Software Requirements Specification Document

Simple IMage Processor *for* Lazy Editors (aka SIMPLE)

1.0.0
10/29/2024

The Lead Eaters
Addison Casner, Derek Jennings, Quinn Pulley, Will Verplaetse

Submitted in partial fulfillment of the requirements of
IT 326 – Principles of Software Engineering

Table of Contents

1. Introduction

1.1 Purpose

This document will provide an overview of SIMPLE, including the definition and specifications of this application. This document is intended for use by the development team and stakeholders to aid in understanding the functional and technical requirements of this application.

1.2 Scope

The software product to be produced is SIMPLE. This product will feature a user-friendly interface for making basic changes to images. These basic edits will include things like applying one or more filters to an image, cropping the image, and flipping the image vertically and/or horizontally. This product will provide simple tools for the user to apply their changes, as well as a preview window for the user to see their changes in real-time. This product will not feature detailed editing capabilities like some of those offered by Photoshop, such as advanced selections, styling, or blending.

1.3 Definitions, Acronyms, and Abbreviations

SIMPLE – Simple IMage Processor *for* Lazy Editors

1.4 Overview

The following sections will give a general description and the specific requirements of SIMPLE. This will include an extensive overview of the application's functional and non-functional requirements, user characteristics, software and hardware environment, and assumptions and dependencies.

2. General Description

2.1 Product Perspective

This product is independent and self-contained. SIMPLE is similar to other image editing applications, such as Photoshop, with the main difference being that this application places priority on ease of use and simplicity. Photoshop's user interface is complex and takes time to learn. This application will feature easy-to-understand functions that clearly correspond to the related operation.

2.2 User Characteristics

The main characteristic of the intended users is inexperience with image editing technologies. A secondary characteristic is reluctance to spend money and time to learn a more complex image-editing application.

2.3 System Environment

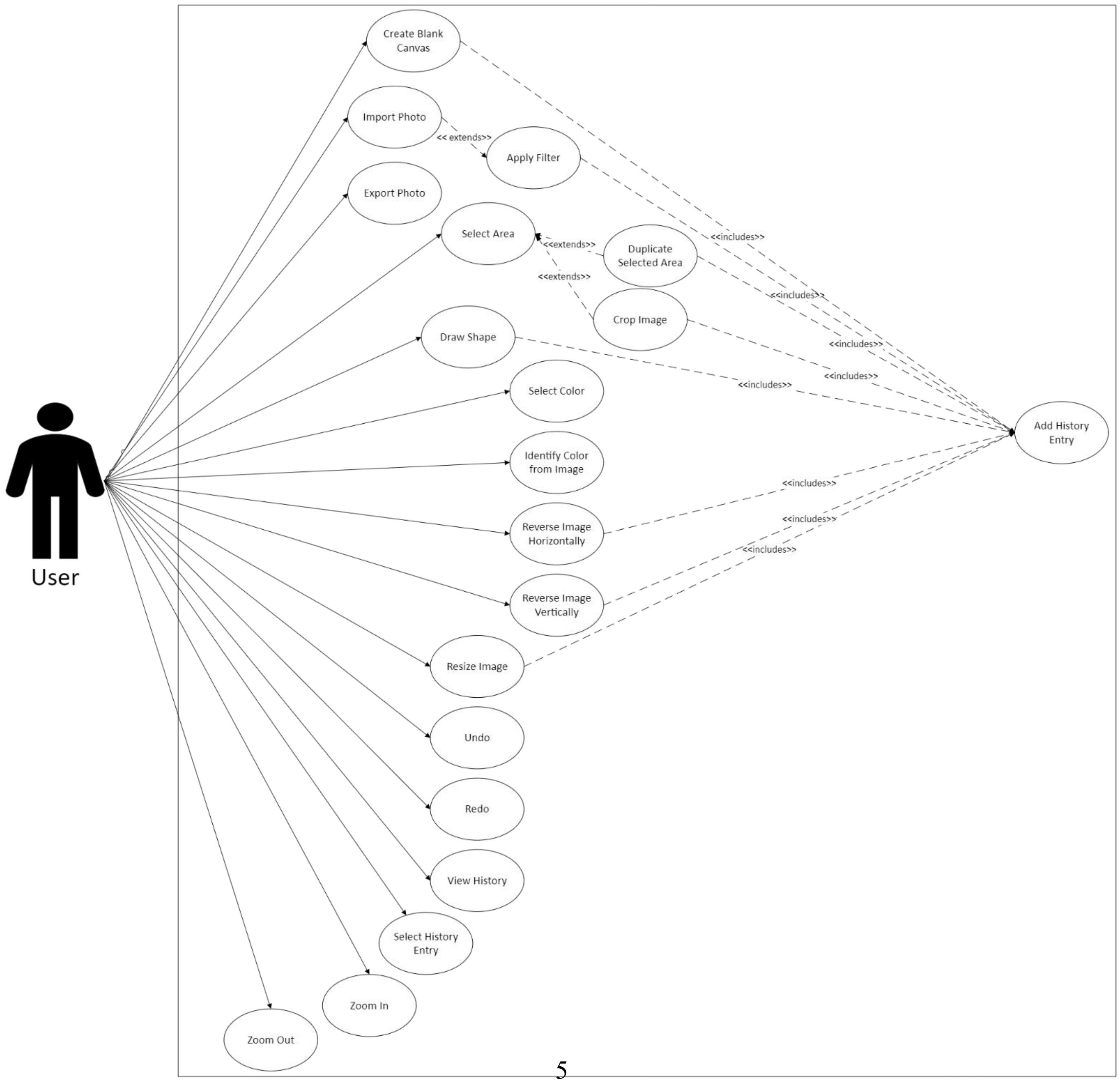
The final product will be a standalone executable file for use on Windows systems. The executable contains the files and python environment necessary to run the file on any compatible system without the need to install python.

2.4 Assumptions and Dependencies

We assume the Windows system can run the packaged software executable, which will have been generated from the original code.

3. Specific Requirements

3.1 Functional Requirements



3.1.1 Create Blank Canvas

a. Description – The system will initialize a blank image of the current color.
b. Actor(s) – User
c. Trigger – The user wants to create a blank canvas.
d. Conditions
    i. Pre – SIMPLE must be open and running.
    ii. Post – A uniform image of the current color will be created with specified dimensions and loaded for use.

3.1.2 Import Image

a. Description – The system will load the image chosen by the user and display it. Only .png and .jpg image types will be supported.
b. Actor(s) – User
c. Trigger – The user wants to view and/or edit their image.
d. Conditions
    i. Pre – SIMPLE must be open and running.
    ii. Post – The system will display the image chosen by the user.

3.1.3 Export Image

a. Description – The system will save the currently loaded image, with any changes, to the user's local files. The file export type will be the same as the type of the imported image. For an image created from a blank canvas, the file export type will be .png.
b. Actor(s) – User
c. Trigger – The user wants to save the image, with any changes, to their local files.
d. Conditions
    i. Pre – An image is currently loaded in the system.
    ii. Post – The currently loaded image, with any changes, will be saved to the user's files.

3.1.4 Select Area

a. Description – The system will select an area of the image, with coordinates specified by the user, in which subsequent edits will take place.
b. Actor(s) – User
c. Trigger – The user wants to select a portion of the image.
d. Conditions

i. Pre – An image is currently loaded in the system.
ii. Post – The selected portion of the image will signify that it is currently selected and future operations will apply only to this area.

3.1.5 Duplicate Selected Area

a. Description – The system will copy the area selected by the user to overwrite another area of the image.
b. Actor(s) – User
c. Trigger – The user wants to copy a section of the image to another spot.
d. Conditions
   i. Pre – An image is currently loaded in the system, and an area of the image is selected.
   ii. Post – The originally-selected area will be copied to where the user specified. A new history entry will be created with the updated image.

3.1.6 Crop Image

a. Description – The system will crop the image to the section specified by the user.
b. Actor(s) – User
c. Trigger – The user wants to crop the image.
d. Conditions
   i. Pre – An image is currently loaded in the system, and an area of the image is selected.
   ii. Post – The image will be cropped to the section specified by the user. A new history entry will be created with the updated image.

3.1.7 Apply Filter

a. Description – The system will apply the filter chosen by the user to the entire image. Filter options include blur, invert, adjust brightness, and grayscale.
b. Actor(s) – User
c. Trigger – The user wants to apply a filter to the image.
d. Conditions
   i. Pre – An image is currently loaded in the system.
   ii. Post – The filter chosen by the user will be applied to the entire image. A new history entry will be created with the updated image.

