Concept - Image Editor

This is a concept for story: https://magnolia.atlassian.net/browse/BL-97 (As a user, I can crop, rotate and scale images stored in the DAM)

Purpose

Enable users to edit images in the way specified on this UX proposal: Editing assets using the Assets app

Proposal

Design

There can be an editor for each media type. The editors will be similar:

- Open in a special media editor dialog.
- Use the actionbar to initiate edit actions.
- Have a footer that by default has "save" and "cancel buttons",
  but that can be replaced/overlayed with a new footer with: action specific information and controls once an action edit mode is initiated.
- Display a visual representation of the media in a media panel.

Known media types include:

- Image
- Video
- Audio
- Document
- Flash

Definitions:

MediaPanel : The area of the control where one sees a representation of the media - such as the image.

EditMode: A state of the MediaEditor. For some edit actions such a crop and stretch, an EditMode is entered where the user can interactively manipulate the image. They can either cancel or apply the changes which exits the EditMode.

Implementation

MediaEditors should be in new module: magnolia-module-dam-media-editors. (Contained within magnolia-module-dam-parent.)

MediaEditor & DialogWithActionbar

Abstract class: DialogWithActionbar
Abstract class: MediaEditor extends DialogWithActionbar
Implementor: ImageEditor

The DialogWithActionbar extends Vaadin CustomComponent class. It will compose a Dialog and an Actionbar. It will need to position the Dialog in a new way (not centered.)

The MediaEditor will require a Presenter, View and ViewImpl.

MediaEditor acts on a media binary, and returns a media binary. It does not know anything about Assets.

- This means that it could act on external resources, or media otherwise not stored in an Asset. Its more generally useful.

The MediaEditor will be opened in a Vaadin Overlay, directly on top of the opening element. The MediaEditor is modal to a sub app. The DialogViewport will not be used.

MediaEditor Actionbar

- Each MediaEditor implementor creates an Actionbar definition with configuration by code. Actionbar presenter used to run the bar.
- (Actionbars are currently configured in the workbench for a subapp - which is not really appropriate.)

(Note, the Message View in the messages part of the pulse shell app also has content and an actionbar. But it is not really a dialog with an actionbar - so it probably would not subclass DialogWithActionbar. However it still makes sense to separate the functionality so that it is available to others needing a dialog with an action bar.)

LowerPriority:The editModes of a MediaEditor need to be configurable for each call. For example - on one call a user can choose the aspect ratio of a crop, but in another call - could be configured to have a fixed aspect ratio.

- MediaEditor should expose a way to configure the editModes. Just as an example, interface will probably not look like this: theMediaEditor.getEditMode("crop").setAspectRatio(4:3);

Changes to Dialog

The dialog footer must be able to change its contents to support the controls that an editing mode can add. For example when a crop is in progress the footer actions change, and a dynamic label appears.

Change Dialog into something capable of holding multiple components so that we can use Vaadin client/server capabilities to add and remove additional components to the footer (or header).

- Change BaseDialog from AbstractSingleComponentContainer to AbstractCompontent and implements HasComponents, so that we can add components to the footer and header.
- Footer and Header remain as client-side structures, they dont change to Vaadin components.
- Updating actions stays the same.
- Biggest change is new technique to add components to footer.

Technique on how components get added to proper location in footer - although it is not a component itself.

- As the Dialog will accept additional Components to be rendered within a footer and/or header - we will attach controls directly to the Dialogs (though through proper methods e.g. setFooterToolbar(Component c)).
- In order to know where exactly to put those components on the client side - we will persist the references to them in the shared state (as those are passed over the wire as string ids - no overhead will occur).
- When the HierarchyChangeEvent occurs we get rid of the redundant widgets from the view, and update it with the ones from the state.
- To sum up: all we have to do is to slightly change the interface of BaseDialog and its Connector and add the slots for new controls to the footer and/or header.

Discussion: Please add what methods will be added - how will BaseDialog or other Dialog classes change.

Editing Modes

Edit Modes are entered for several of the actions like crop, stretch (& advanced rotate), which causes:

- Change in how the mouse or touch interacts with the Media Panel.
- Change in footer.

When an EditMode is activated:

- A new widget is loaded into the MediaPanel. There is a widget per EditMode.
- New label, controls and actions are loaded into the Footer.

Realtime interaction with image

- The crop tool updates the crop area in realtime in response to mouse/touch movement, the Stretch editmode stretches the image in realtime.
- These interactions happen on the client via the widget.
    - If the interaction is implemented with an existing javascript library, then a Vaadin Javascript Extension is created.
    - If the interaction is built from scratch, then a GWT extension is created.
    - Only on clicking "Apply" button are instructions sent to server to perform the image manipulation on the server.
    - Client image is then refreshed with manipulated image from the server.

Realtime update of text in footer in response to mouse/touch interaction in MediaPanel

- DynamicLabel: Extend the client side widget of the Label component such that it can update its value.
- When starting an EditMode we create both the editing widget for the MediaPanel, and the new controls for the footer - including this label.
    - The editing widget gets a reference to the DynamicLabel.
- The editing widget can update the DynamicLabel without going back to the server.

Changes to the media

- While an EditMode is active, only a client side representation of the media will be manipulated.

- When changes from an EditMode are applied - (Or an instantaneous action is performed) they are applied to a temorary file stored on disk. (We can not store temporary state in ram because the binaries could be very large - and this could lead to excessive resource usage on server - if for example many editors were concurrently working on large binaries.)
    - A temporary file does not need to be created as soon as the editor is opened - a user may cancel the operation before making any changes.
- A final altered binary is returned when the user clicks "Save Changes" and exits the MediaEditor.
    - The temporary file is deleted.

All server side image manipulation will be implemented with the Imaging module. Methods may need to be added to Imaging module.

Question:

- When an EditMode is started, how are the dialog parts (footer/header/content) is "configured"?
    - Is there a map of the label, the controls, the actions for each type of EditMode?
    - Is there a MediaPanelToolbar interface, which each EditMode implements?

Future

There could be a configuration on the magnolia-module-dam-media-editors module for each MediaEditor type - and for each EditMode of that MediaEditor. ie:

- mediaEditors
    - imageEditor
        - editModes
            - crop
                - widget: cropWidget
                - footer:
                    - dynamicLabel
                    - constrainAspectRatioControl
                    - actions
                        - Cancel
                        - ApplyCrop
            - rotate
            - flipHorizontal
            - flipVertical