# Storyboard Feature for Krita

Google Summer of Code Project

## Synopsis

A storyboard is a graphic organizer that consists of illustrations and comments displayed in sequence for the purpose of pre-visualizing and planning of motion picture and animation. The storyboard docker would offer a GUI for creation and organization of storyboards. It would manage multiple images and comments in a user-friendly manner and would save users from managing multiple files. It would have multiple display modes to organize the storyboard in a way suitable for the user. It will make Krita more of a stand-alone program for animation.

## Benefits to the community

Animators would not have to use external software for storyboarding. Team collaborations among the Krita animator community would be easier due to better discussions and planning. It would also enable amateur animators to prepare a draft idea easily. This will attract new animators to Krita and help grow the animator community of Krita.

## Deliverables- Project goals

The storyboard docker should be able to perform the following:

1. A UI to show thumbnails for every non-empty frame of the image along with related metadata for each frame. The frames can be reordered by drag and drop operation.
2. Frames would be added to the storyboard docker by default but can be removed from the storyboard docker or the timeline docker.
3. Change the mode of display (Row mode, Column mode, Grid mode).
4. Edit images and metadata for a storyboard element from the storyboard docker.
5. Export the storyboard in the desired format (PDF and SVG).
6. User documentation will be written for the docker, how it interacts with other dockers and the workflow while using it.

## Implementation Details

### UX Design

The docker is going to contain a list of images and associated comments. We will refer to an image and associated comments as a storyboard **item**. Each Storyboard item consists of the following:

**Image/Thumbnail** – A scaled version of the instance of image. Users would be able to draw on it. It would update the canvas image.
**Frame Number –** The frame number corresponding to the frame in the timeline docker.
**Storyboard element name** – A name given to the storyboard element which can be edited by the user.
**Text fields along with their names**– Text field describing different attributes related to that scene such as Dialogue, Action, comment etc.
**Duration:** A spin box to specify the time duration for that storyboard.
This would add a number of hold frames after that frame depending on the fps value.

Storyboard items could be moved around by drag and drop.
Users can choose how to arrange those elements. 3 different modes will be offered.

- **Grid Mode** – Storyboard items will be ordered in a grid or table .
- **Row Mode** – Storyboard items will be ordered in rows, images and comments next to each other in one row and different storyboard items in different rows.
- **Column Mode** –Storyboard items will be ordered in columns, images and comments below each other in one column and different storyboard items in different columns.

There will also be a Show option to decide the visibility of different items. This option will have the following choices:

- Thumbnail only
- Comment only
- Both

These two options will together result in 9 possible ways of displaying the storyboard which gives artists ample choices.

There would be a comments drop-down list consisting of different metadata fields for every storyboard item. It can be used to toggle visibility of different metadata fields. It can be used to add or delete metadata fields on a per-document basis.

There will be scrollbars(horizontal and vertical for different modes) to navigate the storyboard. Storyboard items can be rearranged by drag and drop operations.

For exporting the storyboard in different formats there would be a drop-down button that would list file formats available. PDF, SVG and PNG formats would be offered. Other formats would be added on feedback. Clicking on any formats would open a layout specification widget that would let users choose rows per page, columns per page and page size. Option to specify layout using SVG file would also be offered.

Other changes to UI will be made based on feedback from animators and developers.

# Plugin

The storyboard docker will be part of Krita's plugin system. A class called StoryBoardDockerPlugin along with StoryboardDockerDockFactory will be created to tell the

system how to create the plugin. StoryboardDockerDock is another class that will be created to define the GUI of the docker . It will be derived from either KoCanvasObserverBase or KisMainWindowObserver so as to access the canvas and timeline docker. StoryboardView would use the canvas and create a KisScratchPad like object that could be used to display and edit the canvas.

1. **StoryboardDockerDock** – It sets up UI, functions, signals, actions. Consists of child widgets that manage signals. Passes arguments to child widgets created. It would consist of all UI widgets such as buttons, comboBox, layouts, labels and StoryboardView to manage Storyboard elements. It will also handle creation and updation of new StoryboardItem.
2. **StoryboardDockerDockFactory** – It creates a StoryboardDockerDock object and sets its default area, position and id.
3. **StoryboardDockerPlugin** – Template of the docker plugin. It adds the StoryboardDockerDockFactory object to an instance of KoDockRegistry.