3.1.8 Draw Shape

a. Description – The system will replace all pixels in the area, depending on the shape chosen by the user, with the color chosen by the user.
b. Actor(s) – User
c. Trigger – The user wants to draw a shape.
d. Conditions
   i. Pre – An image is currently loaded in the system.
   ii. Post – All pixels in the area will be replaced by the chosen color. A new history entry will be created with the updated image.

3.1.9 Select Color for Draw Shape

a. Description – The user will select a color for use by draw shape.
b. Actor(s) – User
c. Trigger – The user wants to select a color.
d. Conditions
   i. Pre – SIMPLE must be open and running.
   ii. Post – A color will be selected for draw shape.

3.1.10 Identify Color from Image

a. Description – The user will select a pixel from the image and the system will display the color information about that pixel.
b. Actor(s) – User
c. Trigger – The user wants to know the color information about a pixel in the image.
d. Conditions
   i. Pre – An image is currently loaded in the system.
   ii. Post – The system will display the color information about the pixel chosen by the user.

3.1.11 Reverse Image Horizontally

a. Description – The system will reverse the entire image horizontally.
b. Actor(s) – User
c. Trigger – The user wants to reverse the image horizontally.
d. Conditions
   i. Pre – An image is currently loaded in the system.

   ii. Post – The image will be reversed horizontally. A new history entry will be created with the updated image.

3.1.12 Reverse Image Vertically

a. Description – The system will reverse the entire image vertically.
b. Actor(s) – User
c. Trigger – The user wants to reverse the image vertically.
d. Conditions
   i. Pre – An image is currently loaded in the system.
   ii. Post – The image will be reversed vertically. A new history entry will be created with the updated image.

3.1.13 Resize Image

a. Description – The system will resize the entire image to the dimensions specified by the user.
b. Actor(s) – User
c. Trigger – The user wants to resize the image.
d. Conditions
   i. Pre – An image is currently loaded in the system.
   ii. Post – The image will be resized to the dimensions specified by the user. A new history entry will be created with the updated image.

3.1.14 Undo

a. Description – The system will revert the image to the state prior to the last change made by the user.
b. Actor(s) – User
c. Trigger – The user wants to undo a change they made to the image.
d. Conditions
   i. Pre – An image is currently loaded in the system, and at least one change has been made.
   ii. Post – The image will no longer reflect the last change made by the user. The version that was undone will remain available for the redo operation until it is replaced by a new change.

3.1.15 Redo

a. Description – The system will restore the image to the state including the last change that was undone by the user.

b. Actor(s) – User
c. Trigger – The user wants to redo a change that they have previously undone.
d. Conditions
    i. Pre – An image is currently loaded in the system, and at least one change has been undone.
    ii. Post – The last change that was undone will be reapplied to the image. The history entry for this edit will be restored.

3.1.16 View History

a. Description – The system will display a list of all changes the user has made to the image. If no changes have been made, the list will have a single entry for the unaltered image.
b. Actor(s) – User
c. Trigger – The user wants to view all the changes they have made to the image.
d. Conditions
    i. Pre – An image is currently loaded in the system.
    ii. Post – A list of all changes the user has made to the image will be displayed. If no changes have been made, the list will have a single entry for the unaltered image.

3.1.17 Select History Entry

a. Description – The system will revert the image to its state at the history entry selected by the user from the list of history entries. The undone history entries will remain unless another change is made to the image.
b. Actor(s) – User
c. Trigger – The user wants to revert the state of the image after a particular change.
d. Conditions
    i. Pre – An image is currently loaded in the system.
    ii. Post – The preview window will display the state of the image at the history entry selected by the user from the list of history entries.

3.1.18 Add History Entry

a. Description – The system will add a history entry recording the current state of the image.
b. Actor(s) – User, and System
c. Trigger – The user applies an edit to the image.
d. Conditions
    i. Pre – An image is currently loaded in the system.

ii. Post – A new history entry will be created with the updated image.

3.1.19 Zoom In

a. Description – The system will zoom in on the area selected by the user.
b. Actor(s) – User
c. Trigger – The user wants to zoom in on a portion of the image.
d. Conditions
    i. Pre – An image is currently loaded in the system.
    ii. Post – The image will be zoomed in on the area selected by the user.

3.1.20 Zoom Out

a. Description – The system will zoom out so that more of the image is visible.
b. Actor(s) – User
c. Trigger – The user wants to zoom out to view more of the image.
d. Conditions
    i. Pre – An image is currently loaded in the system, and has been zoomed in.
    ii. Post –The image will be zoomed out so that more of the image is visible.

3.2 Non-Functional Requirements

3.2.1 Performance

Operations performed in the app, including loading, saving, and editing, must be performed in a reasonable amount of time (according to the size of the image). Max image size and number of recorded history entries will vary based on memory available.

3.2.2 Reliability

Operations must be consistent in their results (with the exception of any intentionally randomized functions). Images must be loaded accurately and saved without any errors or file corruption.

3.2.3 Security

The app has no network functionality, so file privacy is not a concern. The security of saved files will be handled by the security protocols of the user's operating system.

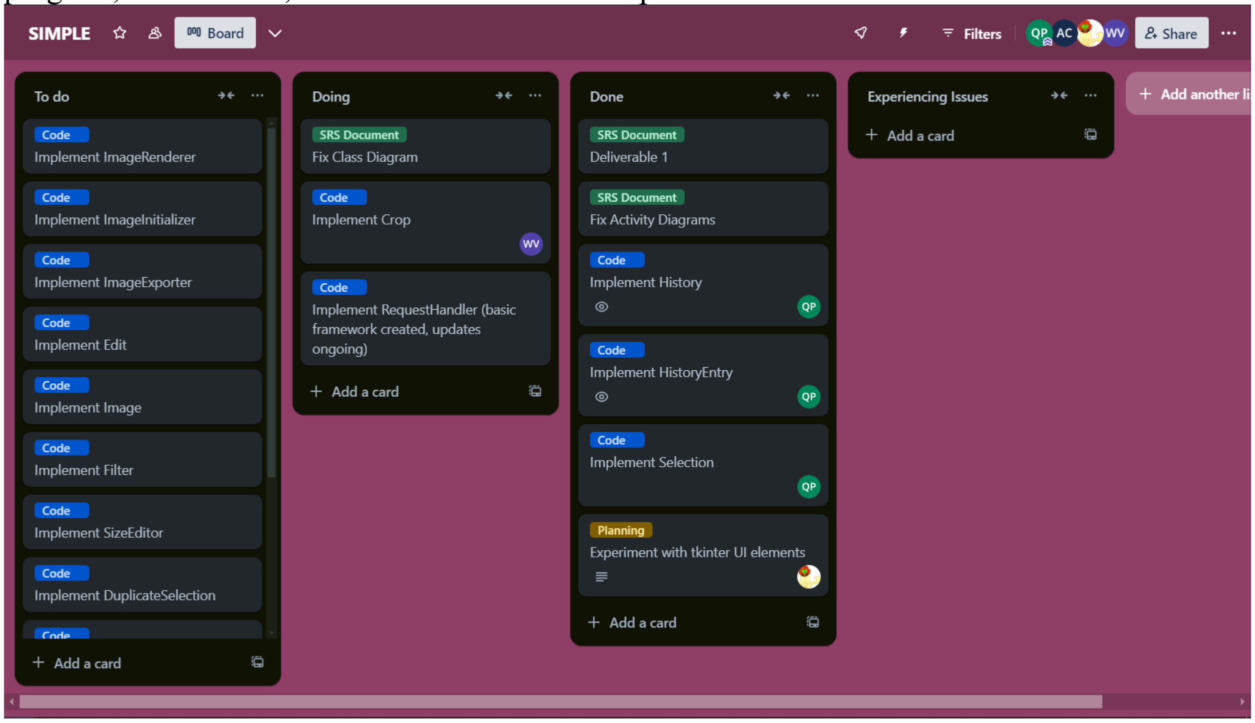
3.2.4 Portability

The app will be contained in a standalone executable file, allowing it to run on any supported Windows system.

4. Design & Development

4.1 Software Development Process

The process model that we have chosen for the development of our application is the Kanban model. This model will allow us to grow our application iteratively and incrementally based on task progress. We plan on having a shared Kanban board where we can see the advancement of each task assigned to each member. This will allow us to divide our work. We also plan on having a weekly meeting on Thursday afternoons at 2 pm, after class, where we will discuss our progress, review code, and determine our next steps.



