligen lokalen Variablen gespeichert (Z. 16, 17). Nachfolgend werden zwei Hilfsfunktionen innerhalb der `build`-Methode deklariert. Solche sogenannten nested functions - deutsch verschachtelten Funktionen - sind im Dart erlaubt, was zu einer weiteren Besonderheit führt. Der Sichtbarkeitsbereich von Variablen ist in Dart lexikalisch. Die Bindung der Variablen ist also durch den umgebenden Quelltext bestimmt. Die lokalen Variablen `model` und `vm` sind also im gesamten Bereich sichtbar, der durch die öffnenden und schließenden geschweiften Klammern der Methode `build` aufgespannt wird (Z. 15-103). Damit sind sie auch innerhalb der beiden verschachtelten Funktionen verfügbar. Innerhalb der Funktionen kann auf model und vm zugegriffen werden, ohne sie über einen Parameter übergeben zu müssen.

Das erste Widget im Inhaltsbereich des Scaffold ist ein `WillPopScope`. Es erlaubt das Verlassen einer Route an eine Abhängigkeit zu knüpfen. Bei dem Eingabeformular handelt es sich um eine Unterseite. Dadurch erscheint in der `AppBar` (Z. 47-48) links von der Überschrift ein Button, der ermöglicht, zur letzten Ansicht zurück zu navigieren (Abb. 8). Dabei stellt sich jedoch die Frage, was mit der bis zu diesem Zeitpunkt eingetragenen Maßnahme passieren soll. Für die Formular-Anwendung soll in diesem Fall die Maßnahme im aktuellen Zustand abgespeichert werden. Dazu wird dem Parameter `onWillPop` als Argument die Funktion `saveRecord`.

Anders als im Übersicht-Bildschirm erhält das `Scaffold` kein Argument für den Parameter `floatingActionButton`. Der Hintergrund dafür ist, dass auf diesem Bildschirm in den nächsten Schritten nicht nur ein, sondern zwei solcher Buttons zur Verfügung stehen sollen. Daher muss der Button manual angelegt werden. Das ist nur mit Hilfe eines `Stack`-Widgets möglich, welcher als Kind des `WillPopScope` eingetragen ist. Ein `Stack` erlaubt es mehrere Ebenen in der Tiefe anzulegen. Das unterste Element soll die Auflistung der Eingabefelder sein. Der `SingleChildScrollView` (Z. 54-79) bietet einen vertikalen Scrollbereich an, in dem die Eingabefelder in einer `Column` (Z. 58-76) untereinander aufgelistet sind. Die Ebene, die über den Eingabefeldern eingeblendet wird, soll die beiden Aktions-Buttons zeichnen. Das Widget `Align` erlaubt in dieser Ebene festzulegen, wo die Elemente angeordnet sein sollen (Z. 80-99). Wie für den `FloatingActionButton` üblich wurde die untere rechte Bildschirm-Ecke gewählt (Z. 81). Die Buttons sollen in Zukunft übereinander angeordnet sein, weshalb ein `Column`-Widget zum Einsatz kommt. Zum ersten Mal taucht der Parameter `mainAxisSizeAuf`. Mit dem Argument `MainAxisSize.min` nimmt die `Column` in der Höhe nur so viel Platz ein, wie durch die Kindelemente notwendig. Als bisher einziges Element in der `Column` taucht nun der `FloatingActionButton` auf (Z. 87-95), der die aktuell eingetragenen Daten abspeichern (Z. 92) und zur Übersicht zurückkehren soll (Z. 93). Wenn der Nutzer den Mauszeiger über diesen Button bewegt, wird ein Tooltip angezeigt: "Validiere und speichere Massnahme"(Z. 88). Der Tooltip ist als Konstante angelegt (Z. 7). Das hat vor allem den Grund, dass er auch für den folgenden Integrationstest genutzt wird. Elemente können darin über einen beinhaltenden Text oder Tooltip gefunden werden.

```
 7 const saveMassnahmeTooltip = "Validiere und speichere Massnahme";
 8
 9 class MassnahmenDetailScreen extends StatelessWidget {
10   static const routeName = '/massnahmen-detail';
11
12   const MassnahmenDetailScreen({Key? key}) : super(key: key);
13
14   @override
15   Widget build(BuildContext context) {
16     final vm = AppState.of(context).viewModel;
17     final model = AppState.of(context).model;
18
19     Future<bool> saveRecord() {
...        // ...
28     }
29
30     Widget createMassnahmenTitelTextFormField() {
...        // ...
44     }
45
46     return Scaffold(
47         appBar: AppBar(
48           title: const Text('Maßnahmen Detail'),
49         ),
50         body: WillPopScope(
51           onWillPop: () => saveRecord(),
52           child: Stack(
53             children: [
54               SingleChildScrollView(
55                 child: Center(
56                   child: Padding(
57                     padding: const EdgeInsets.all(8.0),
58                     child: Column(
...                      // ...
76                     ),
77                   ),
78                 ),
79               ),
80               Align(
81                 alignment: Alignment.bottomRight,
82                 child: Padding(
83                   padding: const EdgeInsets.all(16.0),
84                   child: Column(
85                     mainAxisSize: MainAxisSize.min,
86                     children: [
87                       FloatingActionButton(
88                         tooltip: saveMassnahmeTooltip,
89                         heroTag: 'save_floating_action_button',
90                         child: const Icon(Icons.check, color: Colors.white),
91                         onPressed: () {
92                           saveRecord();
93                           Navigator.of(context).pop();
94                         },
95                       )
96                     ],
97                   ),
98                 ),
99               )
100            ],
101          ),
102        ));
103  }
104 }
```

Listing 38: Die Struktur des Bildschirms MassnahmenDetailScreen, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

5.13.1 Ausgabe der Formularfelder

Listing 39 zeigt die Ausgabe der Formularfelder in einer Column 58. Das Auswahlfeld für den letzten Status verwendet ein selbstgeschriebenes `Widget` namens `SelectionCard` (Z. 61-72). Da die Menge der Auswahloptionen auch den Namen der Liste enthält, kann er als Titel der Selektionskarte verwendet werden (Z. 62) In diesem Fall ist das der Text „Status“. Die Auswahloptionen, welche der Auswahlbildschirm anzeigen soll, sind dem Parameter `allChoices` hinterlegt 63.

Die Selektionskarte soll ihren eigenen Zustand pflegen. Sie erhält dazu lediglich den initialen Wert, der aktuellen im ViewModel gespeichert ist. Bei allen Änderungen, die innerhalb der Selektionskarte erfolgen, sollen die gleichen Änderungen auch im ViewModel nachgepflegt werden. Sollte also der Wert des letzten Status im ViewModel verfügbar sein (Z. 65), so wird er als Startwert dem Parameter `initialValue` (Z. 64-67) übergeben. Dabei ist zu beachten, dass das Argument eine Menge ist. Sie wird mit den öffnenden und schließenden geschweiften Klammern erstellt. Das collection-if wird hier verwendet, um genau ein Element diesem `Set`-Literal hinzuzufügen, sollte es nicht `null` sein. Ist das Element allerdings `null`, so bleibt das `Set`-Literal einfach leer. Für mehr Informationen zum Set-Literal und dem collection-if siehe `Kapitel einfügen`.

Wenn der Benutzer einerAuswahloptionen selektiert, so wird die dementsprechende anonyme Funktion aufgerufen. Sie ist für den Parameter `onSelect` hinterlegt, (Z. 68-69). Das gleiche gilt für Auswahloptionen, welche deselektiert werden (Z. 70-71). Das Auswahlfeld erlaubt nur einen Wert. Deshalb reicht es aus, den Wert bei Selektion zu ersetzen und ihn bei Deselektion zu leeren, also ihn auf `null` zu setzen.

```
58 child: Column(
59   crossAxisAlignment: CrossAxisAlignment.start,
60   children: [
61     SelectionCard<LetzterStatus>(
62       title: letzterStatusChoices.name,
63       allChoices: letzterStatusChoices,
64       initialValue: {
65         if (vm.letzterStatus.value != null)
66           vm.letzterStatus.value!
67       },
68       onSelect: (selectedChoice) =>
69           vm.letzterStatus.value = selectedChoice,
70       onDeselect: (selectedChoice) =>
71           vm.letzterStatus.value = null,
72     ),
73     createMassnahmenTitelTextFormField(),
74     const SizedBox(height: 64)
75   ],
76 ),
```

Listing 39: Die Ausgabe der Formularfelder, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

5.13.2 Eingabefeld für den Maßnahmentitel

Unterhalb der ersten Selektionskarte soll das Eingabefeld für den Maßnahmentitel erscheinen (Z. 73). Listing 40 zeigt die Implementierung der verschachtelten Funktion zum Zeichnen dieses Eingabefeldes. Es handelt sich um das Widget `TextFormField` (Z. 34-41).

Hier wird klar, wovon die Selektionskarte inspiriert ist. Denn auch das `TextFormField` erhält einen initialen Wert über den Parameter `initialValue`. Sobald sich der Wert das Formular-

```
30 Widget createMassnahmenTitelTextFormField() {
31   return Card(
32     child: Padding(
33       padding: const EdgeInsets.all(16.0),
34       child: TextFormField(
35         initialValue: vm.massnahmenTitel.value,
36         decoration: const InputDecoration(
37             hintText: 'Maßnahmentitel', labelText: 'Maßnahmentitel'),
38         onChanged: (value) {
39           vm.massnahmenTitel.value = value;
40         },
41       ),
42     ),
43   );
44 }
```

Listing 40: Die Funktion createMassnahmenTitelTextFormField, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

feld das ändert, kann der neue Wert im ViewModel über die anonyme Funktion aktualisiert werden, welche dem Parameter `onChanged` übergeben wurde.