# MVC Design

The docker would need multiple views for underlying data. To implement this efficiently we will use MVC Design. Qt offers classes to implement this architecture efficiently and it has been used in other plugins before.
Classes to be created:

1. **StoryboardItem** – A single storyboard item consisting of members corresponding to the parts described in UI. This class is simply a representation of the storyboard item's data in raw form. It has no information about orientation or modes.
2. **StoryboardView** – This is the view class. It will be derived from QTableView. It will keep and update an array of StoryboardItem objects. It will accept user inputs and emit signals to StoryboardDelegate to handle them. It will implement the UI at the centre of the storyboard docker. It will arrange StoryboardItems based on orientation, mode, rows, columns. It will also handle mouse events for StoryboardItems. It will allow the user to drag and drop StoryboardItems.
3. **StoryboardDelegate** – This is the delegate class. It will be derived from QAbstractItemDelegate. It will implement the UI for a single storyboard item. It will also handle any editing and keep the Model and View in sync. It will take signals from Storyboard View and modify information in StoryboardModel.
4. **StoryboardModel** – This is the model class. This class will store StoryboardItems. It will hold them in the form of a list and this data will be used to construct the view. It will also be changed by StoryboardDelegate when data is updated. It will load the data at startup and update data at the time of saving.

For exporting the storyboard, layout would be designed based on user input. Users can design layout using custom options such as rows per page, columns per page and page size or using an SVG file.

### PDF exporting

For exporting to pdf we can use QPrinter which allows "printing" to PDF files. To do so we can set up a QPrinter instance like this

```
QPrinter printer(QPrinter::HighResolution);
printer.setOutputFormat(QPrinter::PdfFormat);
printer.setOutputFileName("path/to/file.pdf");
```

Since QPrinter inherits QPaintDevice, anything that supports outputting graphical content to QpaintDevice can thus be used for generating PDFs. We can get a QPaintDevice from KisPaintDevice since KisPaintDevice is derived from QpaintDevice.
We can also use Qt's Graphics View framework along with QPrinter as it is more suitable for arbitrarily placed and transformed 2D graphical items.
To handle different numbers of rows and columns we can change the separation between rows, columns and size of the elements. By adjusting these parameters we can easily fit any number of storyboard elements on a page.

### SVG Exporting

For exporting in SVG format we can use the QSvgGenerator class along with QPainter. It would be similar to export to pdf. Once we have the paint device we can use it to create an SVG file using QSvgGenerator class. If this approach does not work then I'll create SVG files manually.

## Integration

### Saving and Loading Storyboard

The data in the Storyboard docker would consist of images and comments. The image would be loaded using the frame number and accessing that frame from the KeyframeChannel of all KisNodes. The comments and other metadata can be saved by adding a list of storyboard structs to KisDocument. The kra_saver and kra_loader files would be changed to manage proper loading and saving of data.

### Interaction between timeline docker and storyboard docker

Any non-empty frame should be added to the storyboard docker. When a frame is selected in one of the dockers it should be selected in the other docker as well. Any changes in the frame should mirror in the storyboard docker and vice-versa. Deletion of frame from storyboard docker should not mirror in timeline docker. Duration field in storyboard docker should add hold frame after a frame. Swapping storyboard items should swap the frames in the timeline docker. Clicking on a frame in either docker should highlight the same in the other docker if it exists.