To do
- Code: Implement ImageRenderer
- Code: Implement ImageInitializer
- Code: Implement ImageExporter
- Code: Implement Edit
- Code: Implement Image
- Code: Implement Filter
- Code: Implement SizeEditor
- Code: Implement DuplicateSelection
- Code
- Add a card

Doing
- SRS Document: Fix Class Diagram
- Code: Implement Crop (WV)
- Code: Implement RequestHandler (basic framework created, updates ongoing)
- Add a card

Done
- SRS Document: Deliverable 1
- SRS Document: Fix Activity Diagrams
- Code: Implement History (QP)
- Code: Implement HistoryEntry (QP)
- Code: Implement Selection (QP)
- Planning: Experiment with tkinter UI elements
- Add a card

Experiencing Issues
- Add a card

Add another li

Figure 1: SIMPLE Kanban Board

4.2 Deliverable 1

4.2.1 Description

For this deliverable, we started with correcting parts 1-3 of the SRS document after the feedback we received from our previous submission. We also determined which process model would best aid our development and started creation of the Kanban board and Class Diagram. Additionally, each team member was tasked with creating 2 Activity Diagrams to illustrate a select number of our Use Cases. Finally, we started development on the application by setting up a GitHub repository with some starting code to learn the Python libraries we are utilizing.

4.2.2 Design

4.2.2.1 Class Diagram

The class diagram is shown in Figure 1 below, followed by explanations of each class.

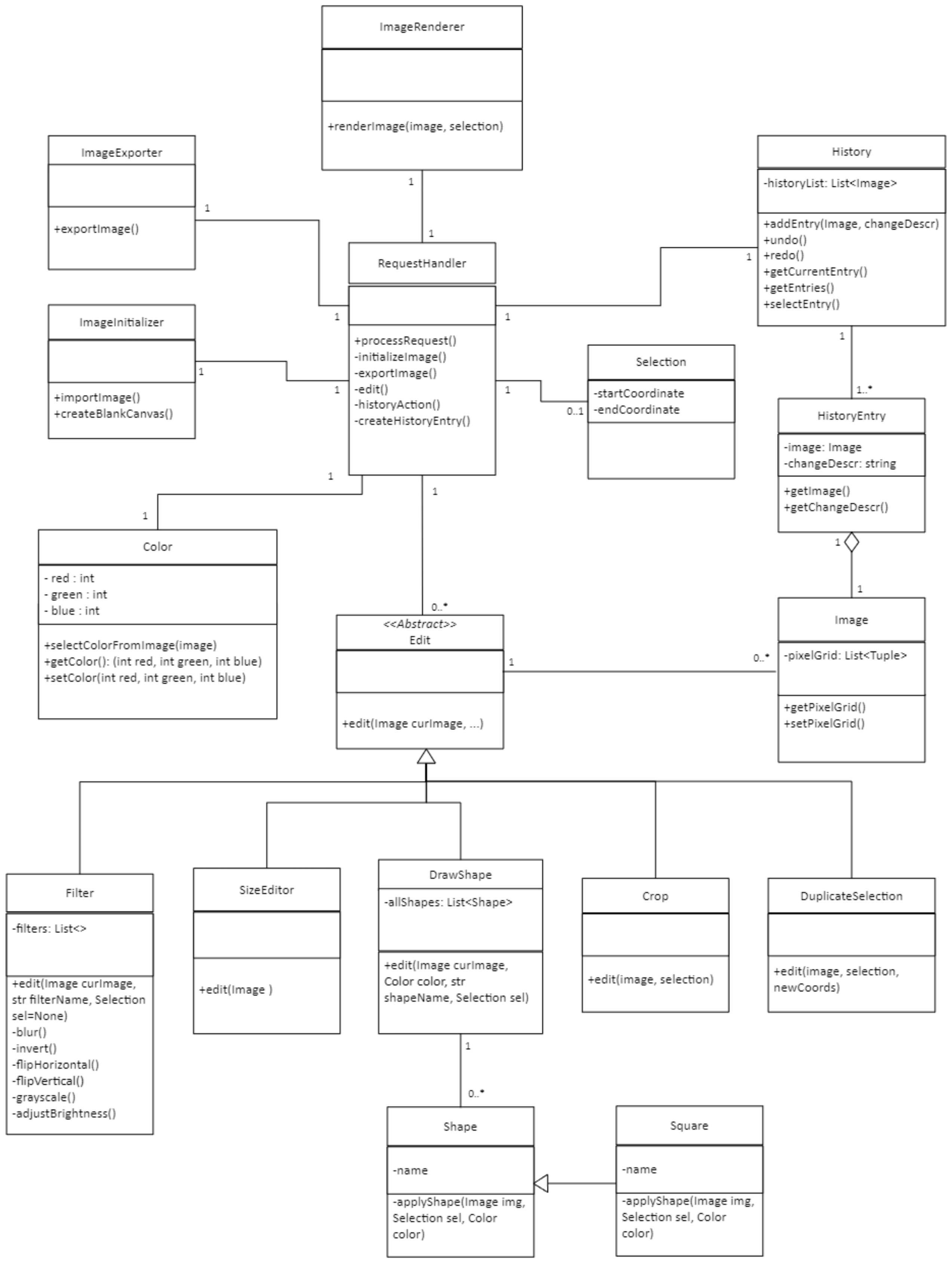


Figure 2: SIMPLE Class Diagram

The RequestHandler class manages the use cases and dispatches requests from the user for the corresponding classes to fulfill.

Image handles an instance of the image that is being manipulated.

ImageInitializer has the job of creating a blank canvas or importing an image in for the user.

ImageExporter deals with exporting the image to the user's local storage.

History stores and manages all historical versions of the image.

HistoryEntry represents the version of the image after a certain edit action occurs.

ImageRenderer displays the most recent history entry to the user, along with any special temporary effects such as the selection box.

The Selection class holds the coordinates of the selected area and is passed by the RequestHandler to edit that need the selected area.

Edit is an abstract superclass for edits that are applied to the image. All the following classes are subclasses of Edit (or related to its subclasses).

The SizeEditor class edits the size of the image based on a factor passed by the RequestHandler.

The Crop class crops the image based on the selected area passed by the RequestHandler.

The DuplicateSelection class duplicates the selected area to the specified location based on the arguments passed by the RequestHandler.

Filter contains a list of filter functions. RequestHandler reads the list of filters from the Filter class, from which the user can select and then RequestHandler will pass the desired filter down to the Filter class (along with the Image and the Selection, if applicable) to apply the filter.

DrawShape contains a list of Shapes (such as Square), which each handle the drawing of their respective shapes within a given area. RequestHandler reads the list of Shapes from the DrawShape class, from which the user can select and then RequestHandler will pass the desired Shape down to the DrawShape class (along with the Image, Selection, and Color) to draw the shape.

4.2.2.2 Activity Diagram

The activity diagram shown in Figure 2 explains the activities involved and their flow for importing an image (Use case 2: Import Image).

