Zettel Dokumentation

Release 0.9.1

Andres Quast

13.10.2020

Inhalt:

KAPITEL 1

Neue Erfassungsmaske erstellen

Aufsetzend auf einer geeigneten Maske erfolgt die Erstellung einer neuen Maske am einfachsten durch Kopieren und anschließendes Anpassen der benötigten Dateien.

Unter `/app/models/` befinden sich die einzelnen Modelle für die Eingabemasken. Diese erweitern die Klasse `ZettelModel`. Wird eine neue Maske angelegt, so ist dafür ein neues Model entweder als Java-Klasse zu erzeugen, die die Klasse `ZettelModel` erbt, oder einfacher durch Kopieren und Anpassen einer bestehenden Klasse (z.B. `ResearchData.java` zu `ResearchDataKtbl.java`).

1) In der neuen Klasse müssen nun die Variablen, die den Klassennamen enthalten entsprechend angepasst werden
2) Die neue Klasse muss anschließend in der Klasse `ZettelRegister` unter `app/services/` eingetragen werden. Das erfolgt innerhalb des Constructors der Klasse.
3) In `app/services/` muss zusätzlich eine Klasse erstellt werden, die dem neuen Model entspricht (z.B. `ResearchDataKtblZettel.java`).

Innerhalb des nun erstellten Models (`app/models/ResearchDataKtbl.java`) wird zunächst nur festgelegt, wie die Felder angezeigt werden und ob sie z.B. verpflichtend sind. Die Einbeziehung neuer Formularfelder erfolgt erst jetzt.

KAPITEL 2

Neue Formularfelder in Maske integrieren

Dieses Kapitel befasst sich ausschließlich mit der Einrichtung von neuen Formularfeldern im Modul Zettel und der Anbindung von Etikett zur Bereitstellung der benötigten Label. Nicht beschrieben wird, wie die Auswertung der neuen Formularfelder in der Regal-Api erfolgt. Dieses Thema wird in einem separatenKapitel bzw. in der Regal-Api Dokumentation aufgegriffen.

2.1 Bisherige Umsetzung

2.1.1 Liste der im Beispiel anzupassenden Dateien

- `app/views/ResearchDataKtbl`
- `services/ZettelFields`
- `services/KtblDataHelper`
- `conf/info.ktbl.livestock.properties`
- `conf/labels.json`

2.1.2 Umsetzungsweg

Die Einbindung neuer Formularfelder erfolgt zunächst durch Ergänzung des entsprechenden Scala-Views unter `app/views/` (z.B. `researchDataKtbl.scala.html`) um neue Formular-Felder. Beispiel:

```
@accordionPanel(services.ZettelFields.ktblHeaderZF.getLabel(),"ktbl-section"){
        <br />
        @singleSelect(myForm,"livestock",services.ZettelFields.livestockZF.getLabel(),
↪"select-livestock",KtblDataHelper.getLivestockType(),11)
        <br />
}
```

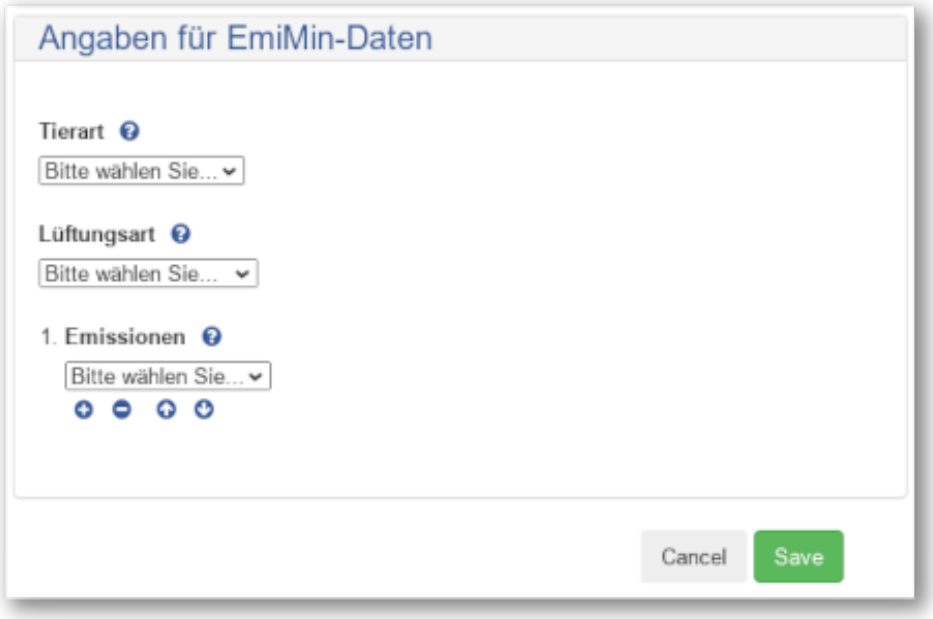

Abb. 1: Beispiel eines Formularbereichs mit zwei singleFields und einem multifield