## Testing

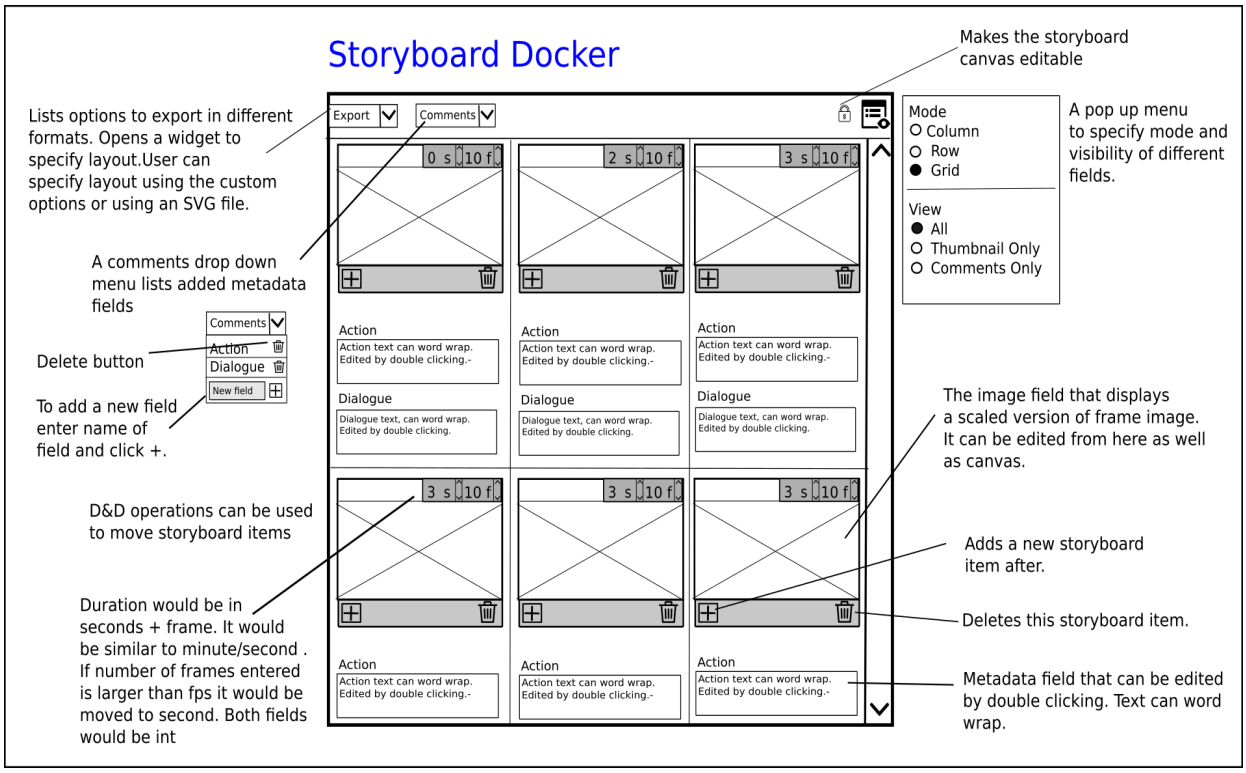
Tests will be written in the Qt Test Framework for the following :