5.13.3 Speicher-Routine

Die Funktion die dem Parameter `onWillPop` des `WillPopScope` übergeben wurde, ist in Listing 41 zu sehen. Die Voraussetzung für diese Funktion ist, dass ihr Rückgabetyp ein `Future<bool>` ist. Das erlaubt der Methode asynchron zu sein. Der `Future`, der von der Funktion zurückgegeben werden soll, muss in der Zukunft den Wert `true` zurückgeben, wenn den Navigator erlaubt werden soll, zurück zu navigieren. Da die Implementierung der Methode allerdings nicht asynchron ist, soll der Wahrheitswert direkt zurückgegeben werden. Mit dem benannten Konstruktor `value` der Klasse `Future` ist es möglich, genau das zu tun `27`. Der Wahrheitswert ist damit in einem `Future`-Objekt gekapselt und steht ohne Verzögerung zur Verfügung. Aktuell soll die Maßnahme lediglich abgespeichert werden (Z. 25), da noch keine Validierung erfolgt.

Der Benutzer erhält noch eine Mitteilung, dass die Maßnahme erstellt wurde. Das aktuelle `Scaffold`-Objekt kann über `ScaffoldMessenger.of` adressiert werden (Z. 20). Sollte bereits eine Mitteilung vorliegen, wird diese wieder versteckt, um Platz für die neue zu machen (Z. 21). Anschließend wird eine sogenannte `Snackbar` mit dem entsprechenden Text angezeigt (Z. 22-23).

```
19 Future<bool> saveRecord() {
20   ScaffoldMessenger.of(context)
21     ..hideCurrentSnackBar()
22     ..showSnackBar(
23         const SnackBar(content: Text('Massnahme wird gespeichert ...')));
24
25   model.putMassnahmeIfAbsent(vm.model);
26
27   return Future.value(true);
28 }
```

Listing 41: Die Funktion saveRecordAndGoBackToOverviewScreen, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/screens/massnahmen_detail/massnahmen_detail.dart

5.14 Widget SelectionCard

Das Listing 42 zeigt die Struktur des Widgets SelectionCard. Die Klasse hat einen generischen Typparameter (Z. 15). `<ChoiceType extends Choice>` bedeutet, dass die SelectionCard nur für Typen verwendet werden kann, die von `Choice` erben. Das ist eine wichtige Voraussetzung, da auf den übergebenen Werten Operationen ausgeführt werden sollen, die nur `Choice` unterstützt. Alle Parameter, die dem Konstrukt übergeben werden, leiten ebenso von diesen Typparameter ab. Einzige Ausnahme dabei ist der `titel` `16`.

```
7  typedef OnSelect<ChoiceType extends Choice> = void Function(
8      ChoiceType selectedChoice);
9
10 typedef OnDeselect<ChoiceType extends Choice> = void Function(
11     ChoiceType selectedChoice);
12
13 const confirmButtonTooltip = 'Auswahl übernehmen';
14
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 OnSelect<ChoiceType> onSelect;
20   final OnDeselect<ChoiceType> onDeselect;
21
22   SelectionCard(
23       {required this.title,
24       required Iterable<ChoiceType> initialValue,
25       required this.allChoices,
26       required this.onSelect,
27       required this.onDeselect,
28       Key? key})
29       : selectionViewModel = BehaviorSubject<BuiltSet<ChoiceType>>.seeded(
30             BuiltSet.from(initialValue)),
31         super(key: key);
```

Listing 42: Die Klasse SelectionCard, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/widgets/selection_card.dart

Mit dem Stream `selectionViewModel` verwaltet die `SelectionCard` ihren eigenen Zustand. Der Stream ist mit dem generischen Typen `BuiltSet<ChoiceType>` konfiguriert. Das macht es unmöglich, den aktuell hinterlegten Wert anzupassen, ohne das Gesamtobjekt auszutauschen. Der Tausch des Objektes wiederum bewirkt, dass ein Ereignis über den Stream ausgelöst wird. Über dieses Ereignis zeichnet die SelectionCard Teile seiner Oberfläche neu. Allerdings erhält der Konstruktor kein Argument des Typs `BehaviorSubject` sondern stattdessen vom `Iterable<ChoiceType>` (Z. 24). Damit wird der Benutzer nicht darauf eingeschränkt, einen Stream zu übergeben. Er kann auch eine gewöhnliche Liste oder Menge setzen. Die Umwandlung der ankommenden Kollektion erfolgt in der Initialisierungsliste `29-30`. Nur so ist es möglich, die Instanzvariable mit `final` als unveränderbar zu kennzeichnen. Initialisierungen solcher Variablen müssen im statischen Kontext der Objekterstellung geschehen. Der Konstruktor-Körper gehört dagegen nicht mehr zur statischen Teil. Im Konstruktor-Körper können Operationen der Instanz verwendet werden, denn das Objekt existiert bereits. Der Versuch eine mit `final` gekennzeichnete Instanzvariable im Konstruktor-Körper zu setzen, führt zu einem Compilerfehler in Dart. Der Konstruktor `seeded` des `BehaviorSubject` wird mit einem `BuiltSet` gefüllt (Z. 29). Dieses wiederum wird mit dem benannten Konstruktor `from` von `BuiltSet` mit der Kollektion aufgerufen (Z. 30). Er wandelt die Liste in eine unveränderbare Menge um. Die Liste aller Auswahloptionen `allChoices` (Z. 18) gewährleistet über den generischen Typ-Parameter, das nicht aus versehen Auswahloptionen übergeben werden, die nicht zum Typ der `SelectionCard` passen. Die Rückruf-Funktionen (Z. 19, 20) die bei Selektion und Deselektion von Optionen ausgelöst

werden, bieten einen besonderen Vorteil, dadurch, dass sie mit dem generischen Typen konfiguriert sind. Die Signaturen der Rückruf-Funktionen (Z. 7-8, 10-11) geben nämlich vor, dass der erste Parameter vom Typen `ChoiceType` sein musst. Wenn nun der Benutzer der SelectionCard einen Typ wie etwa `LetzterStatus` für den Typparameter übergibt, so erhält er auch eine Rückruffunktion, dessen erster Parameter vom Typ `LetzterStatus` ist. Ohne eine Typumwandlung - englisch type casting - von (Z. Choice) in `LetzterStatus`, können Operationen auf das Objekt angewendet werden, die nur `LetzterStatus` unterstützt.

Das erste Element, welches von der `build`-Methode zurückgeben wird, ist ein `StreamBuilder` (Listing. 43 Z. 47). Er horcht auf das `selectionViewModel` (Z. 48). Sobald also eine Selektion getätigt wurde, aktualisiert sich auch die dazugehörige Karte. Das Aussehen einer Karte wird durch das Widget `Card` erreicht (Z. 51). Dadurch erhält es abgerundete Ecken und einen Schlagschatten, der es vom Hintergrund abgrenzt. Ein `ListTile` Widget erlaubt es dann, den übergebenen `titel` als Überschrift zu setzen (Z. 54) und die aktuell ausgewählten Selektionen als Untertitel anzuzeigen (Z. 56). Zu diesem Zweck wandelt die Methode `map` alle Elemente von `selectedChoices` in `String`-Objekte um, indem es von dem `Choice`-Objekt lediglicht den Beschreibungstext `description` verwendet. Anschließend sammelt der Befehl `join` die resultierende `String`-Objekte ein, formt sie in einen gemeinsamen `String` zusammen und trennt sie darin jeweils mit einem `", "` voneinander.

```
34 Widget build(BuildContext context) {
35   final focusNode = FocusNode();
36
37   navigateToSelectionScreen() async {
38     focusNode.requestFocus();
39
40     Navigator.push(
41         context,
42         MaterialPageRoute(
43             builder: (context) =>
44                 createMultipleChoiceSelectionScreen(context)));
45   }
46
47   return StreamBuilder(
48       stream: selectionViewModel,
49       builder: (context, snapshot) {
50         final selectedChoices = selectionViewModel.value;
51         return Card(
52           child: ListTile(
53             focusNode: focusNode,
54             title: Text(title),
55             subtitle:
56                 Text(selectedChoices.map((c) => c.description).join(", ")),
57             trailing: const Icon(Icons.edit),
58             onTap: navigateToSelectionScreen,
59           ),
60         );
61       });
62 }
```

Listing 43: Die Build Methode der SelectionCard, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/widgets/selection_card.dart

Das `ListTile` erhält ein `FocusNode`-Objekt (Z. 53), damit der Benutzer beim Zurücknavigieren von der Unterseite im Formular wieder in der gleichen vertikalen Position der Karte landet, die er zuvor ausgewählt hat. Der Benutzer würde ansonsten in Formular wieder an der obersten Position herauskommen. Der `FocusNode` wird einmal zu Anfang der `build`-Methode erstellt (Z. 35). Damit ist er außerhalb des `StreamBuilder` und bleibt somit beim Neuzeichnen der Karte erhalten.

Klickt der Benutzer die Karte an, navigiert er schließlich zur Unterseite, wo er die Auswahloptionen präsentiert bekommt. Die verschachtelte Funktion `navigateToSelectionScreen` kommt dafür zum Einsatz (Z. 37-45). Da das Wechseln zu Unterseite bevorsteht, fordert der `focusNode` den Fokus für das angeklickte `ListTile` an (Z. 38). Schließlich navigiert der Benutzer mit `Navigator.push` zur Unterseite. Es handelt sich um den Auswahlbildschirm, auf dem der Benutzer die gewünschte Option anwählen kann. Die Besonderheit dieses Mal: die Route ist nicht als Widget deklariert und wird nicht über einen Namen aufgerufen, so wie es bei dem Übersichtsbildschirm und der Eingabemaske war. Stattdessen baut eine Funktion bei jedem Aufruf die Seite neu gebaut. Das dynamische Bauen der Seite hat einen besonderen Vorteilen, der am Listing 44 erklärt wird.