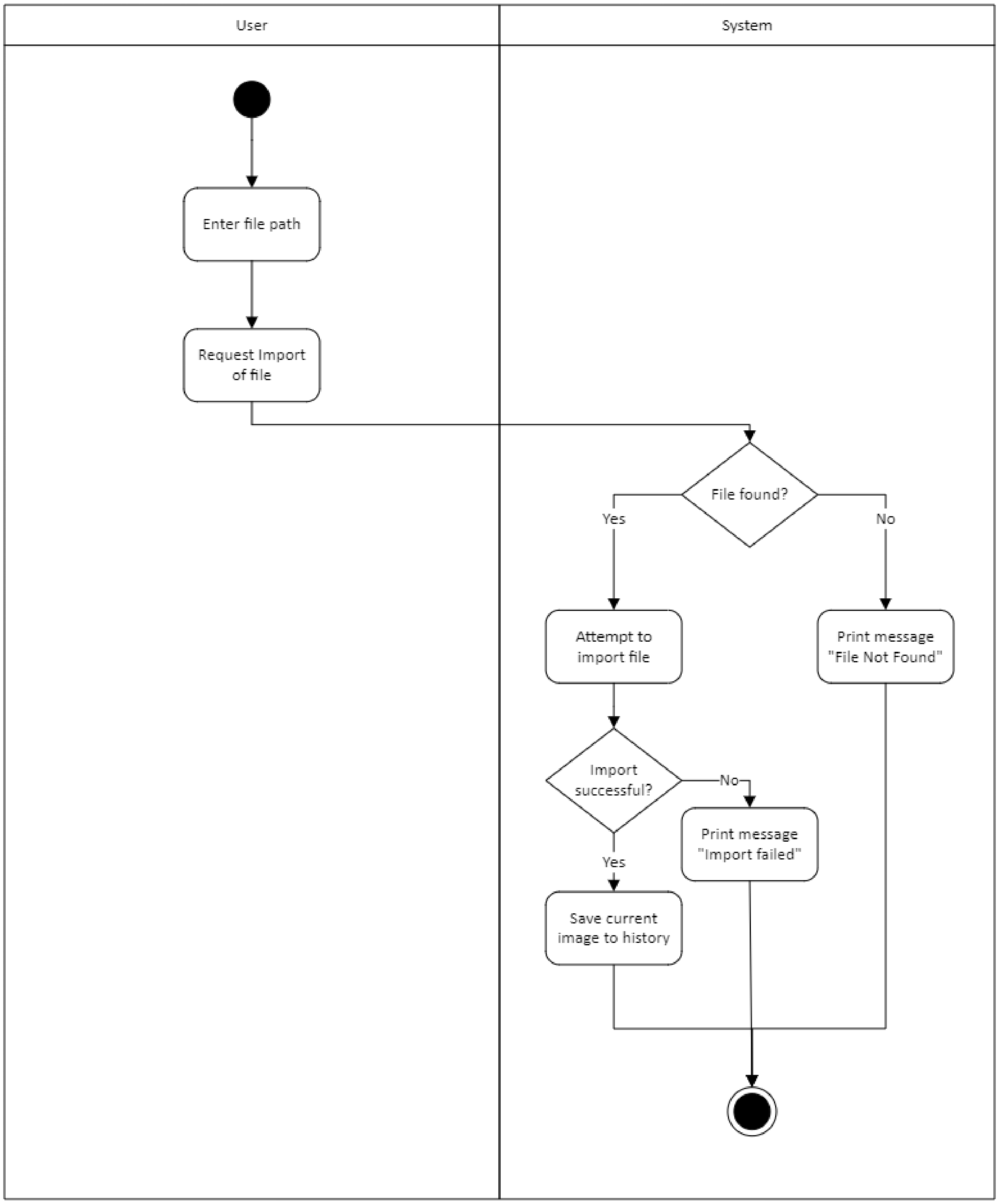


Figure 3: Import Image Activity Diagram

The activity diagram shown in Figure 3 explains the activities involved and their flow for exporting an image (Use case 3: Export Image).

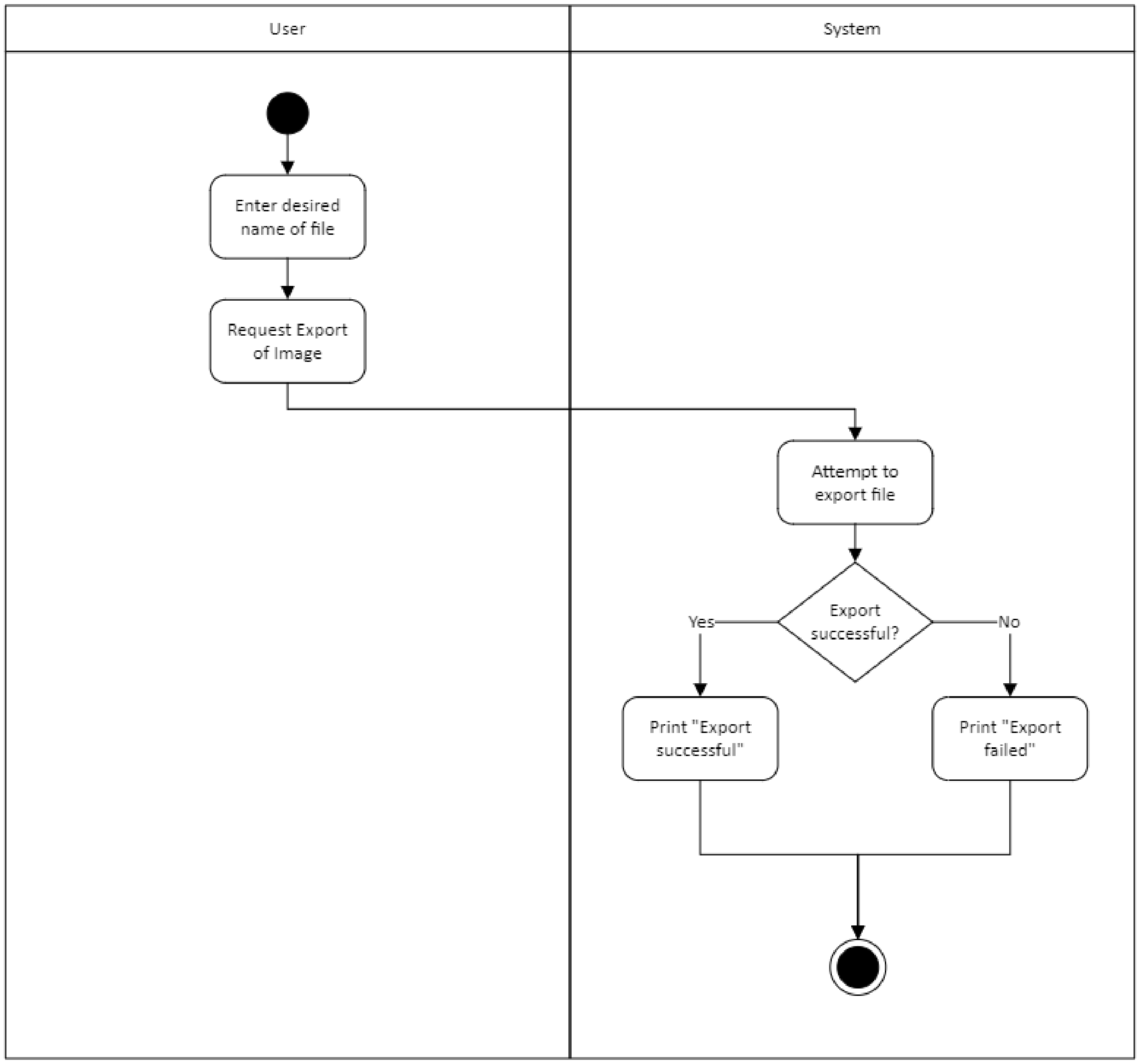


Figure 4: Export Image Activity Diagram

The activity diagram shown in Figure 4 explains the activities involved and their flow for selecting an area of an image (Use case 4: Select Area).