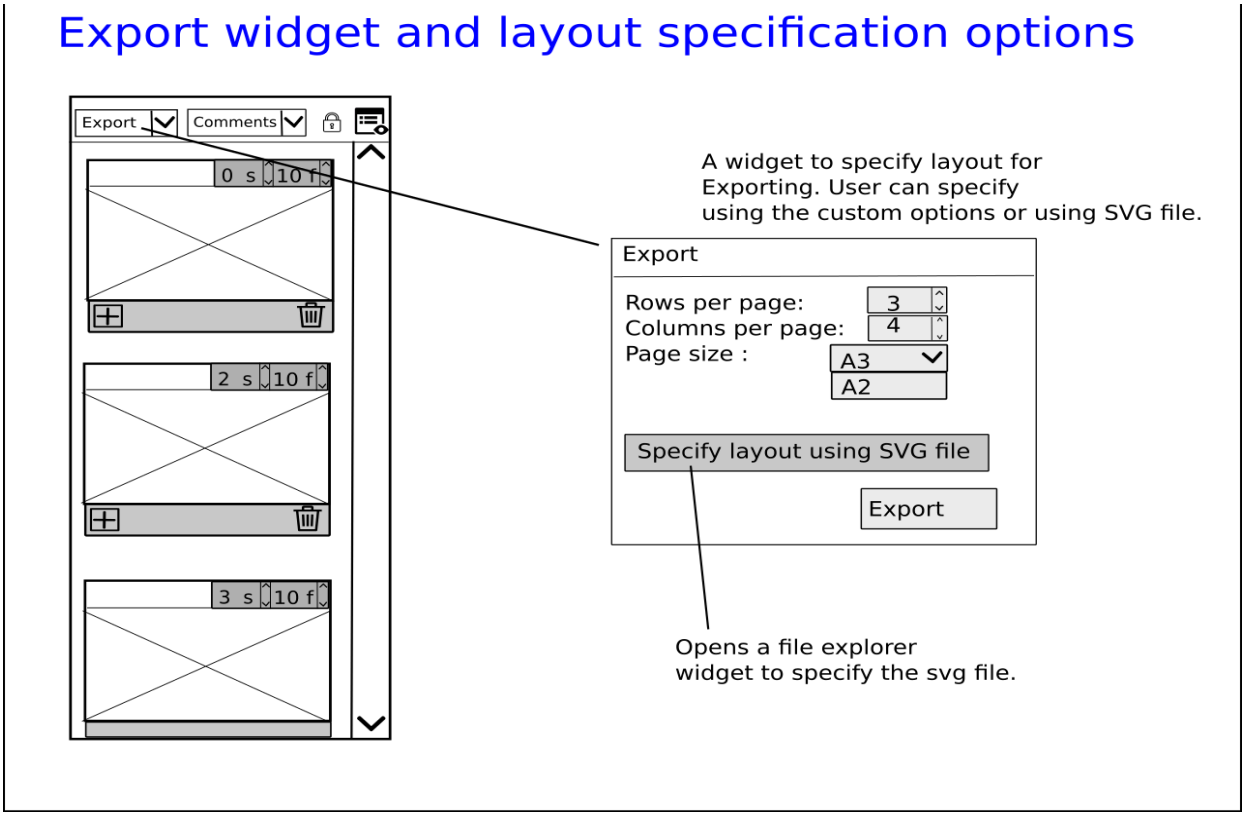
1. Interaction between delegate and model class.
2. Interaction between timeline docker and storyboard docker.
3. Saving and loading storyboard metadata when opening or closing Krita.
4. Export as a pdf functionality.
5. Export as svg functionality.

# Documentation

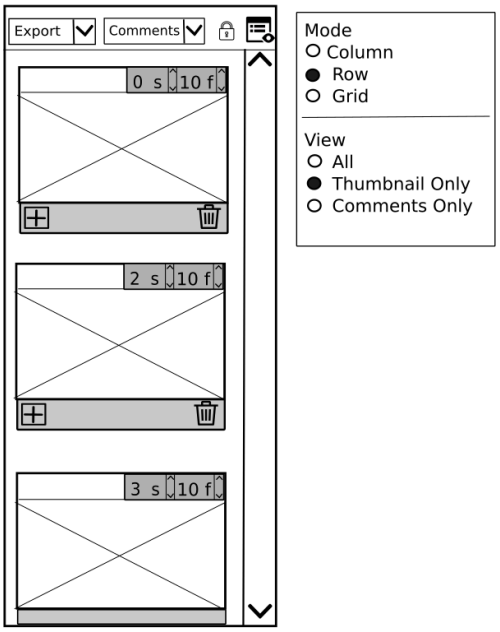
Documentation for the new Storyboard Docker and related workflow will be written with feedback from the animator community.

