

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 Auswahlmöglichkeiten mitunter zahlreich sein. Dem Nutzer sollen daher nur solche Auswahlmöglichkeiten angeboten werden, die zusammen mit der zuvor getroffenen Auswahl sinnvoll sind.

Im Wesentlichen ergibt sich die Kompatibilität der Auswahlmöglichkeiten 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.

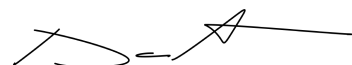
Die Masterarbeit umfasst folgende Teilaufgaben:

- 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	5
Tabellenverzeichnis	7
Listingverzeichnis	9
I. Vorbereitung	11
II. Implementierung	13
1. Schritt 5	15
2. Schritt 5 neu	19
III. Anhang	23
Eidesstattliche Erklärung	25

Abbildungsverzeichnis

1.1. Das <i>Card-Widget</i> wird einmal neu gezeichnet	17
1.2. Das <i>Card-Widget</i> wird zweimal mal neu gezeichnet	18
1.3. Das <i>Card-Widget</i> wird siebenmal neu gezeichnet	18
2.1. Das <i>Card-Widget</i> wird sechsmal neu gezeichnet	19
2.2. Das <i>Card-Widget</i> wird einmal neu gezeichnet	21
2.3. Das <i>Card-Widget</i> wird zweimal neu gezeichnet	22

Tabellenverzeichnis

Listingverzeichnis

1.1. Schritt 5 Der <i>Stream validityChanged</i> in Schritt 5	17
2.1. Schritt 4 Der Klassenvariable <i>ha</i> des Typs <i>ZielflaecheChoice</i> wird eine Be- dingung hinzugefügt	20
2.2. Schritt 5 Der <i>Stream validityChanged</i> in Schritt 5	21

Teil I

VORBEREITUNG

Teil II

IMPLEMENTIERUNG

1. Schritt 5

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

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

Das Problem kann folgendermaßen entschärft werden: Noch bevor das *Widget* `SelectionCard` den `StreamBuilder` in der `build`-Methode zurückgibt, wird ein neuer *Stream* namens `validityChanged` erstellt (Listing 2.2, S. 21, Z. 51-54).

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

Formularen nicht kompatibel.

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

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

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

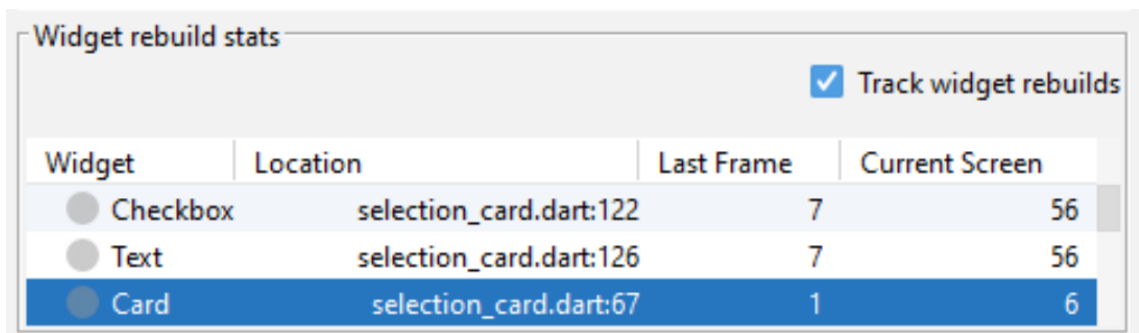
Dieses Verhalten kann auch bei Ausführung der Applikation im Debugmodus in *Android Studio* beobachtet werden. Der *Flutter Performance*-Tab gibt eine Übersicht über die Anzahl der im letzten *Frame* neu gezeichneten *Widgets* (Abb. 1.1). Angenommen für die *Förderklasse* ist *Agrarumwelt-(und Klima)Maßnahme: nur Vertragsnaturschutz* und für die *Kategorie* ist *Düngemanagement* ausgewählt. Wenn nun für die *Förderklasse* die Option *Agrarumwelt-(und Klima)Maßnahmen, tw. auch mit Tierwohlaspekten, aber OHNE Vertragsnaturschutz* selektiert wird, so ist im *Flutter Performance*-Tab zu beobachten, dass


```

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

```

Listing 1.1.: Der *Stream* *validityChanged* in Schritt 5, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-5/conditional_form/lib/widgets/selection_card.dart](#)



Widget	Location	Last Frame	Current Screen
Checkbox	selection_card.dart:122	7	56
Text	selection_card.dart:126	7	56
Card	selection_card.dart:67	1	6

Abbildung 1.1.: Das *Card*-Widget wird einmal neu gezeichnet, Quelle: Eigene Abbildung

das *Widget* `Card` nur einmal neu gezeichnet wurde.