1. Zunächst wird ein `@accordionPanel` als übergreifende Maske für die neuen Formularfelder vereinbart. Das Akkordion-Panel ermöglicht, die Formularfelder eingeklappt oder ausgeklappt anzuzeigen.
2. Ein `@singleSelect` wird angelegt. Hierbei ist zu beachten, dass die hier definierten Klassen und Methodenaufrufe auch existieren. Dafür ist bisher

a) in der Klasse `services.ZettelFields` eine neue Etikett-Instanz livestockZF anzulegen. *Bisher muss also für jeden in einem Formularfeld benötigten Bezeichner eine (hardcodierte) Etikett-Instanz in der Klasse ``service.ZettelFields`` deklariert werden.*

b) In einer der Helper-Klassen eine neue spezifische Methode z.B. `KtblDataHelper.getLivestockType()` erzeugt werden.

c) Bisher wurden fast immer auch die möglichen Auswahloptionen für `@singleSelect` in der Klasse `services.ZettelFields` als HashMap untergebracht. Dadurch muss die gesamte Zettel-Applikation neu erzeugt werden, wenn sich an den Auswahloptionen etwas ändert.

3. In der Datei `conf.labels.json` muss für den Bezeichner des neuen Feldes ein Json-Etikett angehängt werden, der über die neu angelegte Methode (z.B. `services.ZettelFields.livestockZF.getLabel()` angesprochen wird.

```
{
        "uri": "info:regal/zettel/ktblHeader",
        "comment": "",
        "label": "Angaben für EmiMin-Daten",
        "icon": "",
        "name": "",
        "referenceType": "String",
        "container": "",
        "weight": "1",
        "type": "CONTEXT",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        "multilangLabel": {

        }
},
        {
        "uri": "info:regal/zettel/livestock",
        "comment": "",
        "label": "Tierart",
        "icon": "",
        "name": "Tierart",
        "referenceType": "String",
        "container": "",
        "weight": "1",
        "type": "CONTEXT",
        "multilangLabel": {

        }
}
```

2.2 Vereinfachte Umsetzung mit generischen Klassen und Methoden

Am Beispiel der Anlage des multiselects-Formularfelds soll eine **neue vereinfachte** Möglichkeit vorgestellt werden, um neue Formularfelder anzulegen.

2.2.1 Liste der im Beispiel anzupassenden Dateien

- `app/views/ResearchDataKtbl`
- `conf/info.ktbl.livestock.properties`
- `conf/labels.json`

2.2.2 Umsetzungsweg

Die Einbindung neuer Formularfelder erfolgt zunächst durch Ergänzung des entsprechenden Scala-Views unter `app/views/` (z.B. `researchDataKtbl.scala.html`) um neue Formular-Felder. Beispiel:

```
@accordionPanel(services.ZettelFields.ktblHeaderZF.getLabel(),"ktbl-section"){

        [...]

        @multiselect(myForm, jsonMap, "emissions", services.ZettelFields.
↪getEtikettByName("emissionsZF", "info.ktbl.emission.de.emissions"),"select-emissions
↪",GenericDataHelper.getFieldSelectValues("ktbl.livestock.properties", "info.ktbl.
↪emission"),42)
        <br />
}
```

1. Ein `@multiselect` wird angelegt. Hierbei ist zu beachten, dass die Klassen und Methodenaufrufe existieren. Dafür wird

a) eine neue generische Methode `services.ZettelFields.getEtikettByName("emissionsZF", "info.ktbl.emission.de.emissions")` aufgerufen, die die benötigte Etikett-Instanz „on the fly erzeugt“.

b) die generische Klasse GenericDataHelper mit der generischen Methode . `getFieldSelectValues("Dateiname", "NamensPattern")`[1] aufgerufen.

c) Eine Konfigurations-Datei `conf/Dateiname` angelegt[2]. Diese enthält als Schlüssel Literale, die mit dem NamensPattern beginnen[3].

Das folgende Beispiel zeigt die Konfigurationsdatei für mehrere Formularfelder, inklusive einer ersten Vorbereitung für unterschiedliche Sprachen.

```
# Tierart
info.ktbl.livestock.de.cattle=Rind
info.ktbl.livestock.de.swine=Schwein
info.ktbl.livestock.de.hens=Huhn
info.ktbl.livestock.de.turkey=Pute
info.ktbl.livestock.de.duck=Ente
info.ktbl.livestock.en.cattle=Cattle
info.ktbl.livestock.fr.cattle=Bovin
# Produktionsrichtung
info.ktbl.livestock.cattle.de.diary_farming=Milchviehhaltung
info.ktbl.livestock.cattle.de.calf_fattening=Kälbermast
# Lüftung
info.ktbl.ventilation.de.enforced_ventilation=zwangsgelüftet
info.ktbl.ventilation.de.self_ventilation=freigelüftet
info.ktbl.ventilation.de.combined_ventilation=kombinierte Lüftung
# Emissionen
info.ktbl.emission.de.ammonia=Ammoniak (NH2)
info.ktbl.emission.de.carbondioxide=Kohlendioxid (C02)
info.ktbl.emission.de.diammoniumoxide=Lachgas (N2O)
info.ktbl.emission.de.methane=Methan (CH4O)
info.ktbl.emission.de.smells=Geruch
info.ktbl.emission.de.particle=Partikel (Staub)
info.ktbl.emission.de.biogene_aerosole=Bioaerosole
info.ktbl.emission.de.others=andere
```