# Mockups

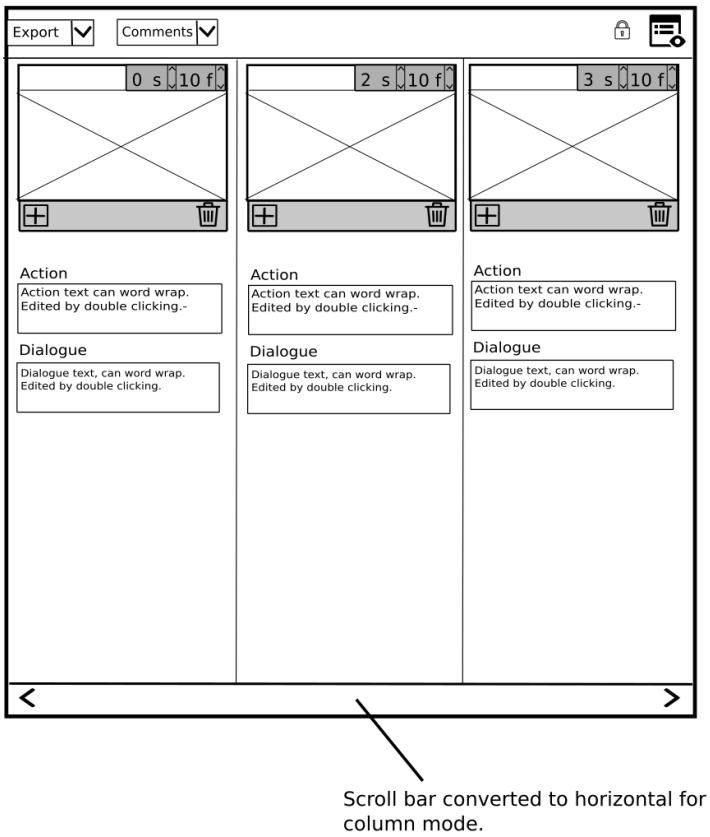


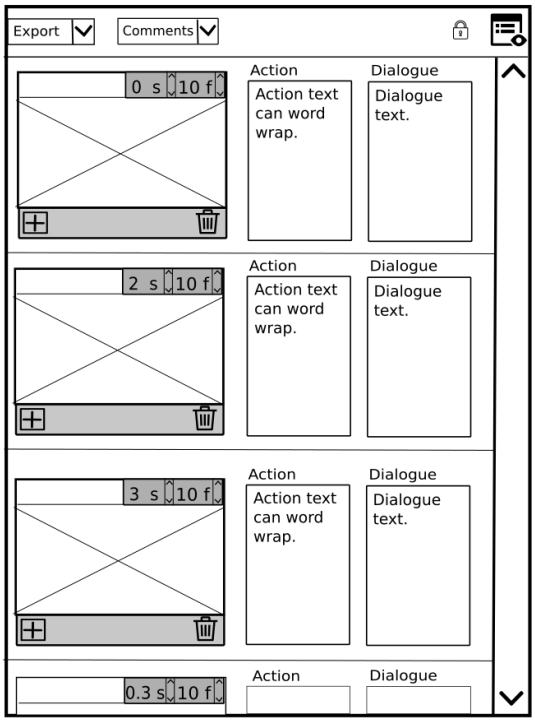
## Row Mode + Thumbnail view



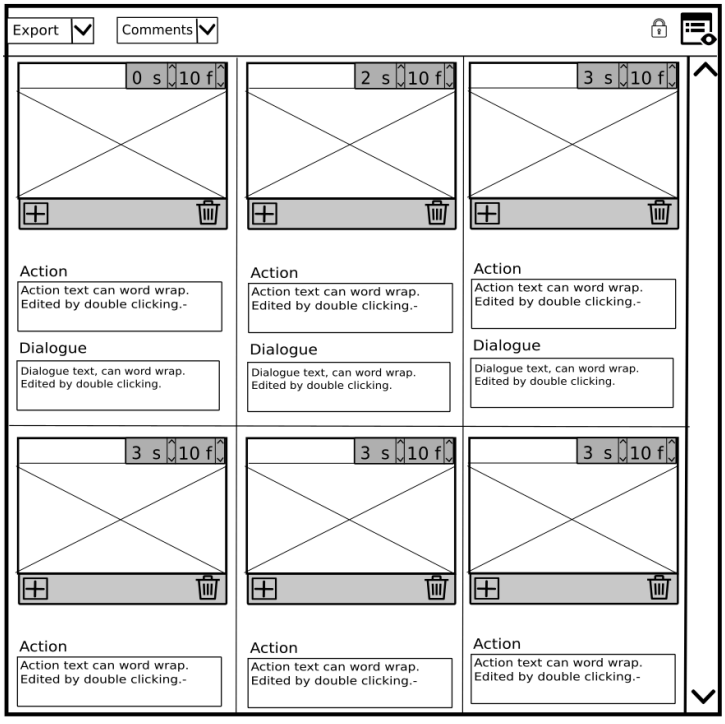

## Column mode + Both view



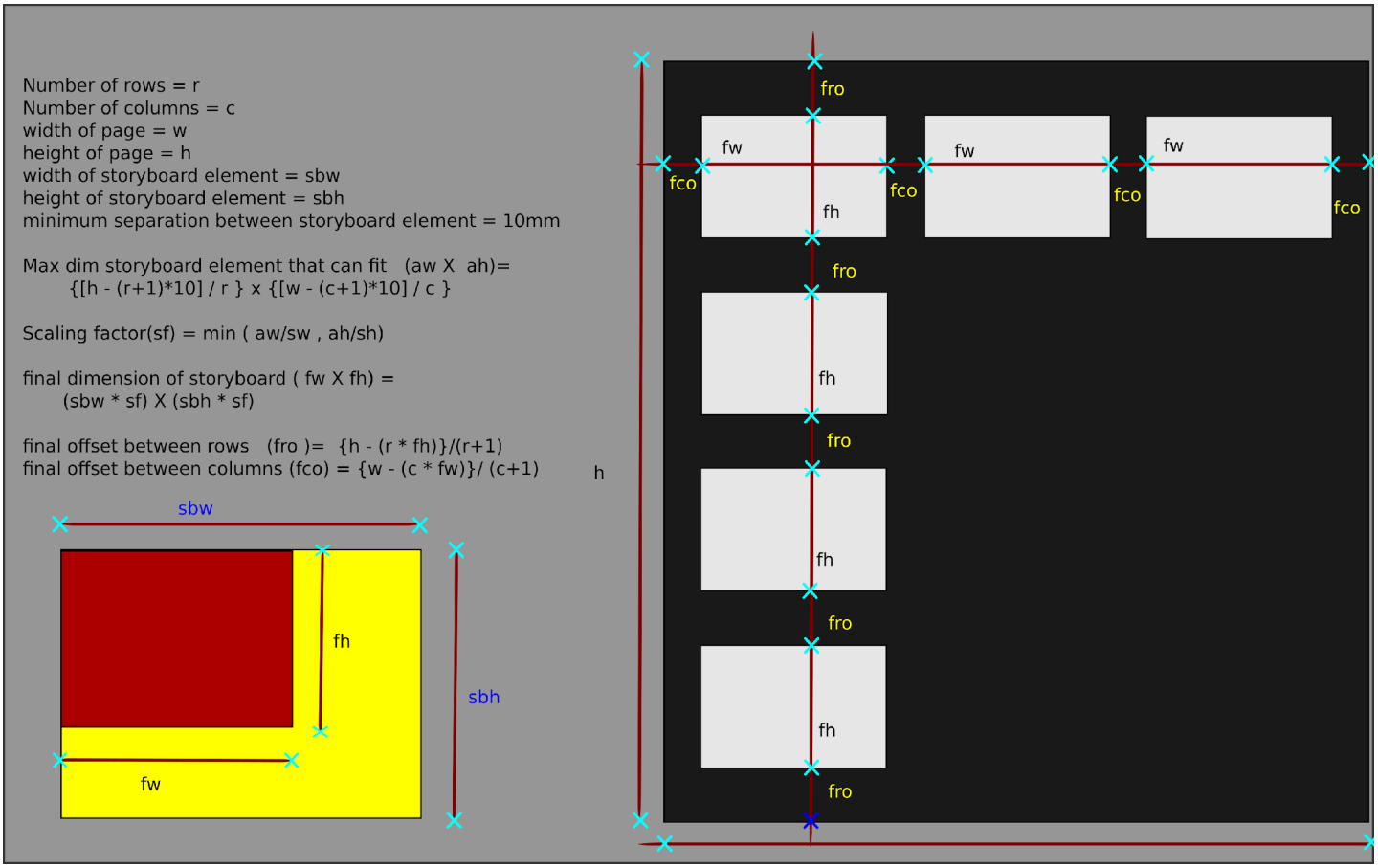

## Row mode + Both view




## Grid view + Both view



## Custom layout design for Export function




# Timeline

### Community Bonding Period ( May 4 - May31)

- ~~I am already familiar with most of the developer community of Krita. I'll try to connect more with developers and artists.~~
- ~~Read the codebase related to animation API and figure out how the timeline docker and storyboard docker would communicate.~~
- ~~Get familiar with the Qt test framework and how to use unit tests.~~
- ~~Get familiar with the documentation system of krita and add some documentation for earlier commits.~~
- Write a draft user documentation detailing workflow and interaction with other parts of Krita.
- ~~Finalize the design of GUI of the docker with feedback from the developers and artists.~~
- ~~Add a blog describing your plans and progress to KDE Planet.~~

### Week 1 ( June 1 - June 7)

- ~~Implement StoryboardDockerDock, Storyboard DockerDockFactory and StoryBoardDockerPlugin classes.~~
- ~~Add the docker to the Krita's plugin system.~~
- ~~Implement GUI for outer docker in StoryboardDockerDock.~~

- ~~Blog progress and plan.~~

**Week 2 ( June 8 - 14)**

- ~~Write unit tests for delegate class and model class. Only the functionalities that are to be implemented in the next 2 weeks would be tested.~~
