Lesson 26: Displaying Images

Created by Adriaan Klue, last modified by Jacques Marais on Sep 28, 2018

> *As a farmer I would like to upload pictures, with dates and descriptions, to track my crops' quality by their appearance over time.*

Lesson Outcomes

By the end of this lesson you should:

- Know how to add a gallery widget to your view XML
- Know the data model requirements for providing the gallery with a collection source
- Know how to filter and sort your gallery images
- Be familiar with the gallery widget's behaviour in terms of user experience

The second and third topics in this list contain nothing new to learn. We simply cover an example of how you would go about providing, sorting and filtering images using methods covered in previous lessons so that, after this lesson, you would have seen that the gallery widget requires nothing special or unique to the widget to accomplish this.

Scenario

We want to provide our farmer users with a feature to track the growth and quality of their crops over time. For this we'll add a new category to the Farmer Profile page, linking to a new page with, amongst other things, a gallery widget to display the farmer's photographs as thumbnails, filtered on the type of crop and sorted according to their dates. To upload the pictures, we'll provide the `<fileupload>` widget as described in lessons 9 and 11.

New & Modified App Files

`./model/objects/CropQualityPicture.mez`

`./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/FarmerProfileMenu.vxml`

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

Widget Overview

The gallery widget will display an image as a resized thumbnail with a caption underneath. It will arrange multiple thumbnails in a grid of up to nine images per page. If there are more than nine images for the gallery to display, paging links will appear at the bottom of the widget to navigate between the pages. Thumbnail size varies according to the number of thumbnails being displayed, and will fill the available widget space. In other words, a single image in the collection source will fill the widget area, while five or more images will display at the smallest thumbnail size.

A gallery is given a collection source in the view XML using the familiar `<collectionSource>` child element. This collection must be of a custom object that has a **`blob`** attribute for the image, specified by `imageAttribute="..."` in the parent XML. You can also specify a **`string`** attribute to be used as the image caption by providing `descriptionAttribute="..."`.

Clicking on a thumbnail will select or deselect an image (a selected thumbnail container will have a dark gray background), which, for the purposes of DSL logic, shows which image is bound to the specified binding variable (see code snippet below).

You can double-click on a thumbnail to view the image full-size (or your screen size limit), and you can hover over a long caption to display the full caption in a tooltip pop-up.

Adding a Gallery Widget

We'll add the gallery in a new view as follows:

FarmerProfileCropQualityPictures.mez

```
<gallery title="gallery_title.farmer_crop_quality_pictu
  <collectionSource function="getFarmerCropQualityPictu
  <binding variable="selectedCropQualityPicture"/>
</gallery>
```

The function `getFarmerCropQualityPictures` returns a collection of a new object called `CropQualityPicture`.

FarmerProfile.mez

```
CropQualityPicture[] getFarmerCropQualityPictures() {
  return CropQualityPicture:relationshipIn(farmer, farm
}
```

CropQualityPicture.mez

```
persistent object CropQualityPicture {
  date pictureDate;
  string description;
  blob image;

  @ManyToOne
  Farmer farmer via cropQualityPictures;
}
```

Add the functionality for uploading gallery images on the same view as follows:

