Lesson 11: FileBrowser, UUID, fromString

Created by Jacques Marais, last modified on Sep 13, 2018

> *As a Farmer I want to upload my Government Assistance Recipient Certificate.*

- Lesson Outcomes
- New & Modified App Files
- UUID and Blob Data Types
- Further Model Additions
- Populating UUID Attributes Using a Text Field
- Populating Blob Data Types Using the File Upload Widget
- Using the File Browser
- Lesson Source Code

Lesson Outcomes

By the end of this lesson you should:

- Be familiar with uuid and blob data types
- Be able to use the `Uuid:fromString` function to convert a **`string`** type to a **`uuid`** type
- Be able to use the file upload and file browser widgets

New & Modified App Files

`./model/roles/Farmer.mez`

`./web-app/lang/en.lang`

`./web-app/presenters/farmer_profile/FarmerProfile.mez`

`./web-app/presenters/farmer_profile/FarmerProfileMenu.mez`

`./web-app/views/farmer_profile/FarmerProfileDocumentation.vxml`

UUID and Blob Data Types

For this lesson we need to add two attributes to the `Farmer` object. These represent the certificate id and the actual certificate data. The code snippet below demonstrates this:

```
uuid governmentAssistanceCertificateId;
blob governmentAssistanceCertificate;
```

Further Model Additions

In addition to the attributes above we also add another **`datetime`** attribute, namely, `documentationProfileUpdatedOn` to the `Farmer` object. This attribute is used to keep track of when last the `governmentAssistanceCertificateId` and `governmentAssistanceCertificate` attributes were updated.

```
datetime documentationProfileUpdatedOn;
```

See the lesson source code for further details on how it's used.

Populating UUID Attributes Using a Text Field

To populate the certificate id attribute we will use a text field, an intermediate unit variable and the Helium built-in function, `Uuid:fromString`, to convert the captured text to a **`uuid`**. The code snippets below demonstrates this:

```
 <textfield label="textfield.government_assistance_reci
    <binding variable="certificateId"/>
</textfield>
.
.
.

<submit label="submit.save" action="saveGovernmentAssis
```

```
unit FarmerProfile;

string certificateId;

string saveGovernmentAssistanceCertificate() {

    if(certificateId == null) {
        Mez:alertError("alert.null_government_assistar
        return null;
    }

    .
    .
    .

    uuid parsedId = Uuid:fromString(certificateId);

    if(parsedId == null) {
        Mez:alertError("alert.invalid_government_assis
        return null;
    }

    farmer.governmentAssistanceCertificateId = parsedI
    certificateId = null;
    return null;
}
```

In the `saveGovernmentAssistanceCertificate` function above we first do a manual validation to check that the user has populated the value from the frontend before saving. We then convert the value to a **`uuid`** using the `Uuid:fromString` function. If the provided value is not a valid uuid, the function will return **`null`**. For this case we also add a manual validation. The final step is to assign the the converted value to the attribute on the `Farmer` object.

All view and unit code added as part of this lesson is in the `FarmerProfileDocumentation` view and the `FarmerProile` unit.

Populating Blob Data Types Using the File Upload Widget

We have already demonstrated in Lesson 9 how the file upload widget can be used to upload CSV files that can then be parsed by Helium. For this use case we simply want to upload the **`blob`** data, store it using our data model and then provide a mechanism to download the data from the frontend. The code snippet below demonstrates the uploading and storing of the blob data representing a government assistance certificate:

```
<fileupload label="fileupload.government_assistance_rec
    <binding variable="farmer">
        <attribute name="governmentAssistanceCertificat
    </binding>
</fileupload>

<submit label="submit.save" action="saveGovernmentAssis
```

Once again we add a manual validation in the to the `saveGovernmentAssistanceCertificate` function to check that the file has been uploaded before saving the result:

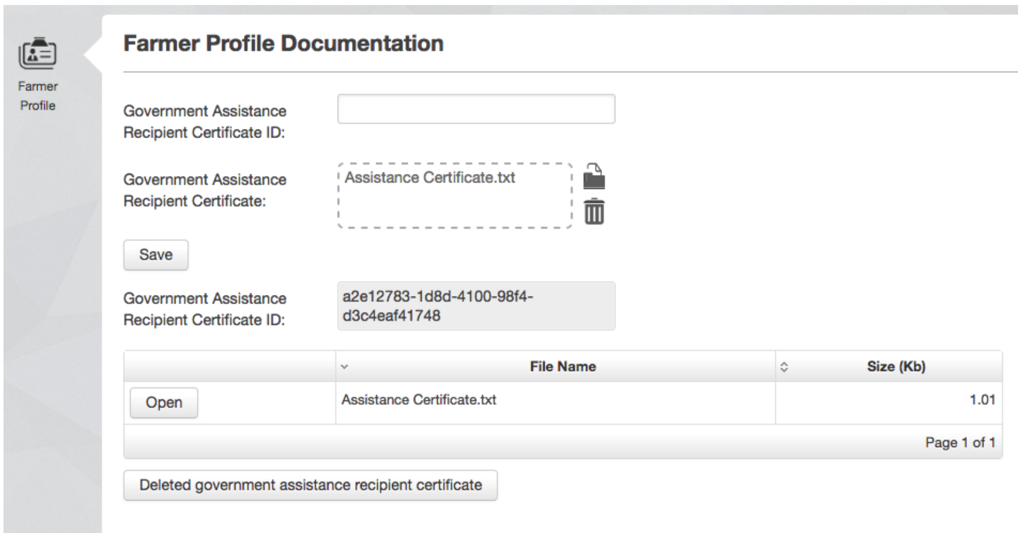
```
if(farmer.governmentAssistanceCertificate == null) {
    Mez:alertError("alert.null_government_assistance_ce
    return null;
}
```

Using the File Browser

Helium provides a file browser widget that is presented as a data table with two columns representing the file name that was uploaded and the file size and a row action labelled "Open" that can be used to download the file. These cannot be altered. Similarly to the data table widget a collection source needs to be provided where the object in the collection contains a blob attribute. In our case this collection will only contain the current farmer user. In addition the blob attribute name needs to be specified:

```
<filebrowser dataAttribute="governmentAssistanceCertifi
    <visible function="showFileBrowser"/>
    <collectionSource function="getCurrentFarmerAsColle
</filebrowser>
```

The screenshot below demonstrates the completed view with certificate that has been uploaded:



Farmer Profile Documentation

Government Assistance Recipient Certificate ID:	
Government Assistance Recipient Certificate:	Assistance Certificate.txt

Government Assistance Recipient Certificate ID: a2e12783-1d8d-4100-98f4-d3c4eaf41748

	File Name	Size (Kb)
Open	Assistance Certificate.txt	1.01

Page 1 of 1

Note that the info widget, file browser and button at the bottom of the view is hidden until a file has been uploaded with a valid id.

Lesson Source Code

Lesson 11.zip

No labels