- Add details in the docs for introduction to storyboard docker, different modes and their uses.
- ~~Implement the StoryboardElement class.~~
- ~~Start implementing rudimentary MVC classes. The aim here is to build a basic MVC architecture. No interaction with external data will be implemented now.~~
- ~~Blog progress and plan.~~

**Week 3 ( June 15 - 21)**

- ~~FInish implementing rudimentary MVC classes.~~
- ~~Integrate the rudimentary MVC classes in the StoryboardDockerDock class.~~
- ~~Get a basic docker ready with editable metadata field and thumbnail field. The data entered needn't be saved.~~
- ~~Blog progress and plan.~~

**Week 4 ( June 22 - 28)**

- ~~Implement various modes in view class ( Grid, Column, Row).~~
- ~~Run unit tests and debug any unwanted behaviour.~~
- Get the basic GUI and modes tested by users on KA and reddit. Ask for feedback on docs.
- ~~Blog progress and plan.~~

**Week 5 ( June 29 - July 5)**

- ~~First evaluation period( June 29 - July 3)~~
- Write unit tests for interaction between delegate class and model class.
- Write unit tests for interaction between timeline docker and storyboard docker.
- Write documentation for how the storyboard interacts with the timeline docker.
- ~~Start implementing UI for individual storyboard elements in delegate class~~.
- ~~Blog progress and plan.~~

**Week 6 (July 6 - 12)**

- ~~Add the functionality to customize metadata fields from outer docker to every delegate class. We should be able to add and remove metadata fields for every storyboard element from one place on a per-document basis.~~