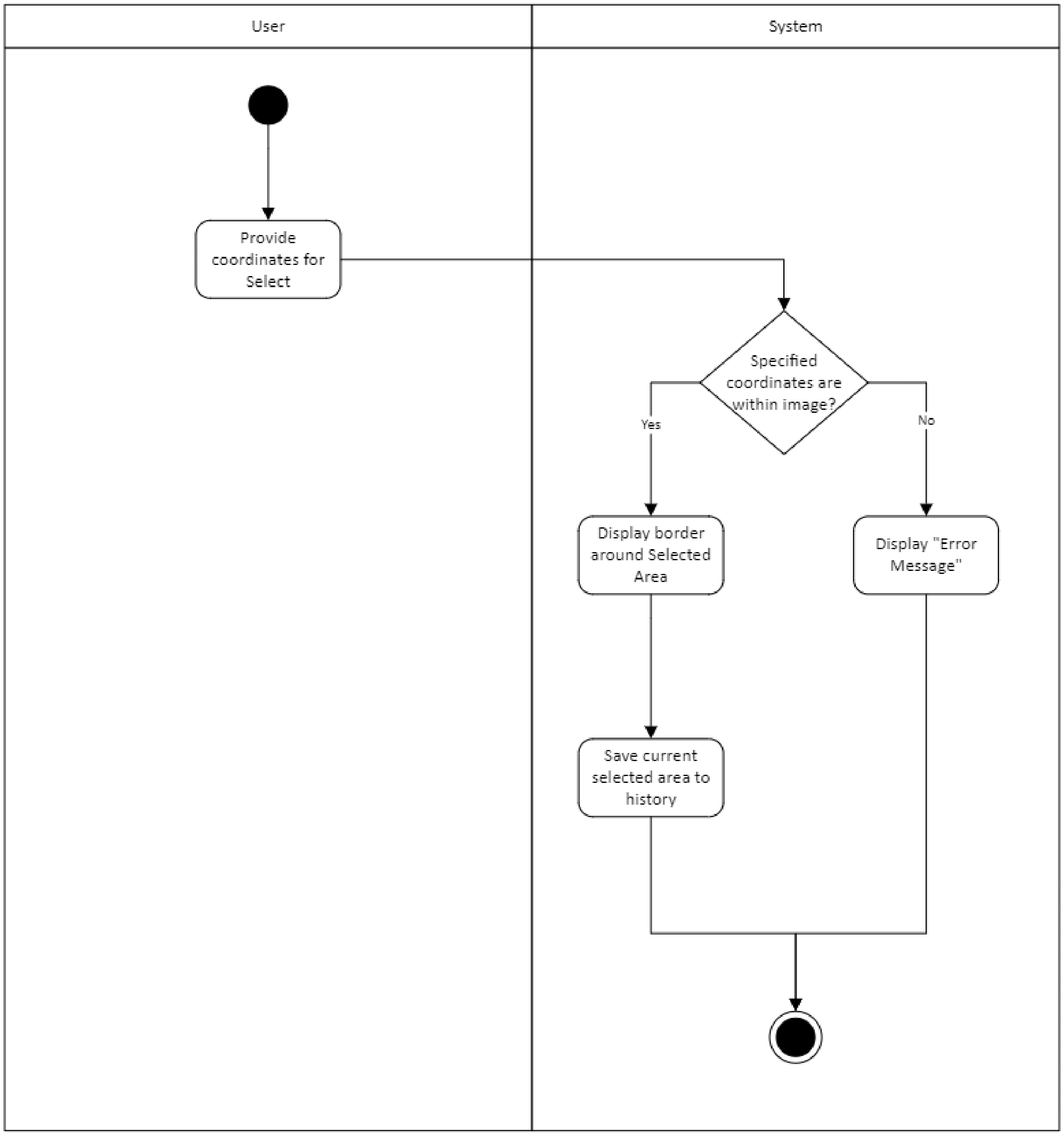

User
System
Provide coordinates for Select
Specified coordinates are within image?
Yes
No
Display border around Selected Area
Display "Error Message"
Save current selected area to history

Figure 5: Select Area Activity Diagram

The activity diagram shown in Figure 5 explains the activities involved and their flow for cropping an image (Use case 6: Crop).

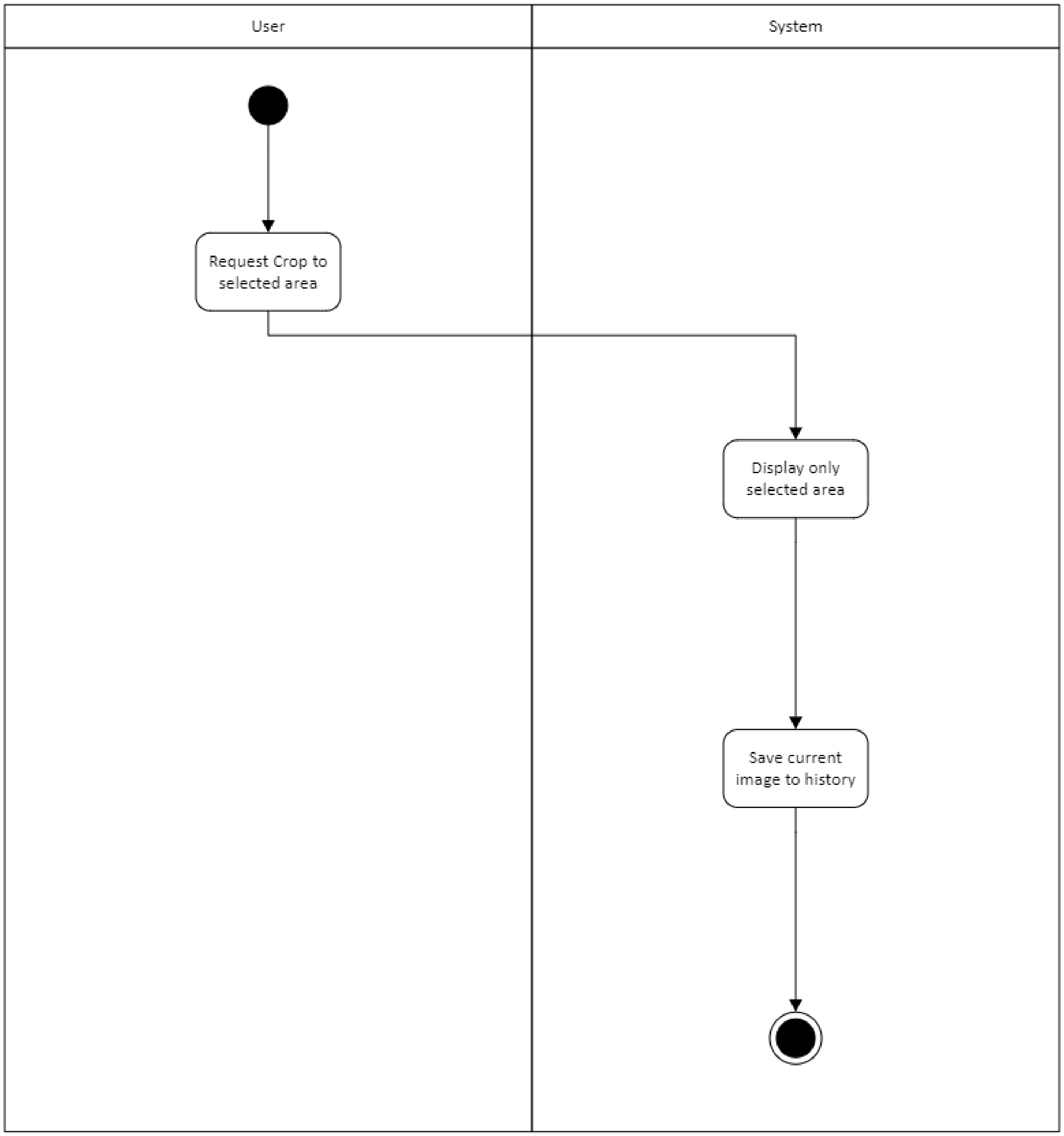

User
System
Request Crop to selected area
Display only selected area
Save current image to history

Figure 6: Crop Activity Diagram

The activity diagram shown in Figure 6 explains the activities involved and their flow for applying a filter to an image (Use case 7: Apply Filter).