5.14.1 Bildschirm für die Auswahl der Optionen

Die Funktion `createMultipleChoiceSelectionScreen` (Listing 44) gibt einen `Scaffold` zurück, der die gesamte Seite enthält (Z. 65). Das erste Kind des `Scaffold` ist wiederum ein `StreamBuilder` (Z. 69). Hier wird der Vorteil der dynamischen Erzeugung der Seite offensichtlich: die Unterseite kann das gleiche ViewModel wiederverwenden, welches auch von der `SelectionCard` genutzt wird. Auch alle weiteren Instanzvariablen der `SelectionCard` können wiederverwendet werden. Würde es sich stattdessen um eine weitere Route handeln, so müssten alle diese Informationen über den Navigator zur neuen Unterseite übergeben werden. Sollte der Nutzer die Auswahl beenden, so müsste auch ein Mechanismus für das zurückgeben der selektierten Daten implementiert werden. Dadurch, dass die `SelectionCard` und der Auswahlbildschirm sich das gleiche ViewModel teilen, kann sogar ein weiterer Vorteil in Zukunft genutzt werden: in einem zweispaltigen Layout könnte auf der linken Seite die Eingabemaske und auf der rechten Seite der Bildschirm der Auswahloptionen eingeblendet werden. Sobald sich Auswahloptionen im rechten Auswahl Bildschirm verändern, so würden sich die Änderungen auf der linken Seite für den Benutzer direkt widerspiegeln.

Innerhalb des `StreamBuilder` werden die Auswahloptionen gebaut. Dazu speichert die lokale Variable `selectedChoices` die aktuellen Selektionen des Streams zunächst zwischen (Z. 72). Die Optionen werden in einem ListView präsentiert (Z. 73). Er ermöglicht es, Listen-Elemente in einem vertikalen Scrollbereich darzustellen. Die Funktion `map` konvertiert alle Objekte in der Liste aller möglichen Optionen `choices` in Elemente des Typs `CheckboxListTile` (Z. 74-98). In der Standard-Variante sind die Checkboxen rechtsbündig. Der Parameter `controlAffinity` kann genutzt werden, um dieses Verhalten zu überschreiben (Z. 80).

Das `CheckboxListTile` erhält einen Titel, der aus dem Beschreibungstext `description` des `Choice`-Objektes gebildet wird (Z. 81). Ob eine Option aktuell bereits ausgewählt ist, kann mit dem Parameter `value` übertragen werden (Z. 82). Sollte sich die Selektion ändern, erfolgt die Mitteilung über die Rückruffunktion `onChanged` (Z. 83-94). Der erste Parameter der anonymen Funktion gibt dabei die ausgewählte Selektion an. Eine Fallunterscheidung überprüft zunächst, ob der Parameter `selected` nicht `null` ist, denn sein Parametertyp `bool?` lässt Null-Werte zu. Durch die Typ-Beförderung ist `selected` innerhalb des Körpers der Fallunterscheidung dann vom Typ `bool` (Z. 84-94).

Darin wird zunächst der Zustand des ViewModels der `SelectionCard` aktualisiert. Die `replace`-Methode des Builder-Objektes kann die gesamte Kollektion im `BuiltSet` austauschen, ungeachtet dessen, dass es sich beim Argument selbst nicht um ein `BuiltSet` handelt. Die `replace`-Methode wandelt das Argument dafür automatisch um. Durch Zuweisung des neuen Wertes erhält das ViewModel der `SelectionCard` ein neues Ereignis. Damit wird die `SelectionCard` und der dazugehörige Auswahlbildschirm aktualisiert. Während der Erstellung dieser Arbeit wurde versucht, die SelectionCard als ein `StatefulWidget` zu

erstellen. Mittels `setState` sollte dafür gesorgt werden, das sowohl SelectionCard als auch der Auswahlbildschirm aktualisiert werden. Doch bei diesem Vorgehen zeichnet sich nur die `SelectionCard` neu. Der Auswahlbildschirm bleibt unverändert, denn er wird zwar von der `SelectionCard` gebaut, doch ist er nicht tatsächlich Kind der `SelectionCard`. In Wahrheit ist der Auswahlbildschirm ein Kind von `MaterialApp` - genau wie `MassnahmenMasterScreen` und `MassnahmenDetailScreen`.

Neben dem ViewModel der `SelectionCard` muss jedoch auch das ViewModel der Eingabemaske aktualisiert werden. Mit den Rückruffunktionen `onSelect` (Z. 90) und `onDeselect` (Z. 92) hat die aufrufende Ansicht die Möglichkeit, auf Selektionen zu reagieren.

Schließlich ist noch der `FloatingActionButton` Teil der Unterseite (Z. 99-103). Mit einem Klick darauf gelangt der Benutzer zurück zur Eingabemaske (Z. 100).

```
64   Widget createMultipleChoiceSelectionScreen(BuildContext context) {
65     return Scaffold(
66       appBar: AppBar(
67         title: Text(title),
68       ),
69       body: StreamBuilder(
70           stream: selectionViewModel,
71           builder: (context, snapshot) {
72             final selectedChoices = selectionViewModel.value;
73             return ListView(children: [
74               ...allChoices.map((ChoiceType c) {
75                 bool isSelected = selectedChoices.contains(c);
76
77                 return CheckboxListTile(
78                     key: Key(
79                         "valid choice ${allChoices.name} - ${c.abbreviation}"),
80                     controlAffinity: ListTileControlAffinity.leading,
81                     title: Text(c.description),
82                     value: isSelected,
83                     onChanged: (selected) {
84                       if (selected != null) {
85                         selectionViewModel.value =
86                             selectionViewModel.value.rebuild((b) {
87                           b.replace(isSelected ? [] : [c]);
88                         });
89                         if (selected) {
90                           onSelect(c);
91                         } else {
92                           onDeselect(c);
93                         }
94                       }
95                     });
96               }).toList(),
97             ]);
98           }),
99       floatingActionButton: FloatingActionButton(
100        onPressed: () => Navigator.of(context).pop(),
101        tooltip: confirmButtonTooltip,
102        child: const Icon(Icons.check),
103      ),
104    );
105  }
106 }
```

Listing 44: Die Funktion createMultipleChoiceSelectionScreen, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/lib/widgets/selection_card.dart

5.15 Integrations-Test zum Test der Oberfläche

Ein automatisierter Integrationstest soll verifizieren, dass die Oberfläche wie vorgesehen funktioniert. Der Integrationstest simuliert einen Benutzer, der die Applikation verwendet, um eine Maßnahme einzutragen. Bei Abschluss des Tests soll überprüft werden, ob die eingegebenen Daten mit den Inhalten der Json-Datei übereinstimmen.

Flutter erlaubt über einen eigenen Testtreiber solche Integrationstest durchzuführen. Dabei wird die Applikation zur Ausführung gebracht, und jeder Schritt so visualisiert, wie es bei der Ausführung der realen Applikation der Fall wäre. Der Entwickler hat damit die Möglichkeit, die Eingaben und Interaktionen zu beobachten und gegebenfalls zu bemerken, warum ein Testfall nicht korrekt ausgeführt wird.

Das Ergebnis des Integrationstests soll allerdings nicht mit der tatsächlich geschriebenen Json-Datei überprüft werden. Der Test soll nicht tatsächlich Daten auf der Festplatte speichern. Das würde die Gefahr bergen, das vergangene Eingaben manipuliert werden. Stattdessen soll der Test in einer Umgebung stattfinden, die keine Auswirkung auf die Haupt-Applikation oder zukünftige Tests haben soll. Zu diesem Zweck können sogenannte Mocks genutzt werden. Das Paket „mockito“ erlaubt über Annotationen solche Mocks für die gewünschten Klassen über Quellcode-Generierung zu erstellen.

Integrationstests werden im Ordner integration_test angelegt. Während des Zeitpunkt der Erstellung dieser Arbeit war es in der Standardkonfiguration der Quellcode-Generierung und dem Paket „mockito“ nicht möglich, Mocks auch im integration_test Ordner zu generieren. Lediglich innerhalb des test Ordners, der für die Unit-Tests vorgesehen ist, hat die Annotation generate mocks funktioniert. Zu diesem Fehlverhalten existiert ein entsprechendes Issue im GitHub Repository des Mockito packages. Ref Um das Generieren von Mocks auch für Integrationstest verfügbar zu machen, hat der Autor dieser Arbeit einen entsprechenden Lösungsansatz recherchieren und im Issue beschrieben. Ref

Damit der integration_test Ordner für die Quellcode-Generierung der Mocks integriert wird, muss ein entsprechender Eintrag in der Build-Konfiguration vorgenommen werden. Damit das Paket „source_gen“ die entsprechenden Dateien analysiert, müssen sie in der Rubrik `sources` angegeben werden (Listing. 46 Z. 3-8). Wird der Ordner integration_test darin eingefügt (Z. 8), bezieht „source_gen“ den Ordner in der Quellcode-Generierung mit ein. Zusätzlich dazu muss die Rubrik `generate_for` von dem `mockBuilder` des „mockito“-Pakets (Z. 11-13) um die gleiche Angabe des Ordners ergänzt werden (Z. 13).

```
1  targets:
2    $default:
3      sources:
4        - $package$
5        - lib/$lib$
6        - lib/**.dart
7        - test/**.dart
8        - integration_test/**.dart
9      builders:
10       mockito|mockBuilder:
11         generate_for:
12           - test/**.dart
13           - integration_test/**.dart
```

Listing 45: Initialisierung des Integrations Tests, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/build.yaml