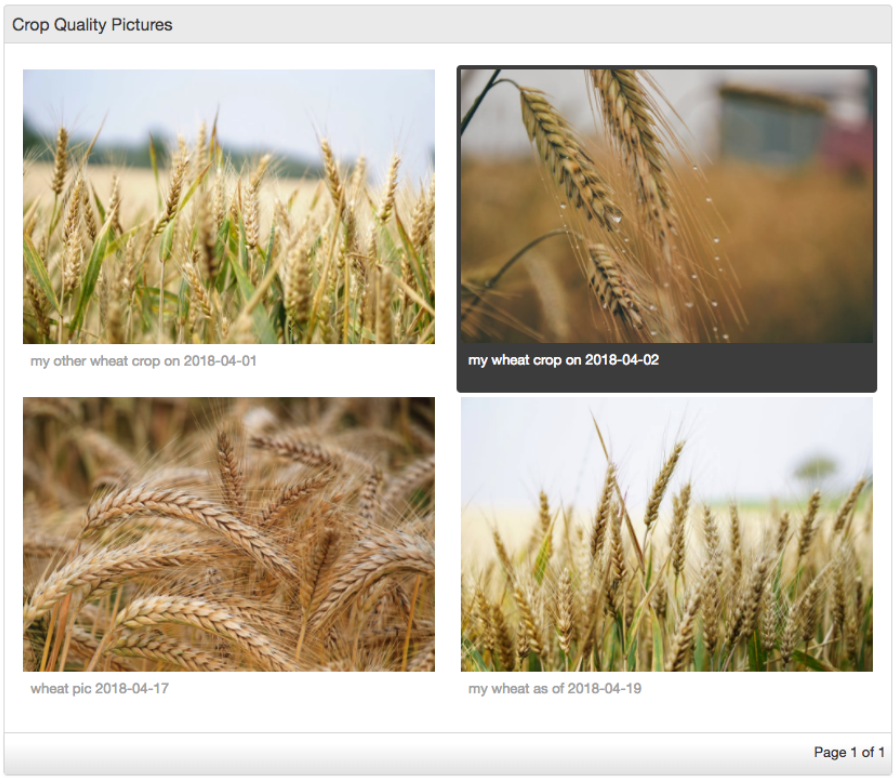
FarmerProfileCropQualityPictures.mez

```
1  <fileupload label="fileupload.upload_new_crop_picture">
2    <binding variable="cropQualityPicture">
3      <attribute name="image"/>
4    </binding>
5  </fileupload>
```

Also add a `<datefield>` for `cropQualityPicture.pictureDate` and a `<textfield>` for `cropQualityPicture.description`.

We'll also add basic edit and delete buttons (and their required DSL logic) for the newly created objects. See the attached source code if you're uncertain about this step.

Upload a few pictures. Afterwards your gallery widget should look something like this:



Filtering and Sorting Gallery Images

In order to filter on the crop type, we add a relationship to the `CropQualityPicture` object:

CropQualityPicture.mez

```
10  @ManyToOne
11  Stock cropType via CropQualityPictures;
```

So that we can change the collection getter function to:

FarmerProfile.mez

```
1
    CropQualityPicture[] getFarmerCropQualityPictures() {
```

```
  if (cropPictureFilter == null) {
    return CropQualityPicture:relationshipIn(farmer,
  } else {
    return CropQualityPicture:and(relationshipIn(farm
        relationshipIn(cropType, cropPictureFilter));
  }
}
```

Also add a `<select>` widget for selecting the `cropPictureFilter`. For this to work, the farmer's crops must be added to his/her profile (feature already available via Farmer Profile since Lesson 8).

Next we'll sort according to the date on the object:

FarmerProfile.mez

```
CropQualityPicture[] getFarmerCropQualityPictures() {
  CropQualityPicture[] result;
  if (cropPictureFilter == null) {
    result = CropQualityPicture:relationshipIn(farmer,
  } else {
    result = CropQualityPicture:and(relationshipIn(far
        relationshipIn(cropType, cropPictureFilter));
  }
  result.sortAsc("pictureDate");
  return result;
}
```

These should be familiar patterns by now. The important thing to see is that you order and filter for the gallery widget using DSL logic before sending the collection to the widget. There is currently no filtering or sorting features built into the gallery widget itself.

Lesson Source Code

Lesson 26.zip

Other Resources

Attached here are some free stock photos, including the examples used in this lesson.

No labels

4 Comments

Siphenkosi Filtane

Can we validate if the uploaded file via fileupload widget is actually an image not other file types?

Jacques Marais

@ Siphenkosi Filtane , for blob attributes, Helium automatically adds additional meta data fields to the object containing the attribute. These meta data fields represent the mime type, file name and file size and the name of these fields are prepended with the blob attribute name. For example, if your blob attribute is "data", Helium adds the fields data_fname__, data_mtype__ and data_size__. These fields can be referenced like any other object attribute from the DSL. You can use data_mtype__ to check the what the type of file being uploaded is. A png file for example will have a value of "image/png".

Lennie de Villiers

@ Siphenkosi Filtane

Chat to me. I have done this already in our app on IoT.

Siphenkosi Filtane

Thanks @ Lennie de Villiers , @ Jacques Marais has shown me how to do the validation and it works fine.

Thank you again @ Jacques Marais