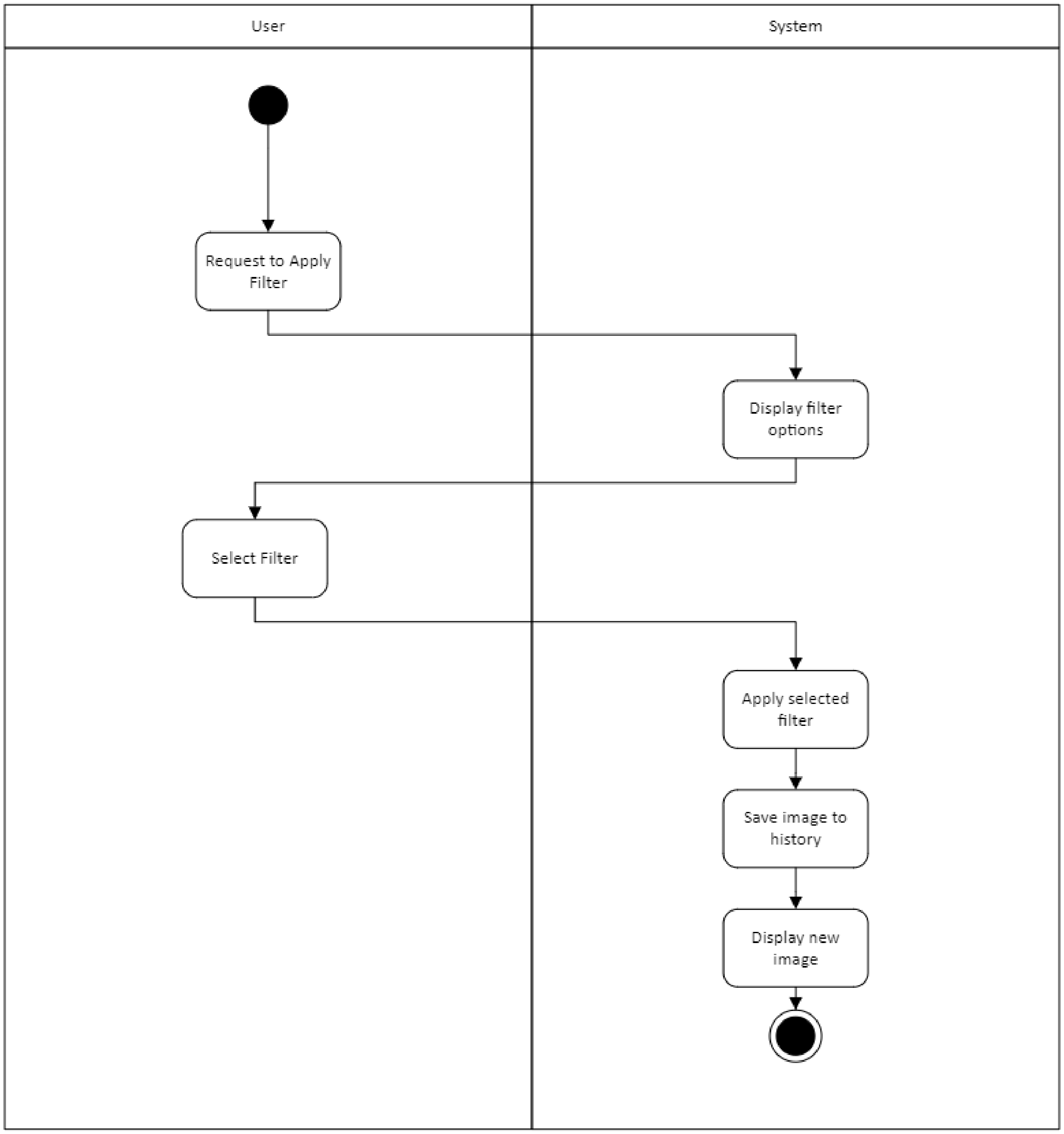


Figure 7: Apply Filter Activity Diagram

The activity diagram shown in Figure 7 explains the activities involved and their flow for identifying a color from an image (Use case 10: Identify Color from Image).

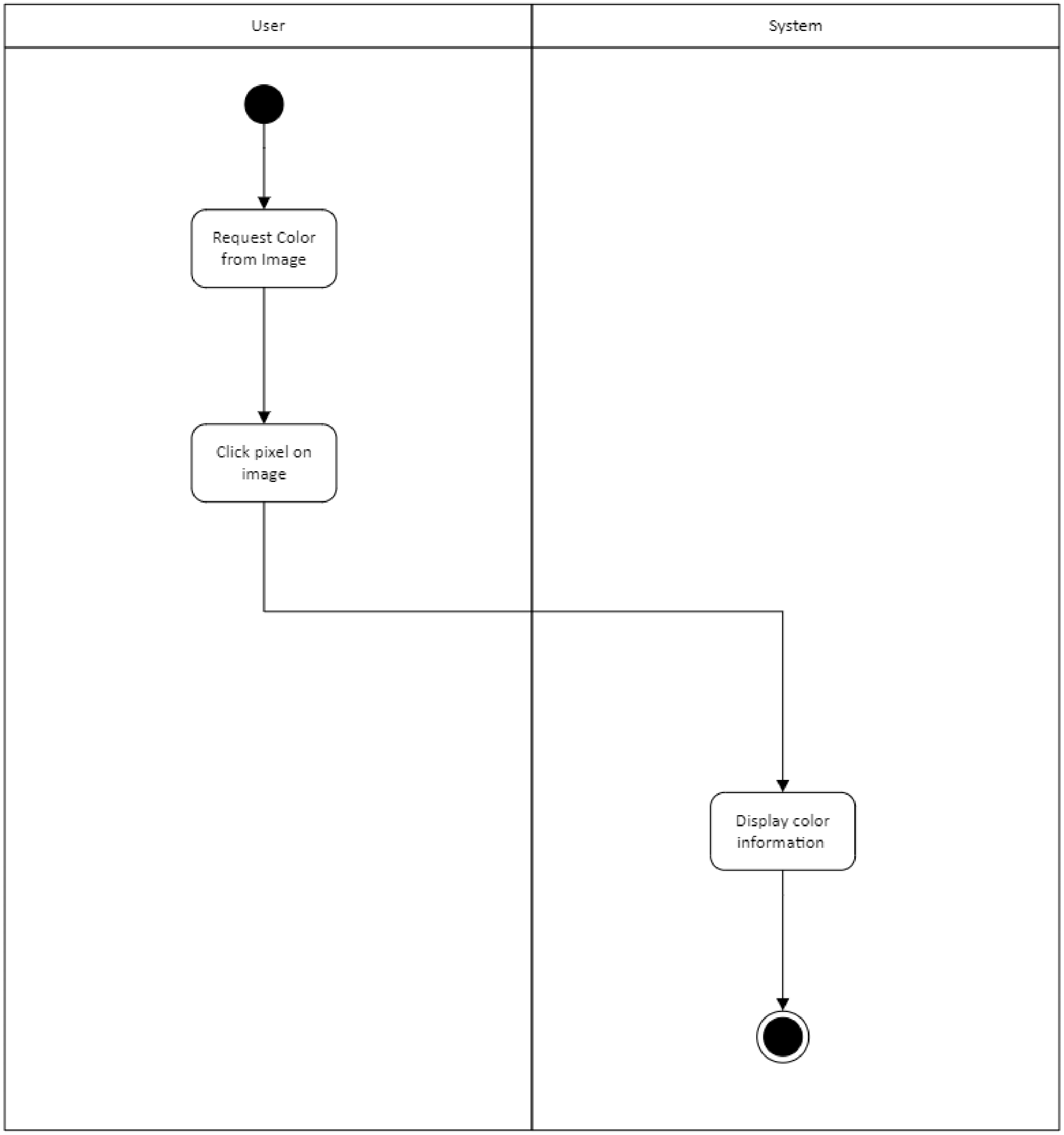


Figure 8: Identify Color from Image Activity Diagram

The activity diagram shown in Figure 8 explains the activities involved and their flow for undoing an edit (Use case 14: Undo).

