**SourceManager**

Class that is used to load items and save them.

**Item** {

Image image = FolderImage;

String? Name

Bool isImage = false;

}

Used to display items in selection widget. Default value is for folder type of item.

If folder, then set name to folder name.

If image, set image value and isImage to True

**Future<List<Item>?> getItemsFor(ItemSource source) async**

Func to retrieve items for passed source.

If source requires auth and user is not auth, it will return empty array.

**DrawingImage** {

Image image;

String Name = “”

ImageSource source;

}

Used to work with image, so no setup for folders.

Every field is required for other classes to work properly. If name is already occupied, it may replace already existing image, so be cautious.

**enum ImageSource** {

IconData icon

String Name

}

Used to differentiate sources.

Currently have device(“Device”) and dropbox(“Dropbox”).

Should have name and icon for proper showcase by SelectorView.

For saving and loading, it could be used without any fields.

**Future<String> saveImage(Uint8List image, DrawingImage drawingImage) async**

Saves image binary to its original source.

We pass DrawingImage struct to know source and name. TODO: make version, where we pass only needed info.

Returns string as result. Returns error value as result. If no error than result is “”.

**Filters**

This file contains 5 filters for image filter. Each filter is an matrix with values, used to init ImageFilter value.

Standart, Sepium, Sepia, Greyscale, Vintage.

First one contains default image values.

Each filter should be a matrix with the size of 5x4 with the values between -2.0 and 2.0. If it does not, artifacts may occur.

For matrix data see <https://kazzkiq.github.io/svg-color-filter/>

**AuthManager**

Currently not implemented due to not having a reason. Currently none of image sources does not require auth. In future may be implemented.

**SelectorView**

Widget for selecting images to draw on. May be used as directory searcher for images.

**ImageSource currentImageSource = .device**

Shows current image source. Starts with device value

**List<Item>? itemList = []**

Holds current item list to display. It is loaded when widget changes source or first loaded

**refreshItemList()**

loads item list for current source, without changing it. Should be called when refreshing item list, for example after saved edited image.

**loadItemsForSource() async**

loads items list for current source. Calls SourceManager func and passes currentImageSource

**selectSource(ImageSource source) async**

call to select source. Changes widgets values and starts to load items, checks auth.

**Widget building**

Widget contains a grid of items. If no items is available, then shows Text(“No image to show”)

When clicked on a valid image, should push a “/canvas” to Navigator with a DrawingImage and reference to refresh func. DrawingImage should be created from Widget values and Item that was clicked.

**DrawingView**

Widget between drawing canvas and selection widget.

On init should pass DrawingImage and refresh func

**saveImage() async**

func that exports image from canvas, by getting \_imageKey.currentState.exportImage()

checks for validity and saving response from SourceManager saveImage() func.

If image valid and saving returned empty value(that means success), shows popup with success message. If return value is not empty or image is not valid, shows popup with error message.

Before popup, pops to previous widget in Navigator.

**Widget building**

Returns canvas inside of WillPopScope to disable swipe to back gesture.

We need to pass \_imageKey to be able to export image later, image and saveImage func

\_paint\_over\_image

Canvas widget for app. Requires key, image and saveImage func to be passed, other values are optional.

On init sets up text controllers for color text fields and drawing controller. Also takes passed image and converts to binary array by calling \_resolveAndConvertImage(), so underlaying canvas it can work with it.

This widget consists of several distinct parts.

Tools

Tools are shown in the middle bottom when selected index = 0.

Currently there are 3 main tools, pen, lines and eraser. There can be more in the future.

Each tool is built using -buildToolButton where we pass image, isSelected and func onPressed

When we select tool, it calls func in \_controller.setMode(\*mode\*) and elevates a little bit, because it is selected now.

Color picker

Widget for selecting color to draw with. It is displayed as small circle with the tint of selected color. When tapped, shows PopupMenu with a grid of colors to select from and HEX text field and 3 field for Red, Green, Blue. When selecting color, should call \_updateColorInputs(Color color) func. It updates other text fields values, so everything is in sync with each other.

Filters

Filters widget can be triggered by selected filters in selection button below the tools. IF index = 0, then we show tools and other instruments, otherwise with show filters widget.

Filters widget is horizontal scrollview with our image and applied filters from Filters file. When tapped, sets selectedFilter, which is over canvas, so image displays filtered.

NavBar

Have 2 buttons, undo and clear. When canvas is edited or either of these buttons are pressed, we check the paint history in \_controller. If empty, clear and undo is not active. Clear removes all the painting available and undo removes last action.

Other

To the bottom left there is X button. Does not save image and just pops Navigator

To the bottom right there is save image button. Calls saveImage func passed by DrawingView. Also there is shapes button. After clicking shows shapes popupMenu. Each shape is its own tool in \_controller class.

To the left there is slider, which sets the stroke width value in \_controller.

All these widgets hide when we use filters and shows when we use instruments.

**\_controller**

Canvas controller. Used to hold values for Canvas. If we want to change mid run some values, we change them there. Used in DrawImage() class in \_image\_painter file

**\_image\_painter**

Widget for actual drawing, because in it we can draw lines and curves. Requires \_controller to be passed when init called.

**@override paint()**

Overrides paint func to display drawn lines. Firstly displays image, then saves it as base layer, so that eraser can work.

Goes through the \_controller history and draws each of items on canvas.

For displaying current action, we check if \_controller.isbusy. Then we take start, end, mode values from controller, the paint it