Das ergibt Sinn, denn es hat sich nichts an der Validität eines anderen Auswahlfeld geändert. Lediglich die Selektionskarte für die Förderklasse muss neu gezeichnet werden, da sich seine Selektion angepasst hat. Wird nun aber die *Förderklasse* *Ökolandbau* ausgewählt, so ist zu beobachten, dass das `Card`-Widget zweimal gebaut wurde: Einmal für die Selektionskarte der *Förderklasse*, da sich dessen *ViewModel* änderte; Ein weiteres Mal für die Selektionskarte der *Kategorie*, da die Auswahl *Düngemanagement* nicht länger valide ist und die Karte deshalb mit einem roten Hintergrund eingefärbt werden muss (Abb. 1.2).

Ohne die Änderungen in diesem Schritt zeigt der *Flutter Performance*-Tab, dass sich bei jeder Auswahl einer Option sechs `Card`-Elemente aktualisieren (Abb. 1.3). Das ist der Fall, weil es in Summe sechs Auswahlfelder gibt.

Widget rebuild stats			
			<input checked="" type="checkbox"/> Track widget rebuilds
Widget	Location	Last Frame	Current Screen
<input type="radio"/> Checkbox	selection_card.dart:122	7	42
<input type="radio"/> Text	selection_card.dart:126	7	42
<input checked="" type="radio"/> Card	selection_card.dart:67	2	7

Abbildung 1.2.: Das *Card-Widget* wird zweimal mal neu gezeichnet, Quelle: Eigene Abbildung

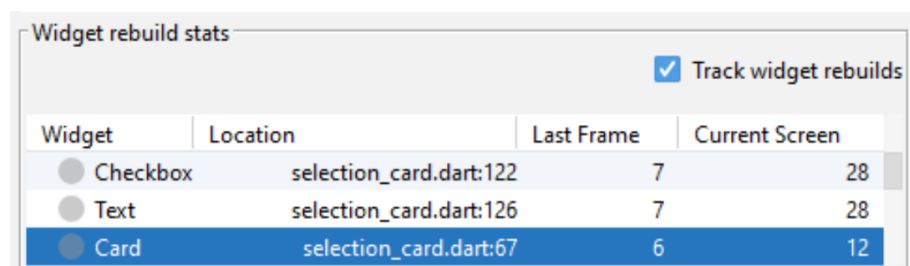
Widget rebuild stats			
			<input checked="" type="checkbox"/> Track widget rebuilds
Widget	Location	Last Frame	Current Screen
<input type="radio"/> Checkbox	selection_card.dart:122	7	28
<input type="radio"/> Text	selection_card.dart:126	7	28
<input checked="" type="radio"/> Card	selection_card.dart:67	6	12

Abbildung 1.3.: Das *Card-Widget* wird siebenmal neu gezeichnet, Quelle: Eigene Abbildung

2. Schritt 5 neu

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

Durch das Lösen dieses Problems ist ein neues Problem entstanden: Alle Selektionskarten müssen bei einer Selektion neu gezeichnet werden. Dieses Verhalten kann auch bei Ausführung der Applikation im Debugmodus in *Android Studio* beobachtet werden. Der *Flutter Performance*-Tab gibt eine Übersicht über die Anzahl der im letzten *Frame* neu gezeichneten *Widgets*. Dieser zeigt, dass sich bei jeder Auswahl einer Option sechs *Card*-Elemente aktualisieren (Abb. 2.1). Das ist der Fall, da es im Formular in Summe sechs Selektionskarten mit einem darin befindlichen *Card*-Widget gibt.



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

Abbildung 2.1.: Das *Card*-Widget wird sechsmal neu gezeichnet, Quelle: Eigene Abbildung

Bei einer geringen Anzahl von Auswahlfeldern sollte das noch keine gravierenden Auswirkungen auf das Laufzeitverhalten der Applikation haben. Doch je zahlreicher die Auswahlfelder werden, desto länger dauert die Aktualisierung der Oberfläche.

Das Problem kann folgendermaßen entschärft werden: Noch bevor das *Widget* `SelectionCard` den `StreamBuilder` in der `build`-Methode zurückgibt, wird der *Stream* `validityChanged` erstellt (Listing 2.2, S. 21, Z. 51-54).

Es handelt sich um eine sogenannte Transformation des *Streams* `priorChoices`, welcher die Momentaufnahme aller ausgewählten Optionen im gesamten Formular übermittelt. Immer dann, wenn der *Stream* `priorChoices` ein neues Ereignis sendet, geschieht für die Abwandlung dieses *Streams* folgendes: Die Methode `map` wandelt jedes Ereignis in ein neues Objekt um (Z. 52). Die aktuelle Momentaufnahme der Auswahloptionen im Formular wird dazu im Parameter `choices` gespeichert. Bei der Umwandlung des Ereignisses werden

die ausgewählten Optionen der aktuellen Selektionskarte über `selectionViewModel.value` abgerufen (Z. 53). Sollte es sich beispielsweise bei der aktuellen Selektionskarte um das Auswahlfeld der *Zieleinheit* handeln, so könnte der ausgewählte Wert *ha* sein.