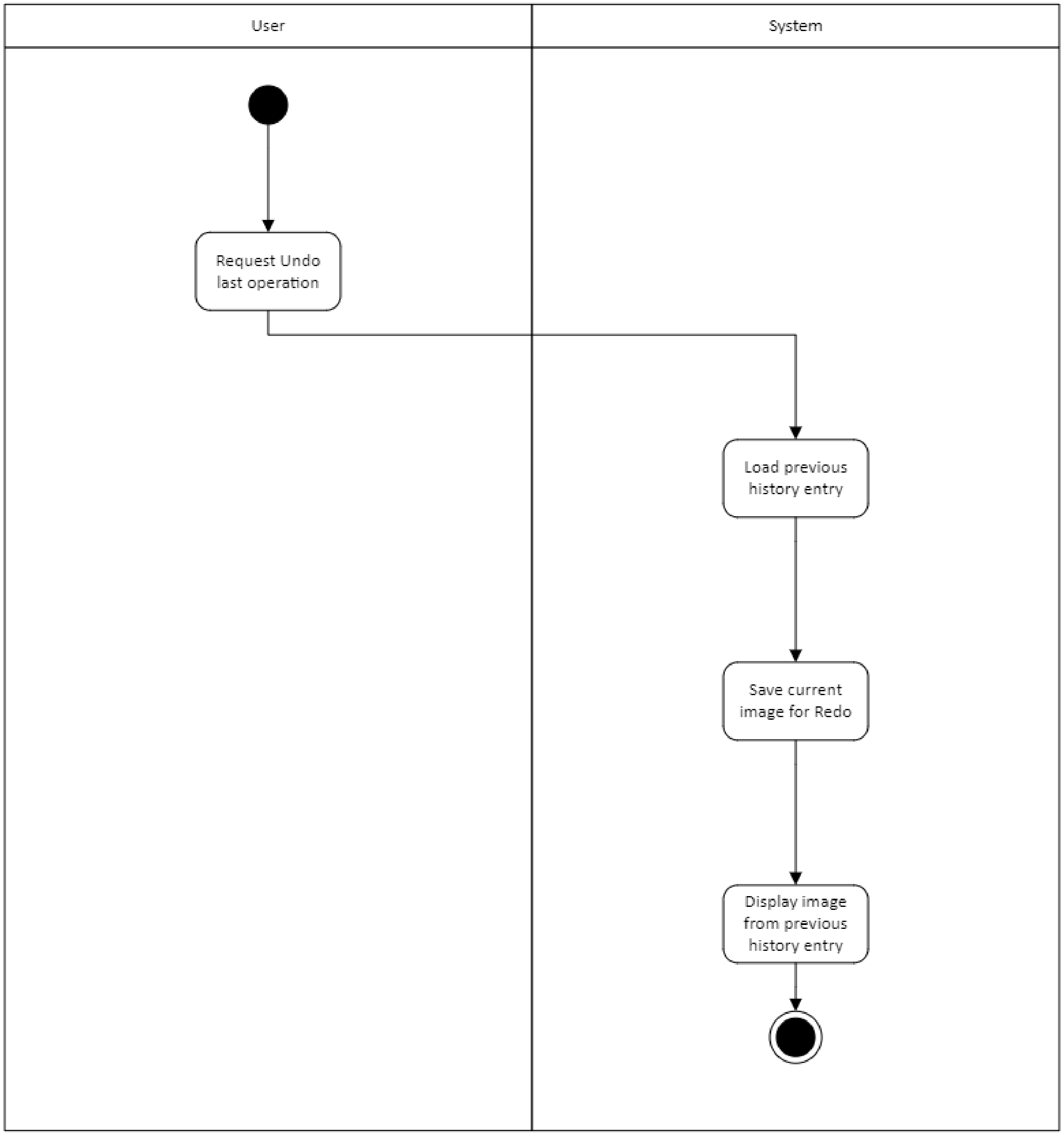

User
System
Request Undo last operation
Load previous history entry
Save current image for Redo
Display image from previous history entry

Figure 9: Undo Activity Diagram

The activity diagram shown in Figure 9 explains the activities involved and their flow for viewing the edit history (Use case 16: View History).

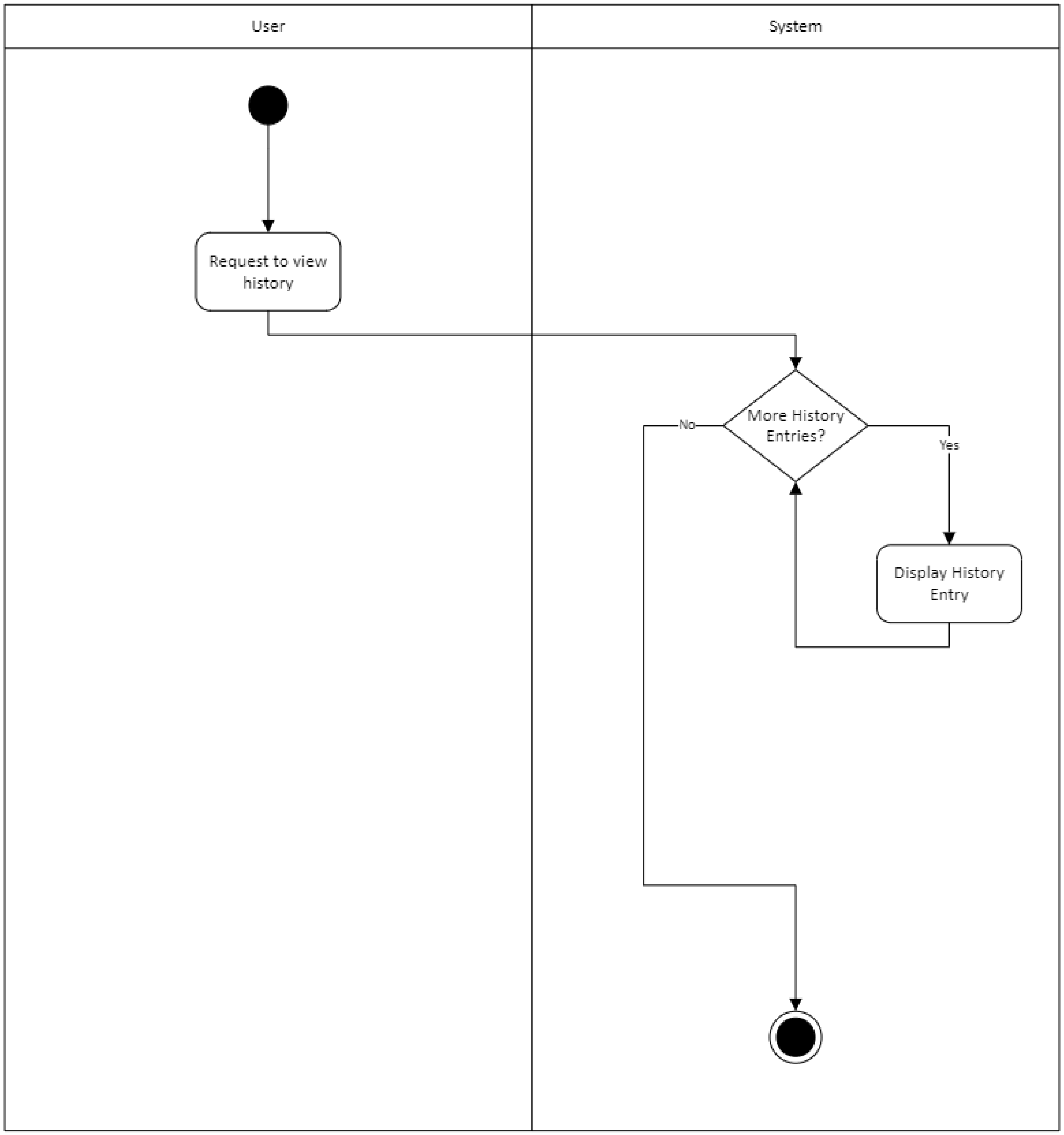


Figure 10: View History Activity Diagram

4.2.3 Development

4.2.3.1 Description

We have started development on the application by setting up a GitHub repository with some starting code to learn the Python libraries we are utilizing. We have also invited Rishi to view our repository. Link to the private repository: https://github.com/GameDJ/image-editor

4.3 Deliverable 2

4.3.1 Description

For this deliverable, we fixed the Activity Diagrams, updated them in the document, and updated the Class Diagram according to the feedback we received on Deliverable 1. We also started working on implementing the code for the classes from the updated Class Diagram. Implementation for many of these classes can now be found in our GitHub repository.

4.3.2 Design

4.3.2.1 Class Diagram

The updated class diagram is shown in Figure 11 below.

The RequestHandler class manages the use cases and dispatches requests from the user for the corresponding classes to fulfill.

Image handles an instance of the image that is being manipulated.

ImageInitializer has the job of creating a blank canvas or importing an image in for the user. This is an abstract class that has concrete implementations for each image type. The image types supported by SIMPLE are PNG and JPEG and are handled by PngInitializer and JpegInitializer.

ImageExporter deals with exporting the image to the user's local storage. This is an abstract class that has concrete implementations for each image type. The image types supported by SIMPLE are PNG and JPEG and are handled by PngExporter and JpegExporter.

History stores and manages all historical versions of the image.

HistoryEntry represents the version of the image after a certain edit action occurs.

Renderer is an abstract class that renders an image to be displayed to the user.

ImagerRenderer renders an unmodified version of an image.

RenderAddOn is an abstract class for rendering an image to be displayed to the user with additional special effects.

SelectionRenderer renders the image with additional special effects representing the current selection boundary.

Selection holds the coordinates of the selected area and is passed by the RequestHandler to edits that target a selection area.

Color holds the red, green, and blue values of the color of a pixel selected from the Image or chosen for DrawShape.

Arguments holds all additional values, aside from the Image, passed to the edit method that determine which edit will be made, and it contains additional information required for the desired edit. An instance can be defined using the builder pattern to chain desired attributes.

Edit is an abstract superclass for edits that are applied to the image. All the following classes are subclasses of Edit (or related to its subclasses).

The SizeEditor class edits the size of the image based on a factor passed by the RequestHandler.

The Crop class crops the image based on the selected area passed by the RequestHandler.

The DuplicateSelection class duplicates the selected area to the specified location based on the arguments passed by the RequestHandler.

Filter contains a list of filter functions. RequestHandler reads the list of filters from the Filter class, from which the user can select and then RequestHandler will pass the desired filter down to the Filter class (along with the Image and the Selection, if applicable) to apply the filter.

DrawShape contains a list of Shapes (such as Rectangle), which each handle the drawing of their respective shapes within a given area. RequestHandler reads the list of Shapes from the DrawShape class, from which the user can select and then RequestHandler will pass the desired Shape down to the DrawShape class (along with the Image, Selection, and Color) to draw the shape.