Text hier

[1] Das NamensPattern für die Auswahl der Tierart lautet im Beispiel `info.ktbl.livestock`. Die Sprachvariable trennt das NamensPattern von den Auswahloptionen.

[2] Die Datei enthält Schlüssel-Werte-Paare, die durch ein = getrennt werden. Das Verhalten beim Einlesend er Datei orientiert sich an der `Properties`-Klasse aus `java.utils`

[3] Durch den NamensPattern ist es möglich, für ein Formularfeld nur bestimmte Werte aus dieser Datei zu übernehmen. Es kann aber auch für jedes Formularfeld eine eigene Datei angelegt werden.

KAPITEL 3

Bisher verfügbare Formulartypen

Die verschiedenen Formulartypen sind bisher nicht dokumentiert, es wird hier versucht, eine Kurzübersicht zu geben. Alle Formulartypen verweisen jeweils auf Templates im Verzeichnis views. In den Templates gibt es weitere Informationen z.B. zu den Parametertypen, die ich hier zunächst weglasse.

3.1 singleSelect

Template: `views/singleSelect.scala.html`

Parameter aus dem Beispiel:

- `myForm` = Verweis auf die verwendete Model (zumeist `model.ZettelModel`) ?
- `livestock` = Name des Forms ?
- `services.ZettelFields.livestockZF.getLabel()` = Methode der Klasse ZettelFields, die den Label des Feldes livestockZF beim Etikett-Service erfragt und abholt.
- `GenericDataHelper.getFieldSelectValues("ktbl.livestock.properties", "info.ktbl.livestock")` = Statische Methode einer Helper-Klasse, die die zur Auswahl stehenden Werte erfragt und abholt.
- `11` = ist die Position des neuen Formularfeldes im Erfassungsformular. Hiermit ist die Reihenfolge der Formulare steuerbar.

```
@singleSelect(myForm,"livestock",services.ZettelFields.livestockZF.getLabel(),"select-
↪livestock",GenericDataHelper.getFieldSelectValues("ktbl.livestock.properties",
↪"info.ktbl.livestock"),11)
```

`@singleSelect` erzeugt ein Auswahl-Formular mit der Möglichkeit eine Option auszuwählen. Äquivalent zu <select> in html:

```
<select id="cars" name="cars" size="1">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
  </select>
```

`singleSelect` gibt wohl keine Möglichkeit eine Option als Vorausgewählt zu markieren. Ebenso habe ich bisher keine Möglichkeit entdeckt, mehr als eine Option im Auswahlfeld sichtbar zu machen.

3.2 multiselect

Template: `views/multiselect.scala.html`

Parameter aus dem Beispiel:

- `myForm` = Verweis auf das verwendete Model (zumeist model.ZettelModel) ?
- `livestock` = Name des Forms ?
- `services.ZettelFields.livestockZF.getLabel()` = Methode der Klasse ZettelFields, die den Label des Feldes livestockZF beim Etikett-Service erfragt und abholt.
- `GenericDataHelper.getFieldSelectValues("ktbl.livestock.properties", "info.ktbl.livestock")` = Statische Methode einer Helper-Klasse, die die zur Auswahl stehenden Werte erfragt und abholt.
- `11` = ist die Position des neuen Formularfeldes im Erfassungsformular. Hiermit ist die Reihenfolge der Formulare steuerbar.

```
@multiselect(myForm, myMap, "livestock",services.ZettelFields.livestockZF.getLabel(),
↪"select-livestock",GenericDataHelper.getFieldSelectValues("ktbl.livestock.properties
↪", "info.ktbl.livestock"),11)
```

`@multiselect` erzeugt zunächst ein Auswahl-Formular mit der Möglichkeit eine Option auszuwählen. Äquivalent zu <select> in html.

```
<select id="cars" name="cars" size="1">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
  </select>
```

Zusätzlich erzeugt `multiselect` aber auch noch Buttons mit denen die Nutzenden bei Bedarf weitere dieser Felder im Formular erzeugen können oder löschen können. Damit ist eine Mehrfachauswahl zu einem Feld möglich