Anschließend kann mit der Annotation and generate mocks (Listing. 46 Z. 20) ein Mock für `MassnahmenJsonFile` angefordert werden. In der Kommandozeile ist `flutter pub run build_runner build` einzugeben, damit der entsprechende Quellcode generiert wird. Mit dem Mock kann der

Integrationstest ausgeführt werden, ohne dass befürchtet werden muss, dass die Json-Datei tatsächlich beschrieben wird. Stattdessen kann darauf gehorcht werden, wenn Operationen auf dem Objekt ausgeführt werden.

```
18 const durationAfterEachStep = Duration(milliseconds: 1);
19
20 @GenerateMocks([MassnahmenJsonFile])
21 void main() {
22   testWidgets('Can fill the form and save the correct json', (tester) async {
23     final binding = IntegrationTestWidgetsFlutterBinding.ensureInitialized()
24         as IntegrationTestWidgetsFlutterBinding;
25     binding.framePolicy = LiveTestWidgetsFlutterBindingFramePolicy.fullyLive;
26
27     final massnahmenJsonFileMock = MockMassnahmenJsonFile();
28     when(massnahmenJsonFileMock.readMassnahmen()).thenAnswer((_) async => {});
```

Listing 46: Initialisierung des Integrations Tests, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/integration_test/app_test.dart

Die Funktion `testWidgets` startet den Test und erhält als ersten Parameter das `tester`-Objekt (Z. 22). Darüber ist die Interaktion mit der Oberfläche während des Tests möglich. In den Zeilen 22 bis 25 wird der Testtreiber initialisiert. Ref. Anschließend wird ein Objekt der generierten Klasse `MockMassnahmenJsonFile` erstellt. Wenn das Model nun während der Applikation versucht, aus der Json-Datei zu lesen, soll der Mock eine leere Liste von Maßnahmen zurückgeben (Z. 28). Dazu wird die entsprechende Methode `when` verwendet. Als erster Parameter wird die Methode `readMassnahmen` des Mocks übergeben. Im darauffolgenden Aufruf `thenAnswer` kann angegeben werden, welche Rückgabe die Methode liefern soll.

Über den `tester` kann mit Hilfe der Methode `pumpWidget` ein beliebiges Widget in der Test-Ausführung konstruiert werden. In diesem Fall ist es die gesamte Applikation, die getestet werden soll. Dementsprechend ist hier erneut der komplette Haupteinstiegspunkt angegeben (Listing 47). Doch der Konstruktor von (Z. MassnahmenModel) erhält dieses Mal nicht das `MassnahmenJsonFile`, sondern den entsprechenden Mock (Z. 31).

```
30 await tester.pumpWidget(AppState(
31     model: MassnahmenModel(massnahmenJsonFileMock),
32     viewModel: MassnahmenFormViewModel(),
33     child: MaterialApp(
34       title: 'Maßnahmen',
35       theme: ThemeData(
36         primarySwatch: Colors.lightGreen,
37         accentColor: Colors.green,
38         primaryIconTheme: const IconThemeData(color: Colors.white),
39       ),
40       initialRoute: MassnahmenMasterScreen.routeName,
41       routes: {
42         MassnahmenMasterScreen.routeName: (context) =>
43             const MassnahmenMasterScreen(),
44         MassnahmenDetailScreen.routeName: (context) =>
45             const MassnahmenDetailScreen()
46       },
47     )));
```

Listing 47: Initialisierung des Widgets für den Integrations Tests, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/integration_test/app_test.dart

Weil während des Integrationstest immer wieder die gleichen Operationen wie das Selektieren einer Selektions-Karte, das Auswählen einer Option, das Anklicken des Buttons zum Akzeptieren der Auswahl und das Füllen eines Eingabefeldes auftauchen, wurden entsprechende Hilfsfunktionen erstellt.

Der Funktion `tabSelectionCard` (Listing 49) benötigt lediglich die Liste der Auswahloptionen `choices`, die ihr hinterlegt ist.

```
49 Future<void> tabSelectionCard(Choices choices) async {
50   final Finder textLabel = find.text(choices.name);
51   expect(textLabel, findsWidgets);
52
53   final card = find.ancestor(of: textLabel, matching: find.byType(Card));
54   expect(card, findsOneWidget);
55
56   await tester.ensureVisible(card);
57   await tester.tap(card);
58   await tester.pumpAndSettle(durationAfterEachStep);
59 }
```

Listing 48: Die Hilfsmethode tabSelectionCard, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/integration_test/app_test.dart

Um Objekte während des Testens in Oberfläche zu finden, stellt die Klasse `Finder` nützliche Funktionalitäten zur Verfügung. `Finder`-Objekte können über Fabrikmethoden des Objekts `find` abgerufen werden.

Fabrikmethoden Bei der Fabrikmethode handelt es sich um ein klassenbasiertes Erzeugungsmuster. Anstatt ein Objekt einer Klasse direkt über einen Konstruktor zu erstellen, erlaubt ein Erzeuger das Objekt zu konstruieren. Dabei entscheidet der Erzeuger darüber, welche Implementierung der Klasse zurückgegeben wird. Der aufrufende Kontext muss die konkrete Klasse dazu nicht kennen.[39] Er arbeitet lediglich mit der Schnittstelle. In diesem Fall ist `find` dieser Erzeuger. Über die Fabrikmethode `text` wird ein `_TextFinder` konstruiert, jedoch über die Schnittstelle `Finder` zurückgegeben. Eine weitere Fabrikmethode ist `ancestor`. Sie gibt einen `_AncestorFinder` zurück, welcher ebenso hinter der Schnittstelle `Finder` versteckt wird. `Ref`. Die Fabrikmethoden werden hier deshalb verwendet, weil sie die Lesbarkeit verbessern. Anstatt `Finder titel = new _TextFinder("Maßnahmentitel")` ist `Finder titel = find.text("Maßnahmentitel")` deutlich leichter zu erfassen.

Um die Selektions-Karten zu finden, wird lediglich der Titel- Text benötigt. Angenommen der Test ruft `tabSelectionCard` mit dem Argument `letzterStatusChoices` auf, so entspricht choices.name dem String `"Status"`. Der Ausdruck `find.text("Status")` lokalisiert den Titel innerhalb der Selektions-Karte (Z. 50).

Die Funktion `expect` erwartet als ersten Parameter einen `Finder` und als zweiten einen sogenannten „Matcher“ (Z. 51). Der Aufruf von `expect` mit dem entsprechenden `Finder`-Objekt und dem Matcher `findsWidgets` verifiziert, dass mindestens ein entsprechendes Text Element gefunden wurde.

Wurde das Text-Element gefunden, so muss noch den Vater gesucht werden, der vom Typ `Card` ist (Z. 53). Das kann mit `find.ancestor` erfolgen. Über den Parameter `of` erhält er den `Finder` des Kind-Elements und der Parameter `matching` erhält als Argument die Voraussetzung, die vom Vater-Objekt erfüllt werden soll, als weiteren `Finder`. `find.byType(Card)` sucht also alle Elemente vom Typ `Card`. `find.ancestor` sucht anschließend alle Entsprechungen, in der eine `Card` ein Vater des `Finder textLabel` ist. Wiederum überprüft die Funktion `expect`, dass die Karte gefunden wurde. Doch dieses Mal muss es genau ein Widget sein, welches mit dem „Matcher“ `findsOneWidget` verifiziert werden kann (Z. 54). Sollte mehr als nur eine Karte gefunden werden, so wäre nicht klar, welche geklickt werden soll.

[39] Vgl. Gamma u. a., *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*, S. 107–116.

Um eine Karte tatsächlich anzuwählen muss sie im sichtbaren Bereich sein. Die Methode „ensureVisible“ scrollt den Bildschirm zur entsprechenden Position, damit die Karte sichtbar ist (Z. 56). Schließlich sorgt `tab` mit dem `Finder` `card` dafür, dass die Karte ausgewählt wird. `pumpAndSettle` (Z. 58) ist eine obligatorische Methode, die nach jeder Aktion durchgeführt werden muss. Sie sorgt dafür, dass der Test so lange pausiert, bis alle Aktionen in der Oberfläche und damit auch alle angestoßenen Animationen vorüber sind. Zusätzlich kann eine Dauer angegeben werden, die darüber hinaus gewartet werden soll.

`tabConfirmButton` funktioniert ähnlich (Listing 49). Das Finden des Buttons ist jedoch einfacher, da es nur einen Button zum Akzeptieren auf jeder Oberfläche gibt. Der Button enthält keinen Text, lässt sich aber auch über seinen Tooltip lokalisieren (Z. 62). Die Hilfsfunktion klickt den Button (Z. 63) und wartet dann erneut auf Vollendung aller angestoßenen Animationen (Z. 64).

```
61 Future<void> tabConfirmButton() async {
62   var confirmChoiceButton = find.byTooltip(confirmButtonTooltip);
63   await tester.tap(confirmChoiceButton);
64   await tester.pumpAndSettle(durationAfterEachStep);
65 }
```

Listing 49: Die Hilfsmethode tabConfirmButton, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/integration_test/app_test.dart

Ist der Integrationstest aktuell in dem Auswahlbildschirm, so sorgt `tabOption` dafür, dass Auswahloptionen gewählt wird (Listing 50). Dazu wird die gewünschte Option dem Parameter `choice` übergeben. Um die Checkbox der Option zu finden, muss jedoch zunächst der Text der Auswahloption gefunden werden (Z. 68). Erst wenn verifiziert wurde, dass auch nur genau ein Label mit diesem Text existiert, läuft der Test weiter (Z. 69).

```
67 Future<void> tabOption(Choice choice, {bool tabConfirm = false}) async {
68   final choiceLabel = find.text(choice.description);
69   expect(choiceLabel, findsOneWidget);
70
71   await tester.ensureVisible(choiceLabel);
72   await tester.tap(choiceLabel);
73   await tester.pumpAndSettle(durationAfterEachStep);
74
75   if (tabConfirm) {
76     await tabConfirmButton();
77   }
78 }
```

Listing 50: Die Hilfsmethode tabOption, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/integration_test/app_test.dart

Ein Klick auf das Text-Label reicht bereits aus, denn damit wird das Vater-Element - das `CheckboxListTile` - ebenfalls getroffen. Der `tester` holt es in den sichtbaren Bereich `71`, klickt es `72` und wartet auf Abschluss aller Animationen (Z. 73). Sollte der optionale Parameter `tabConfirm` auf `true` gesetzt sein (Z. 75), so wird der Auswahlbildschirm anschließend direkt wieder geschlossen, nachdem die Option ausgewählt wurde (Z. 76).