Mit dem Aufruf `.any((c) => !c.conditionMatches(choices))` wird nun überprüft, ob der ausgewählte Wert – im Fall eines Einfachauswahlfeldes – oder die ausgewählten Werte – bei einem Mehrfachauswahlfeld – mit der neuen Momentaufnahme der Selektionen im Formular kompatibel sind. Für die *Zieleinheit ha* gelten folgenden Bedingungen: Für die *Zielfläche* dürfen die Option *keine Angabe/Vorgabe* und *bitte um Unterstützung* nicht gewählt sein (Listing 2.1, Z. 166-167). Das bedeutet im Umkehrschluss, dass nur die Optionen *AL*, *GL*, *LF*, *DK/SK*, *HFF*, *Landschaftselement/Biotop o.Ä.* oder *Wald/Forst* gewählt sein dürfen.

```
164 static final ha = ZieleinheitChoice("ha", "ha",
165     condition: (choices) =>
166         !choices.contains(ZielflaecheChoice.ka) &&
167         !choices.contains(ZielflaecheChoice.contact));
```

Listing 2.1.: Der Klassenvariable *ha* des Typs *ZielflaecheChoice* wird eine Bedingung hinzugefügt, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-5/conditional_form/lib/choices/choices.dart](#)

Wurde also beispielsweise bei der neuen Selektion in der *Zielfläche* die Option *keine Angabe/Vorgabe* ausgewählt, so würde die Option *ha* invalide werden, da sie nicht mit den *Zieleinheit*-Optionen *keine Angabe/Vorgabe* bzw. *bitte um Unterstützung* kompatibel ist.

Die Methode `map` (Listing 2.2, S. 21, Z. 52) wandelt also das neue Ereignis der Momentaufnahme aller Selektionen im Formular in einen einzigen Wahrheitswert um. Ist der Wahrheitswert `true`, bedeutet dies, dass alle ausgewählten Optionen in der aktuellen Selektionskarte valide sind. Ist er dagegen `false`, so ist wenigstens eine der Auswahloptionen mit den restlichen Selektionen der anderen Auswahlfelder im Formular inkompatibel.

Der resultierende *Stream* wird weiter transformiert: Durch die Funktion `distinct` (Z. 54) werden nur Ereignisse gesendet, sofern sie sich von dem letzten Ereignis unterscheiden. Der *Stream* `validityChanged` sendet also immer genau dann Ereignisse, wenn sich etwas an der Validität der Auswahloptionen der aktuellen Selektionskarte ändert.

Doch dieser *Stream* kann nicht für den `StreamBuilder` benutzt werden. Denn wenn sich die Auswahl in der aktuellen Selektionskarte ändert und die Validität dadurch unverändert bleibt, so erfolgt kein neues Zeichnen der Selektionskarte. Es muss aber eine Aktualisierung stattfinden, damit der neue Wert in der Selektionskarte abgebildet wird. Deshalb ist eine Kombination der *Streams* `validityChanged` und `selectionViewModel` erforderlich. Das `BehaviorSubject` `needsRepaint` soll als diese Kombination fungieren (Z. 56). Es wird mit dem Wert `true` initialisiert. Es ist unerheblich, welcher Wert in dem *Stream* aktuell gespeichert ist. Lediglich dass ein neues Ereignis hinzugefügt wird, um die Aktualisierung der Oberfläche auszulösen, ist wesentlich. Mit der Methode `listen` wird nun sowohl auf

```

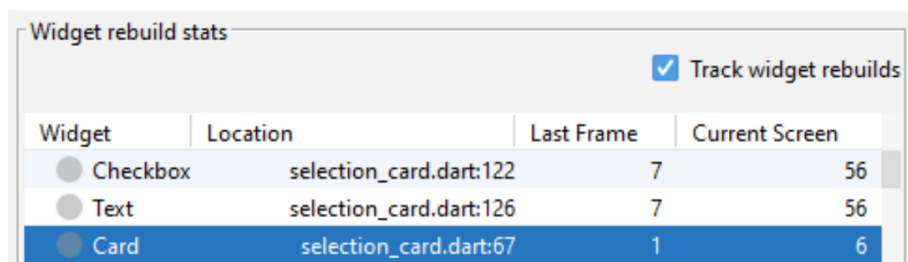
51 final validityChanged = priorChoices
52   .map((choices) =>
53     selectionViewModel.value.any((c) => !c.conditionMatches(choices)))
54   .distinct();
55
56 final needsRepaint = BehaviorSubject.seeded(true);
57 validityChanged.listen((value) => needsRepaint.add(true));
58 selectionViewModel.listen((value)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