- ~~Implement various views in view class ( Thumbnail only, Comment only, Both). These options should affect all storyboard elements.~~
- ~~Implement custom UI for editing metadata fields and duration if needed.~~
- ~~Start integrating the timeline docker with the storyboard docker. Whenever any non empty keyframe is added get the frame number for that frame. Add a scaled version of the canvas to the image field such that changes to either the storyboard canvas or main canvas mirror each other.~~
- ~~Blog progress and plan.~~

**Week 7 (July 13 - 19)**

- ~~Fully integrate timeline docker with storyboard docker such that any change to frames( delete, move or other operations) in timeline docker gets updated in the storyboard docker as well.~~
- ~~Implement functions to handle lock button (to toggle drawing on storyboard docker).~~
- ~~Implement various functions in delegate class to modify and add metadata such as duration and text fields in the model.~~

- ~~Blog progress and plan.~~

**Week 8 ( July 20 - 26)**

- ~~Run unit tests, debug any unwanted behaviour.~~
- Get the views, modes and interaction with the timeline docker tested by users on krita-artists or reddit. Make changes based on the feedback.
- ~~Blog progress and plan.~~
- ~~Review the timeline.~~

**Week 9 ( July 27 - August 2)**

- Second evaluation period(July 27 - 31)
- Write unit tests for saving, loading and export functionality.
- Decide how to save and load data when starting and closing krita. Implement the changes in model class, KisDocument class, kra_saver and kra_loader.
- Blog progress and plan.