Schließlich kann mit der Hilfsfunktionen `fillTextFormField` ein Formularfeld über dessen Titel gefunden und der entsprechende übergebende Text eingetragen werden (Listing 51). Sie findet das `TextFormField`, indem es zunächst nach dem Titel mit `find.text(title)` und anschließend dessen Vater-Element vom Typ `TextFormField` sucht (Z. 83). Sollte sowohl der Hinweistext als auch der Titel den gleichen Text enthalten, so kann es sein, dass zwei solche Elemente gefunden werden. In Wahrheit ist es aber zwei Mal dasselbe `TextFormField`. Mit `.first` wird lediglich das erste Element geliefert (Z. 85). Nachdem feststeht, dass das Element existiert (Z. 85) und es in den sichtbaren Bereich gescrollt wurde (Z. 87), gibt der

Integrationstest den gewünschten Text in das Eingabefeld ein (Z. 88). Anschließend wird erneut auf Abschluss aller Animationen gewartet (Z. 89).

```
80 Future<void> fillTextFormField(
81     {required String title, required String text}) async {
82   final textFormField = find
83       .ancestor(of: find.text(title), matching: find.byType(TextFormField))
84       .first;
85   expect(textFormField, findsOneWidget);
86
87   await tester.ensureVisible(textFormField);
88   await tester.enterText(textFormField, text);
89   await tester.pumpAndSettle(durationAfterEachStep);
90 }
```

Listing 51: Die Hilfsmethode fillTextFormField, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/integration_test/app_test.dart

Während der Integrationstest startet, öffnet sich als Erstes der Übersichts-Bildschirm. Zunächst wird gewartet, dass alle Widgets korrekt initialisiert wurden (Listing. 52 Z. 92). Es folgt der Klick auf den Button zum Erstellen einen neuen Maßnahme (Z. 95). Dazu wird der Button über den entsprechenden `Key` gefunden (Z. 94). Vor allem jetzt ist das Abwarten mittels `pumpAndSettle` (Z. 96) unablässig, denn es wird auf einen anderen Bildschirm navigiert. Angenommen der Test wartet nicht ab, so würden die Aktionen noch immer auf den Elementen des alten Bildschirms Anwendung finden.

```
92 await tester.pumpAndSettle(durationAfterEachStep);
93
94 var createNewMassnahmeButton = find.byKey(createNewMassnahmeButtonKey);
95 await tester.tap(createNewMassnahmeButton);
96 await tester.pumpAndSettle(durationAfterEachStep);
```

Listing 52: Der Button zum Kreieren einer Maßnahme wird ausgelöst, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/integration_test/app_test.dart

Der Integrationstest öffnet nun den Auswahl-Bildschirm, in dem die Selektions-Karte zum Setzen des letzten Statuses angewählt wird (Listing. 53 Z. 98). Anschließend fällt die Wahl auf die Option für „abgeschlossen“ (Z. 98). Dabei sorgt `tabConfirm: true` für die sofortige Rückkehr zum Eingabeformular nach der Auswahl.

```
98 await tabSelectionCard(letzterStatusChoices);
99 await tabOption(LetzterStatus.fertig, tabConfirm: true);
```

Listing 53: Der letzte Status wird ausgewählt, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/integration_test/app_test.dart

Nachfolgend soll der Test das Eingabefeld für den Maßnahmen-Titel überprüfen (Listing 54). Es erfolgt die Erstellung eines beispielhaften Titels anhand des aktuellen Datums und der aktuellen Uhrzeit (Z. 101, 102). Der erstellte Text dient als Eingabe für das Eingabefeld (Z. 104).

Die nötigen Eingaben sind erfolgt. Daher kann der Test nun den Klick auf den Button zum Speichern simulieren (Listing. **??** Z. 106-108). Dadurch würde in der Anwendung nun das Speichern der Maßnahmen in der Json-Datei erfolgen. Doch da stattdessen ein Mock verwendet wurde, passiert dies nicht. Das Model ruft aber dennoch die entsprechenden Methoden - wie zum Beispiel `saveMassnahmen` - auf. Die Methoden haben nur nicht die ursprüngliche Funktion. Stattdessen protokollieren sie sowohl die Aufrufe, als auch die übergebenen Argumente. Durch die Methode `verify` (Z. 111) kann überprüft werden, ob die entsprechende Methode `saveMassnahmen` ausgeführt wurde. Der „Matcher“ `captureAny`

```dart
final now = DateTime.now();
var massnahmeTitle =
    "Test Maßnahmen ${now.year}-${now.month}-${now.day} ${now.hour}:${now.minute}";
await fillTextFormField(title: "Maßnahmentitel", text: massnahmeTitle);
```

Listing 54: Der Maßnahmentitel wird eingegeben, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/integration_test/app_test.dart

ermöglicht die Überprüfung auf irgendeine Übergabe und stellt die übergebenen Argumente über den Rückgabewert bereit.

```dart
var saveMassnahmeButton = find.byTooltip(saveMassnahmeTooltip);
await tester.tap(saveMassnahmeButton);
await tester.pumpAndSettle(durationAfterEachStep);

var capturedJson =
    verify(massnahmenJsonFileMock.saveMassnahmen(captureAny)).captured.last;

var actualMassnahme = capturedJson['massnahmen'][0] as Map;
actualMassnahme.remove("guid");
actualMassnahme["letzteBearbeitung"].remove("letztesBearbeitungsDatum");

var expectedJson = {
  'letzteBearbeitung': {'letzterStatus': 'fertig'},
  'identifikatoren': {'massnahmenTitel': massnahmeTitle},
};

expect(actualMassnahme, equals(expectedJson));
```

Listing 55: Validierung des Testergebnisses, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/integration_test/app_test.dart

Die Rückgabe ist vom Typ `VerificationResult` und enthält eine Getter-Methode mit dem Namen `captured`. Dabei handelt es sich um eine Liste aller Argumente, die in den vergangenen Aufrufen übergeben wurden. Mit `last` lässt sich auf das Argument des letzten Aufrufes zurückgreifen.

Nun soll sich zeigen, ob das übergebene Argument mit dem erwarteten Wert übereinstimmt. Weil das Ergebnis eine Liste mit lediglich einer Maßnahme ist,soll auch ausschließlich diese Maßnahme verglichen werden. Der Schlüssel `'massnahmen'` greift auf die Liste zurück und der Schlüssel `0` auf die erste und einzige Maßnahme. Die lokalen Variable `actualMassnahme` speichert sie zwischen (Z. 113).

Es ist unklar, welche zufällige guid bei der Erstellung der Maßnahme generiert wurde. Auch der Zeitstempel hinter dem Schlüssel `"letzteBearbeitung"` ist unbekannt. Eine mögliche Lösung wären weitere Mocks, welche die Erstellung der guid und des Datums überwachen und - anstelle einer zufälligen - immer die gleiche Zeichenkette zurückgibt. Es ist jedoch auch möglich, die Vergleiche der guid und des Zeitstempels auszuschließen. Dazu reicht es die entsprechenden Schlüssel-Werte-Paare über die Schlüssel `"guid"` und `"letztesBearbeitungsDatum"` aus der Ergebnis-Hashtabelle zu entfernen (Z. 114-115).

Die lokale Variable `expectedJson` speichert das erwartete Ergebnis der eingegebenen Maßnahme (Z. 117-120). Die Methode `expect` und der „Matcher“ `equals` überprüfen beide Objekte auf Gleichheit (Z. 122).

Der Befehl `flutter test integration_test/app_test.dart` startet den Test. Die App öffnet sich und der Ausführung des Tests kann zugesehen werden Punkt am Endeerfolgt in dem Terminal die Ausgabe des Ergebnisses: `All tests passed!`

```
class MassnahmenFormViewModel {
    final letzterStatus = BehaviorSubject<LetzterStatus?>.seeded(null);
    // ...
}
```

Listing 56: Live Template für die Erstellung von built_value Boilerplate-Code in Android Studio, Quelle: Jetbrains Marketplace Built Value Snippets Plugin

```
class ChoiceChangeNotifier extends ChangeNotifier {
    BuiltSet<Choice> _choices = BuiltSet<Choice>();

    BuiltSet<Choice> get choices => _choices;

    set choices(BuiltSet<Choice> choices) {
        _choices = choices;
        notifyListeners();
    }
}
class LetzterStatusViewModel extends ChoiceChangeNotifier {}
```

Listing 57: Live Template für die Erstellung von built_value Boilerplate-Code in Android Studio, Quelle: Jetbrains Marketplace Built Value Snippets Plugin

```
providers: [
  Provider<MassnahmenFormViewModel>(
      create: (_) => MassnahmenFormViewModel()),
  Provider<MassnahmenJsonFile>(create: (_) => MassnahmenJsonFile()),
  Provider(
      create: (context) => MassnahmenPool(
          Provider.of<MassnahmenJsonFile>(context, listen: false))),
  ChangeNotifierProvider(create: (context) => LetzterStatusViewModel()))
],
```

Listing 58: Live Template für die Erstellung von built_value Boilerplate-Code in Android Studio, Quelle: Jetbrains Marketplace Built Value Snippets Plugin

```
Consumer<ConsumerType>(
    builder: (context, choiceChangeNotifier, child) {
      final selectedChoices = choiceChangeNotifier.choices;
      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,
            )
          ],
        ),
      );
    },
  )
```

Listing 59: Live Template für die Erstellung von built_value Boilerplate-Code in Android Studio, Quelle: Jetbrains Marketplace Built Value Snippets Plugin

Teil III

Anhang

A Technologiewahl Anhang

A.1 Stimmen verwendeter Frameworks

Auf diesen Anhang verweisen!