```

Listing 2.2.: Der *Stream* `validityChanged` in Schritt 5, Quelle: Eigenes Listing, Datei: [Quellcode/Schritt-5/conditional_form/lib/widgets/selection_card.dart](#)

den *Stream* `validityChanged` (Z. 57) als auch auf `selectionViewModel` (Z. 58) gehorcht. Jedes empfangene Ereignis wird dabei dem `BehaviorSubject` `needsRepaint` hinzugefügt. Dadurch, dass `needsRepaint` für den `StreamBuilder` verwendet wird (Z. 61), zeichnet sich die Selektionskarte immer dann neu, wenn sich die beinhaltenden Auswahloptionen oder aber dessen Validität ändern.

Ein Beispiel: Für die *Zielfläche* ist *AL* und für die *Zieleinheit* ist *ha* ausgewählt. Beide Optionen sind miteinander kompatibel. Nun erfolgt eine weitere Selektion. Für *Zielfläche* wird nun die Option *GL* gewählt. Auch sie ist mit der *Zieleinheit* *ha* kompatibel. Durch die Selektion hat sich der Wert der Selektionskarte der *Zielfläche* geändert, weshalb sie neu gezeichnet werden muss. Alle anderen Auswahlfelder im Formular sind nicht betroffen. Im *Flutter Performance*-Tab ist zu beobachten, dass das `Widget` `Card` nur einmal neu gezeichnet wurde (Abb. 2.2).



Widget	Location	Last Frame	Current Screen
Checkbox	selection_card.dart:122	7	56
Text	selection_card.dart:126	7	56
Card	selection_card.dart:67	1	6

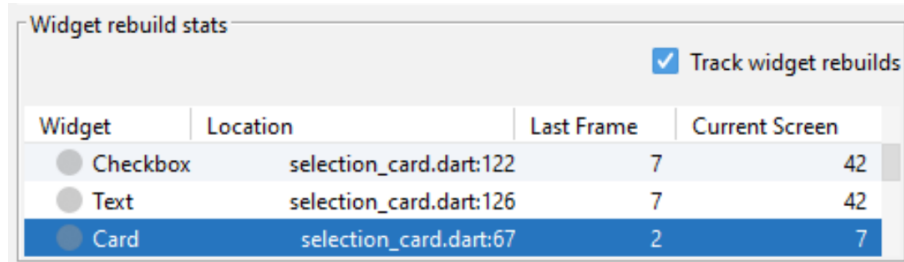
Abbildung 2.2.: Das *Card*-*Widget* wird einmal neu gezeichnet, Quelle: Eigene Abbildung

Durch eine weitere Selektion für *Zielfläche* soll nun provoziert werden, dass die Auswahl der *Zieleinheit* invalide wird. Deshalb wird für die *Zielfläche* nun *keine Angabe/Vorgabe* selektiert. Die *Zieleinheit* *ha* ist damit nicht kompatibel. Deshalb müssen sich nun zwei Auswahlfelder aktualisieren:

2. Schritt 5 neu

- die Selektionskarte *Zielfläche*, weil sich der darin ausgewählte Wert geändert hat und
- die Selektionskarte *Zieleinheit*, da sie zuvor valide war und nun invalide ist.

Der *Flutter Performance*-Tab reflektiert dies, da sich das *Widget Card* nun zweimal neu zeichnet (Abb. 2.3).



The screenshot shows the 'Widget rebuild stats' panel in the Flutter DevTools Performance tab. It includes a table with columns for Widget, Location, Last Frame, and Current Screen. The 'Card' widget is highlighted in blue, indicating it is the current focus. The table shows that the 'Card' widget was rebuilt twice, at frames 2 and 7, while the 'Checkbox' and 'Text' widgets were rebuilt once each at frame 42.

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

Abbildung 2.3.: Das *Card-Widget* wird zweimal neu gezeichnet, Quelle: Eigene Abbildung

Teil III

ANHANG

Eidesstattliche Erklärung

Ich erkläre, dass ich die vorliegende Masterarbeit *Entwicklung einer Formularanwendung mit Kompatibilitätsvalidierung der Einfach- und Mehrfachauswahl-Eingabefelder* selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe und dass ich alle Stellen, die ich wörtlich oder sinngemäß aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe. Die Arbeit hat bisher in gleicher oder ähnlicher Form oder auszugsweise noch keiner Prüfungsbehörde vorgelegen.

Ich versichere, dass die eingereichte schriftliche Fassung der auf dem beigefügten Medium gespeicherten Fassung entspricht.

Wernigerode, den 01.09.2021

Alexander Johr