**Week 10 ( August 3 - 9)**

- Design and implement GUI for export dialog.
- Implement layout specification using SVG file. Implement the custom layout specification.
- Arrange multiple storyboard items according to the layout specified by the user as one paint device.
- Blog progress and plan.

**Week 11 ( August 10 - 16)**

- Implement the export as PDF/PNG functionality.
- Implement the export as SVG functionality.
- Write documentation for export function and options.
- Get the export functionality tested by users, especially layout design. Make changes based on feedback.
- Blog progress and plan.

**Week 12 ( August 17 - 23)**

- Compile all the documentation parts together and finalize it.
- Buffer week for any unforeseen events.
- Blog progress and plan.

**Week 13 ( August 24 - 31)**

- Final evaluation and wrap up
- Blog progress and plan.

Note:

- Semester exams might get postponed into summers (May-June) due to COVID-19 but that would not pose any problem for completion of the project.

# Biography

Saurabh Kumar
Email: saurabhk660@gmail.com
IRC nick: confifu
Phone: +91 7355995290

Prior contributions:

https://invent.kde.org/kde/krita/-/merge_requests/202
https://invent.kde.org/kde/krita/-/merge_requests/207
https://invent.kde.org/kde/krita/-/merge_requests/209
https://invent.kde.org/kde/krita/-/merge_requests/222
https://invent.kde.org/kde/krita/-/merge_requests/233
https://invent.kde.org/kde/krita/-/merge_requests/247

I am currently a 2nd Year undergraduate Computer Science student in Delhi Technological University, New Delhi, India. I have been using Krita for a year now. I like the simplicity and minimalism of digital art. I started using Krita while looking for open source digital art softwares and have never stopped since.

I started contributing to Krita in the winter vacations(Dec-Jan) this year. I have since fixed 4 bugs and implemented two features. The first feature adds functionality to split a layer into local selection masks and the second allows users to add an image as file-layer from the command-line . I am very enthusiastic about Krita, both as a user and as a developer since it is my first interaction with the open source community and digital art.

The storyboard feature is a large project compared to my other contributions and I am still familiarizing myself with Krita. But I am confident that I can complete the promised part. Therefore I hope to utilize the opportunity of GSoC to get help from mentors so that I can have a better understanding of the codebase and accomplish this task.

After GSoC I'll continue to contribute to Krita. I like the community of artists and developers around Krita. I will be in contact with mentors through the IRC channel, Gitlab and Phabricator etc. I will also submit detailed weekly reviews through a weekly blog.

I am not submitting proposals to other organizations. I don't plan on working on anything this summer other than GSoC.

## Alternate Timeline for the last 4 weeks

**Week 9 ( July 27 - August 2)**

- Second evaluation period(July 27 - 31)
- Write unit tests for saving, loading and export functionality.
- Write documentation about Interactions with the docker(general introduction). Also write about modes and views and the other buttons in the docker and in the storyboard item.
- Blog progress and plan.

**Week 10 ( August 3 - 9)**

- Decide how to save and load data when starting and closing krita. Implement the changes in model class, KisDocument class, kra_saver and kra_loader.
- Design and implement GUI for export dialog.
- Write documentation about interaction between timeline docker and storyboard docker.
- Blog progress and plan.

**Week 11 ( August 10 - 16)**

- Implement layout specification using SVG file. Implement the custom layout specification.
- Arrange multiple storyboard items according to the layout specified by the user as one paint device.
- Implement the export as SVG functionality.
- Write documentation for export function and options.
- Get the export functionality tested by users, especially layout design. Make changes based on feedback.
- Blog progress and plan.

**Week 12 ( August 17 - 23)**

- Implement the export as PDF/PNG functionality.
- Write documentation for code.
- Compile all the documentation parts together and finalize it.
- Blog progress and plan.

**Week 13 ( August 24 - 31)**

- Final evaluation and wrap up
- Blog progress and plan.