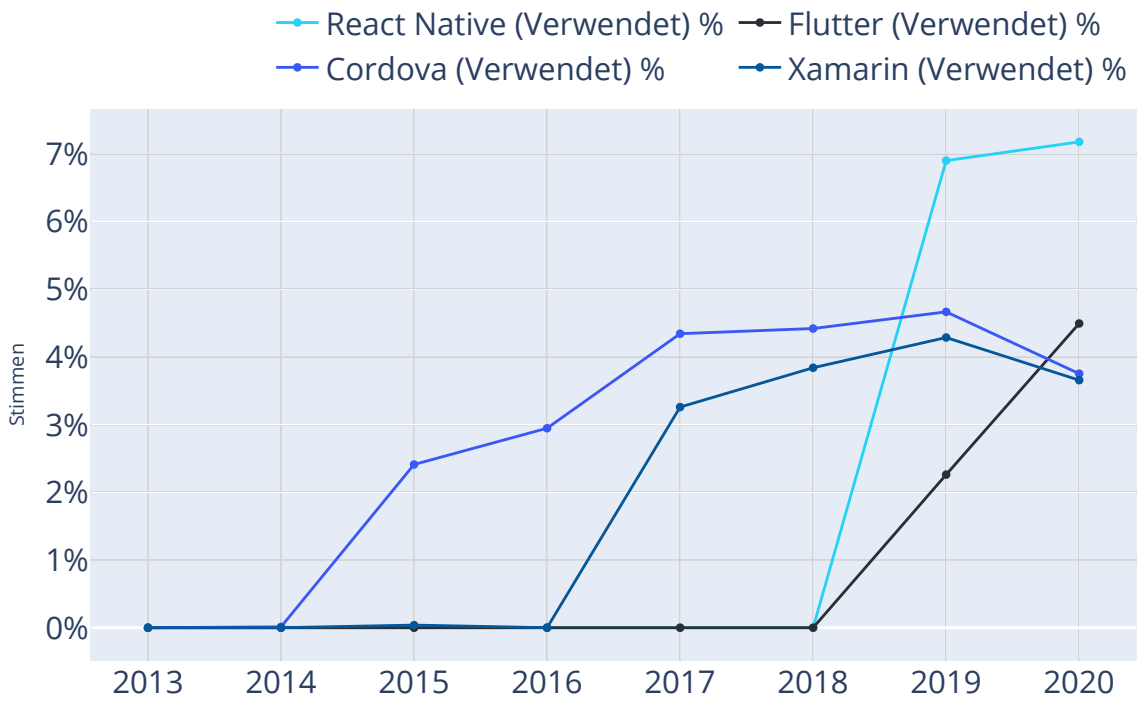


Abbildung 12: Stimmen verwendeter Frameworks, Quelle: Eigene Abbildung, Notebook: Charts/StackOverflowUmfrage/StackOverflowUmfrage.ipynb, Daten-Quelle: FEHLT!

A.2 Stimmen gewünschter Frameworks

Auf diesen Anhang verweisen!

B Vergleich React Native und Flutter Anhang

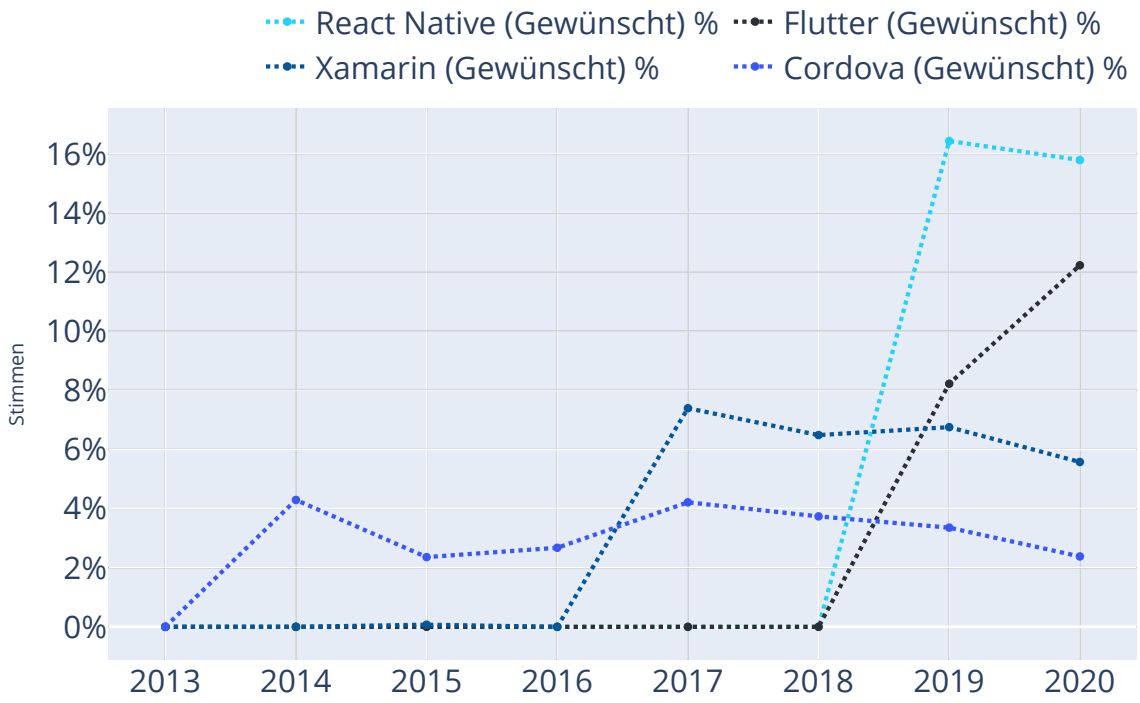


Abbildung 13: Stimmen gewünschter Frameworks, Quelle: Eigene Abbildung, Notebook: `Charts/StackOverflowUmfrage/StackOverflowUmfrage.ipynb`, Daten-Quelle: FEHLT!

```dart
import 'package:flutter/material.dart';
import 'validation.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) => MaterialApp(
        home: Scaffold(
          body: MyCustomForm(),
        ),
      );
}

class MyCustomForm extends StatefulWidget {
  @override
  MyCustomFormState createState() => MyCustomFormState();
}

class MyCustomFormState extends State<MyCustomForm> {
  final _formKey = GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) => Form(
        key: _formKey,
        child: Padding(
          padding: const EdgeInsets.all(8.0),
          child: Column(
            children: [
              TextFormField(
                  decoration: const InputDecoration(labelText: "Name"),
                  validator: (String? value) =>
                      validateNotEmpty("Name", value)),
              TextFormField(
                  decoration: const InputDecoration(labelText: "Email"),
                  validator: (String? value) => validateEmail("Name", value)),
              TextFormField(
                  decoration: const InputDecoration(labelText: "Password"),
                  validator: (String? value) =>
                      validateNotEmpty("Password", value)),
              Padding(
                padding: const EdgeInsets.symmetric(vertical: 16.0),
                child: ElevatedButton(
                  onPressed: () {
                    if (_formKey.currentState!.validate()) {
                      ScaffoldMessenger.of(context).showSnackBar(
                          SnackBar(content: Text('Processing Data')));
                    }
                  },
                  child: Text('Submit'),
                ),
              ),
            ],
          ),
        ),
      );
}
```

Listing 60: , Quelle: Eigenes Listing, Datei: Quellcode/Vergleich/form-in-flutter/lib/main.dart

```dart
final emailPattern = RegExp(
    r'^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\. _⌋
    ↪  [0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$');

String? validateEmail(String label, String? value) {
  if (value == null || value.isEmpty) {
    return '$label is required';
  } else if (!emailPattern.hasMatch(value)) {
    return 'Invalid Email Format';
  } else {
    return null;
  }
}

String? validateNotEmpty(String label, String? value) {
  if (value == null || value.isEmpty) {
    return '$label is required';
  } else {
    return null;
  }
}
```

Listing 61: , Quelle: Eigenes Listing, Datei: Quellcode/Vergleich/form-in-flutter/lib/validation.dart

```tsx
import * as React from 'react';
import {
  Text,
  View,
  StyleSheet,
  Button,
  Alert,
  ScrollView,
} from 'react-native';
import { KeyboardAwareScrollView } from 'react-native-keyboard-aware-scroll-view';

import Constants from 'expo-constants';
import { useForm } from 'react-hook-form';

// You can import from local files
import Input from './components/Input';
import Form from './components/Form';
import validation from './validation';
import Hero from './Hero';

type FormData = {
  name: string;
  email: string;
  password: string;
};

export default () => {
  const { handleSubmit, register, setValue, errors } = useForm<FormData>();

  const onSubmit = (data: FormData) => {
    Alert.alert('data', JSON.stringify(data));
  };

  return (
    <KeyboardAwareScrollView
      contentContainerStyle={styles.container}
      style={{ backgroundColor: '#181e34' }}>
      <Hero />
      <View style={styles.formContainer}>
        <Form {...{ register, setValue, validation, errors }}>
          <Input name="name" label="Name " />
          <Input name="email" label="Email" />
          <Input name="password" label="Password" secureTextEntry={true} />
          <Button title="Submit" onPress={handleSubmit(onSubmit)} />
        </Form>
      </View>
    </KeyboardAwareScrollView>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    paddingTop: Constants.statusBarHeight,
    backgroundColor: '#181e34',
  },
  formContainer: {
    padding: 8,
    flex: 1,
  },
  button: {
    backgroundColor: 'red',
  },
});
```

Listing 62: , Quelle: https://dev.to/elaziziyoussouf/forms-in-react-native-the-right-way-4d46, Datei: Quellcode/Vergleich/form-in-react-native-the-right-way/App.tsx

```
import * as React from 'react';
import { Text, View, StyleSheet, Image } from 'react-native';

export default () => {
    return (
      <View style={styles.container}>
       <Image style={styles.logo} source={require('./assets/hero.jpg')} />
        <Text style={styles.paragraph}>
          Form in React Native, The right Way!
        </Text>
      </View>
    );
  }

const styles = StyleSheet.create({
  container: {
    justifyContent: 'center',
    flex:1,
  },
  paragraph: {
    margin: 24,
    marginTop: 0,
    fontSize: 34,
    fontWeight: 'bold',
    textAlign: 'center',
    color:'#FFF'
  },
  logo: {
   width:'100%',
   height:200
  }
});
```

Listing 63: , Quelle: https://dev.to/elaziziyoussouf/forms-in-react-native-the-right-way-4d46, Datei: Quellcode/Vergleich/form-in-react-native-the-right-way/Hero.tsx

```
export default {
  name: {required: {value: true, message: 'Name is required'}},
  email: {
    required: {value: true, message: 'Email is required'},
    pattern: {
      value: /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@ _|
      ↪  ((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\. _|
      ↪  )+[a-zA-Z]{2,}))$/,
      message: 'Invalid Email Format',
    },
  },
  password: {
    required: {value: true, message: 'Password is required'},
  },
};
```

Listing 64: , Quelle: https://dev.to/elaziziyoussouf/forms-in-react-native-the-right-way-4d46, Datei: Quellcode/Vergleich/form-in-react-native-the-right-way/validation.tsx

```tsx
import * as React from 'react';
import { TextInput, KeyboardAvoidingView, findNodeHandle } from 'react-native';
import { ValidationOptions, FieldError } from 'react-hook-form';

interface ValidationMap {
  [key: string]: ValidationOptions;
}

interface ErrorMap {
  [key: string]: FieldError | undefined;
}

interface Props {
  children: JSX.Element | JSX.Element[];
  register: (
    field: { name: string },
    validation?: ValidationOptions
  ) => void;
  errors: ErrorMap;
  validation: ValidationMap;
  setValue: (name: string, value: string, validate?: boolean) => void;
}

export default ({
  register,
  errors,
  setValue,
  validation,
  children,
}: Props) => {
  const Inputs = React.useRef<Array<TextInput>>([]);

  React.useEffect(() => {
    (Array.isArray(children) ? [...children] : [children]).forEach((child) => {
      if (child.props.name)
        register({ name: child.props.name }, validation[child.props.name]);
    });
  }, [register]);

  return (
    <>
      {(Array.isArray(children) ? [...children] : [children]).map(
        (child, i) => {
          return child.props.name
            ? React.createElement(child.type, {
                ...{
                  ...child.props,
                  ref: (e: TextInput) => {
                    Inputs.current[i] = e;
                  },
                  onChangeText: (v: string) =>
                    setValue(child.props.name, v, true),
                  onSubmitEditing: () => {
                    Inputs.current[i + 1]
                      ? Inputs.current[i + 1].focus()
                      : Inputs.current[i].blur();
                  },
                  //onBlur: () => triggerValidation(child.props.name),
                  blurOnSubmit: false,
                  //name: child.props.name,
                  error: errors[child.props.name],
                },
              })
            : child;
        }
      )}
    </>
  );
};
```


Listing 65: , Quelle: https://dev.to/elaziziyoussouf/forms-in-react-native-the-right-way-4d46, Datei: Quellcode/Vergleich/form_in_react_native_the_right_way/components/Form.tsx

```tsx
import * as React from 'react';
import {
  View,
  TextInput,
  Text,
  StyleSheet,
  ViewStyle,
  TextStyle,
  TextInputProps,
} from 'react-native';
import { FieldError } from 'react-hook-form';
interface Props extends TextInputProps {
  name: string;
  label?: string;
  labelStyle?: TextStyle;
  error?: FieldError | undefined;
}

export default React.forwardRef<any, Props>(
  (props, ref): React.ReactElement => {
    const { label, labelStyle, error, ...inputProps } = props;

    return (
      <View style={styles.container}>
        {label && <Text style={[styles.label, labelStyle]}>{label}</Text>}
        <TextInput
          autoCapitalize="none"
          ref={ref}
          style={[styles.input, { borderColor: error ? '#fc6d47' : '#c0cbd3' }]}
          {...inputProps}
        />
        <Text style={styles.textError}>{error && error.message}</Text>
      </View>
    );
  }
);

const styles = StyleSheet.create({
  container: {
    marginVertical: 8,
  },
  input: {
    borderStyle: 'solid',
    borderWidth: 1,
    borderRadius: 5,
    paddingVertical: 5,
    paddingLeft: 5,
    fontSize: 16,
    height: 40,
    color: '#c0cbd3',
  },
  label: {
    paddingVertical: 5,
    fontSize: 16,
    fontWeight: 'bold',
    color: '#c0cbd3',
  },
  textError: {
    color: '#fc6d47',
    fontSize: 14,
  },
});
```

Listing 66: , Quelle: https://dev.to/elaziziyoussouf/forms-in-react-native-the-right-way-4d46, Datei: Quellcode/Vergleich/form-in-react-native-the-right-way/components/Input.tsx

C 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. Dentsprechend 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 67: Ein automatisierter Testfall überprüft, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-1/conditional_form/test/data_model/storage_test.dart

D Schritt 2 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", "Erschwernisausgleich");
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
         ↪  Vertragsnaturschutz");
13   static final twm_ziel = FoerderklasseChoice(
14       "twm_ziel", "Tierschutz/Tierwohlmaßnahmen mit diesem als Hauptziel");
15   static final contact =
16       FoerderklasseChoice("contact", "bitte um Unterstützung");
17
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 68: Die Klasse FoerderklasseChoice, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-2/conditional_form/lib/choices/choices.dart

```dart
class ZielflaecheChoice extends Choice {
  static final ka = ZielflaecheChoice("ka", "keine Angabe/Vorgabe");
  static final al = ZielflaecheChoice("al", "AL");
  static final gl = ZielflaecheChoice("gl", "GL");
  static final lf = ZielflaecheChoice("lf", "LF");
  static final dk_sk = ZielflaecheChoice("dk_sk", "DK/SK");
  static final hff = ZielflaecheChoice("hff", "HFF");
  static final biotop_le =
      ZielflaecheChoice("biotop_le", "Landschaftselement/Biotop o.Ä.");
  static final wald = ZielflaecheChoice("wald", "Wald/Forst");
  static final contact = ZielflaecheChoice("contact", "bitte um Unterstützung");

  ZielflaecheChoice(String abbreviation, String description)
      : super(abbreviation, description);
}

final zielflaecheChoices = Choices<ZielflaecheChoice>({
  ZielflaecheChoice.ka,
  ZielflaecheChoice.al,
  ZielflaecheChoice.gl,
  ZielflaecheChoice.lf,
  ZielflaecheChoice.dk_sk,
  ZielflaecheChoice.hff,
  ZielflaecheChoice.biotop_le,
  ZielflaecheChoice.wald,
  ZielflaecheChoice.contact
}, name: "Zielfläche");

class ZieleinheitChoice extends Choice {
  static final ka = ZieleinheitChoice("ka", "keine Angabe/Vorgabe");
  static final m3 = ZieleinheitChoice("m3", "m³ (z.B. Gülle)");
  static final pieces =
      ZieleinheitChoice("pieces", "Kopf/Stück (z.B. Tiere oder Bäume)");
  static final gve = ZieleinheitChoice("gve", "GV/GVE");
  static final rgve = ZieleinheitChoice("rgve", "RGV");
  static final ha = ZieleinheitChoice("ha", "ha");
  static final contact = ZieleinheitChoice("contact", "bitte um Unterstützung");

  ZieleinheitChoice(String abbreviation, String description)
      : super(abbreviation, description);
}

final zieleinheitChoices = Choices<ZieleinheitChoice>({
  ZieleinheitChoice.ka,
  ZieleinheitChoice.m3,
  ZieleinheitChoice.pieces,
  ZieleinheitChoice.gve,
  ZieleinheitChoice.rgve,
  ZieleinheitChoice.ha,
  ZieleinheitChoice.contact
}, name: "Zieleinheit");
```

Listing 69: Die Menge foerderklasseChoices, 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 strutktviel =
120       ZielsetzungLandChoice("strutktviel", "Erhöhung der Strukturvielfalt");
121   static final genet_res = ZielsetzungLandChoice("genet_res",
122       "Erhaltung genetischer Ressourcen (Pflanzen, z. B. im Grünland, und Tiere, z. B.
        ↪  bedrohte Rassen)");
123   static final tsch = ZielsetzungLandChoice(
124       "tsch", "Tierschutz/Maßnahmen zum Tierwohl im Betrieb");
125   static final klima = ZielsetzungLandChoice("klima", "Klima");
126   static final contact =
127       ZielsetzungLandChoice("contact", "bitte um Unterstützung");
128
129   ZielsetzungLandChoice(String abbreviation, String description)
130       : super(abbreviation, description);
131 }
132
133 final _zielsetzungLandChoices = {
134   ZielsetzungLandChoice.ka,
135   ZielsetzungLandChoice.bsch,
136   ZielsetzungLandChoice.wsch,
137   ZielsetzungLandChoice.asch,
138   ZielsetzungLandChoice.biodiv,
139   ZielsetzungLandChoice.strutktviel,
140   ZielsetzungLandChoice.genet_res,
141   ZielsetzungLandChoice.tsch,
142   ZielsetzungLandChoice.klima,
143   ZielsetzungLandChoice.contact
144 };
145
146 final hauptzielsetzungLandChoices = Choices<ZielsetzungLandChoice>(
147     _zielsetzungLandChoices,
148     name: "Hauptzielsetzung Land");
```

Listing 70: Die Klasse KategorieChoice, Quelle: Eigenes Listing, Datei: Quellcode/Schritt-2/conditional_form/lib/choices/choices.dart

E Schritt 7 Anhang

Ich wurde zwischengeparkt Weil es auch möglich sein soll, das eine Selektions-Karte nicht nur direkt in der Eingabemaske sondern auch als Kind einer Option anderen Elementes auftaucht, ist ein optionaler Parameter ancestor hinterlegt 49. Ich wurde zwischengeparkt

Ich wurde zwischengeparkt Ist aber die selektions Karte Kind eines anderen Elementes so soll nur nach Elementen besucht werden, die Kind Elemente des angegebenen Vater Elementes sind. Dies kann mit feiner. Descendant erfolgen. Dazu wird das Vater Element dem Parameter of übergeben. Der Parameter matching erhält wiederum ein Feinde Objekt für das Kindelement. In diesem Fall ist dies erneut find. Text, welcher nach dem Titel der Selektion Skate sucht. Ich wurde zwischengeparkt

Ich wurde zwischengeparkt Um nun das Vater-Element zu finden wird nicht wie zuvor fein. Enzersdorf verwendet. Während der Erstellung dieser Arbeit wurde versucht lediglich Feinde Objekte zu benutzen. Doch je häufiger ein feinder Objekt in einem anderen verschachtelt wird, desto länger dauert die Suche nach dem gewünschten Element. Ohne Optimierungen dauerte das Finden des Elementes daher mitunter mehrere Sekunden. Deshalb wurde versucht so häufig es nur geht Alternativen zugfinder Objekten zu verwenden. So kann etwa mittels Methode element und dem Feinde object choice Label das tatsächliche visuelle Objekt gefunden werden. mittels Methode findAncestorWidgetOfExactType Kann über die Vater Elemente des Elementes iteriert werden. Sobald das elements mit dem gewünschten typ CheckboxListTile Gefunden wurde 80, speichert die lokale Variable listtilekey 78 den dem Element hinterlegten key ab. Der Hintergrund dafür ist, dass Methoden wie expekt, Tab, in schwissel und so weiter nur mit Feinde Objekten funktionieren. die Methode bikee konvertiert den Schlüssel in einen fein da, der ein Element über den Schlüssel sucht. sucht.de ist als Suche nach allen existierenden Textelementen und die anschließende Suche von checkboxlist Teil 1 Enten, die diesen Text enthalten. Mithilfe dieser Optimierung konnten die Tests in ihrer Laufzeit Geschwindigkeit deutlich verbessert werden.

Nachdem überprüft wurde, dass von dem listtile genau ein Element existiert 84, wird es in den sichtbaren Bereich gerückt 88, und anschließend angeklickt 87, sowie auf abschließen alle Animationen gewartet 88. Ich wurde zwischengeparkt

Ich wurde zwischengeparkt Damit jedoch auch Elemente innerhalb von des List teils gefunden werden können, wird der feinder, der nach dem key das ist teils sucht zurückgegeben, damit er gegebenenfalls wieder verwendet werden kann. Ich wurde zwischengeparkt

Literatur

<Formik /> | Formik Docs API. Juni 2021. URL: https://web.archive.org/web/20210409184616if_/https://formik.org/docs/api/formik (Zitiert auf der Seite 26).

Adobe Inc. *FAQ | PhoneGap Docs*. Aug. 2016. URL: https://web.archive.org/web/20200806024626/http://docs.phonegap.com/phonegap-build/faq/ (Zitiert auf der Seite 22).

– *Update for Customers Using PhoneGap and PhoneGap Build*. Aug. 2020. URL: https://web.archive.org/web/20200811121213/https://blog.phonegap.com/update-for-customers-using-phonegap-and-phonegap-build-cc701c77502c?gi=df435eca31bb (Zitiert auf der Seite 23).

Borenkraout, Matan. *Native Testing Library Introduction | Testing Library Docs*. Nov. 2020. URL: https://web.archive.org/web/20210128142719/https://testing-library.com/docs/react-native-testing-library/intro/ (Zitiert auf der Seite 27).

Does redux-form work with React Native? Juni 2021. URL: https://web.archive.org/web/20210602234346/https://redux-form.com/7.3.0/docs/faq/reactnative.md/ (Zitiert auf der Seite 25).

Facebook Inc. *The React Native Ecosystem*. Juni 2021. URL: https://web.archive.org/web/20210602191504/https://github.com/facebook/react-native/blob/d48f7ba748a905818e8c64fe70fe5b24aa098b05/ECOSYSTEM.md (Zitiert auf der Seite 27).

Fowler, Martin. *Inversion of Control Containers and the Dependency Injection pattern*. Jan. 2004. URL: http://web.archive.org/web/20210707041912/https://martinfowler.com/articles/injection.html (Zitiert auf der Seite 43).

– *InversionOfControl*. Juni 2005. URL: http://web.archive.org/web/20050628234825/https://martinfowler.com/bliki/InversionOfControl.html (Zitiert auf der Seite 56).

Gamma, Erich u. a. *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Pearson Deutschland GmbH, 2009 (Zitiert auf den Seiten 37, 68).

Google LLC. *Build a form with validation*. Juni 2021. URL: https://web.archive.org/web/20210122020924/https://flutter.dev/docs/cookbook/forms/validation (Zitiert auf der Seite 26).

– *Dart - Language tour - spread operator*. Juli 2021. URL: https://web.archive.org/web/20210625070139/https://dart.dev/guides/language/language-tour#spread-operator (besucht am 08. 07. 2021) (Zitiert auf der Seite 54).

– *Dart Programming Language Specification 5th edition*. Apr. 2021. URL: https://web.archive.org/web/20210702071617/https://dart.dev/guides/language/specifications/DartLangSpec-v2.10.pdf (Zitiert auf den Seiten 30, 31, 35).

– *Desktop support for Flutter*. Juni 2021. URL: https://web.archive.org/web/20210531034514/http://flutter.dev/desktop/ (Zitiert auf der Seite 11).

– *Flutter - Beautiful native apps in record time*. Juni 2021. URL: https://web.archive.org/web/20210630233338/https://flutter.dev/ (Zitiert auf der Seite 12).

– *Flutter - Introduction to widgets*. Juni 2021. URL: http://web.archive.org/web/20210603081649/https://flutter.dev/docs/development/ui/widgets-intro (Zitiert auf der Seite 12).

– *Forms | Flutter Docs Cookbook*. Juni 2021. URL: https://web.archive.org/web/20201102003629/https://flutter.dev/docs/cookbook/forms (Zitiert auf der Seite 26).

– *Häufig gestellte Fragen zu Google Trends-Daten - Google Trends-Hilfe*. Mai 2021. URL: https://support.google.com/trends/answer/4365533 (Zitiert auf der Seite 21).

– *Web support for Flutter*. Mai 2021. URL: http://web.archive.org/web/20210506012158/https://flutter.dev/web (Zitiert auf der Seite 12).

Gosling, James u. a. *The Java® Language Specification Java SE 16 Edition*. Feb. 2021. URL: https://web.archive.org/web/20210514051033/https://docs.oracle.com/javase/specs/jls/se16/jls16.pdf (Zitiert auf den Seiten 30, 31).

Gossman, John. *Introduction to Model/View/ViewModel pattern for building WPF apps*. Okt. 2005. URL: https://web.archive.org/web/20101103111603/http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx (Zitiert auf der Seite 42).

Johnson, Ralph E und Brian Foote. „Designing reusable classes“. In: *Journal of object-oriented programming* 1.2 (1988), S. 22–35 (Zitiert auf der Seite 43).

Lynch, Max. *The Last Word on Cordova and PhoneGap*. März 2014. URL: https://web.archive.org/web/20210413012559/https://blog.ionicframework.com/what-is-cordova-phonegap/ (Zitiert auf der Seite 22).

React Native | Formik Docs. Juni 2021. URL: https://web.archive.org/web/20210507005917if_/https://formik.org/docs/guides/react-native (Zitiert auf der Seite 25).

React Native | React Hook Form - Get Started. Juni 2021. URL: https://web.archive.org/web/20210523042601if_/https://react-hook-form.com/get-started/ (Zitiert auf der Seite 25).

reduxForm | Redux Form - API. Juni 2021. URL: https://web.archive.org/web/20210506221401/https://redux-form.com/7.4.2/docs/api/reduxform.md/#-code-validate-values-object-props-object-gt-errors-object-code-optional- (Zitiert auf der Seite 26).

register | React Hook Form - API. Juni 2021. URL: https://web.archive.org/web/20210406032209/https://react-hook-form.com/api/useform/register (Zitiert auf der Seite 26).

Spolsky, Joel. „How Hard Could It Be?: The Unproven Path“. In: *inc.com* (Nov. 2008). URL: http://web.archive.org/web/20081108094045/http://www.inc.com/magazine/20081101/how-hard-could-it-be-the-unproven-path.html (Zitiert auf der Seite 21).

Stack Exchange, Inc. *Stack Overflow Insights - Developer Hiring, Marketing, and User Research*. Mai 2021. URL: https://insights.stackoverflow.com/survey/ (Zitiert auf den Seiten 21, 22).