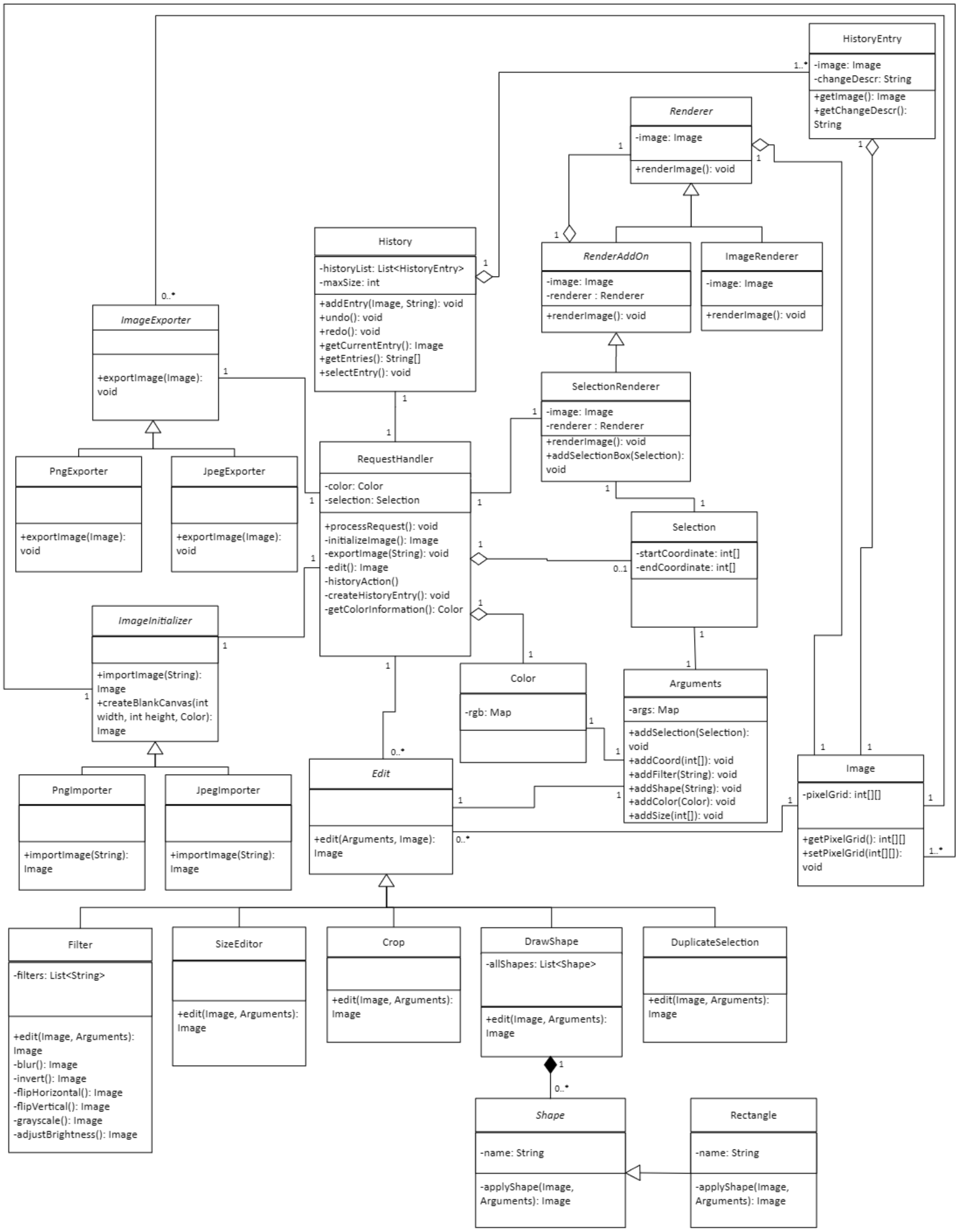
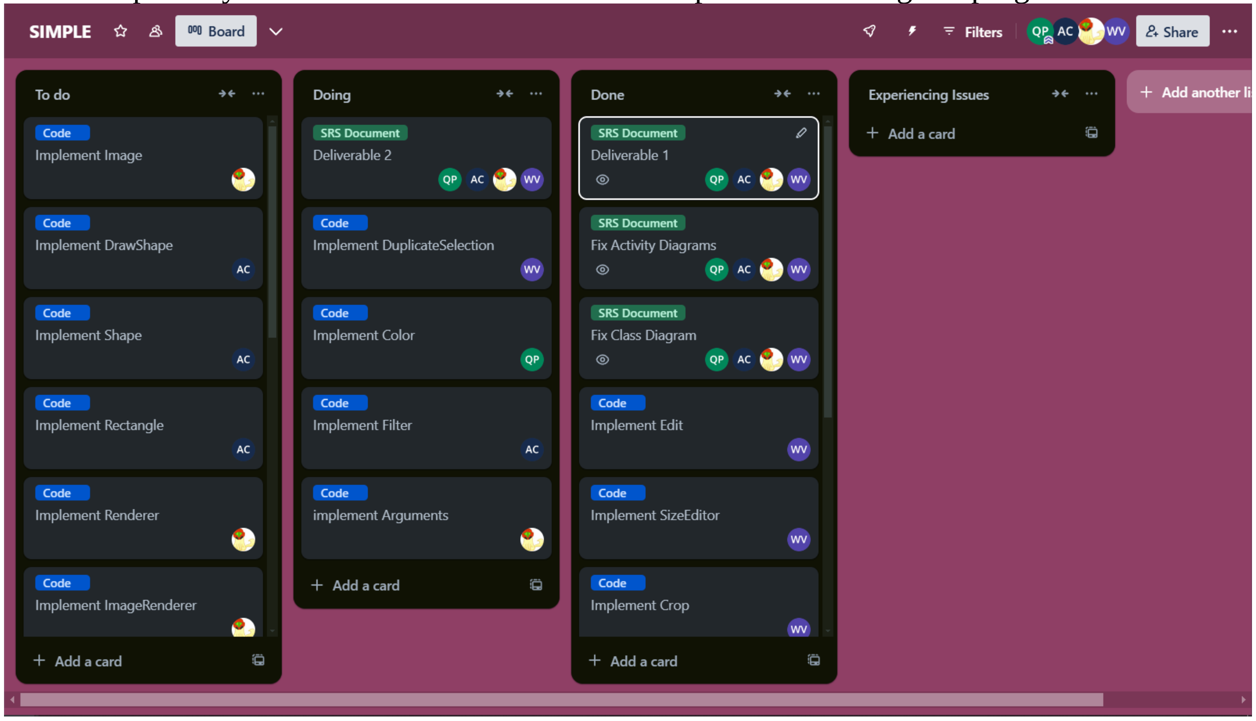


Figure 11: Updated SIMPLE Class Diagram

4.3.3 Development

4.3.3.1 Description

We have made progress in implementing the code for the classes according to the class diagram in Figure 11. Implementation for many of these classes can now be found in our GitHub repository. Our Kanban board has also been updated following our progress.



To do

- Code — Implement Image
- Code — Implement DrawShape — AC
- Code — Implement Shape — AC
- Code — Implement Rectangle — AC
- Code — Implement Renderer
- Code — Implement ImageRenderer
- + Add a card

Doing

- SRS Document — Deliverable 2 — QP AC WV
- Code — Implement DuplicateSelection — WV
- Code — Implement Color — QP
- Code — Implement Filter — AC
- Code — implement Arguments
- + Add a card

Done

- SRS Document — Deliverable 1 — QP AC WV
- SRS Document — Fix Activity Diagrams — QP AC WV
- SRS Document — Fix Class Diagram — QP AC WV
- Code — Implement Edit — WV
- Code — Implement SizeEditor — WV
- Code — Implement Crop — WV
- + Add a card

Experiencing Issues

- + Add a card

+ Add another li

Figure 12: Updated SIMPLE Kanban Board

4.4 Deliverable 3

4.4.1 Description

For this deliverable, we updated the Class Diagram according to the feedback we received on Deliverable 2. We also created Sequence Diagrams for four of the use cases and continued working on implementing the classes as shown in the Class Diagram. Our GitHub repository reflects the updates and new implementations we have made since Deliverable 2.

4.4.2 Design

4.4.2.1 Class Diagram

The updated class diagram is shown in Figure 13 below.

The FilterType enumeration is used to determine which filter has been chosen by the user to apply to the image.

The ShapeType enumeration is used to determine which shape has been chosen by the user to draw on the image.

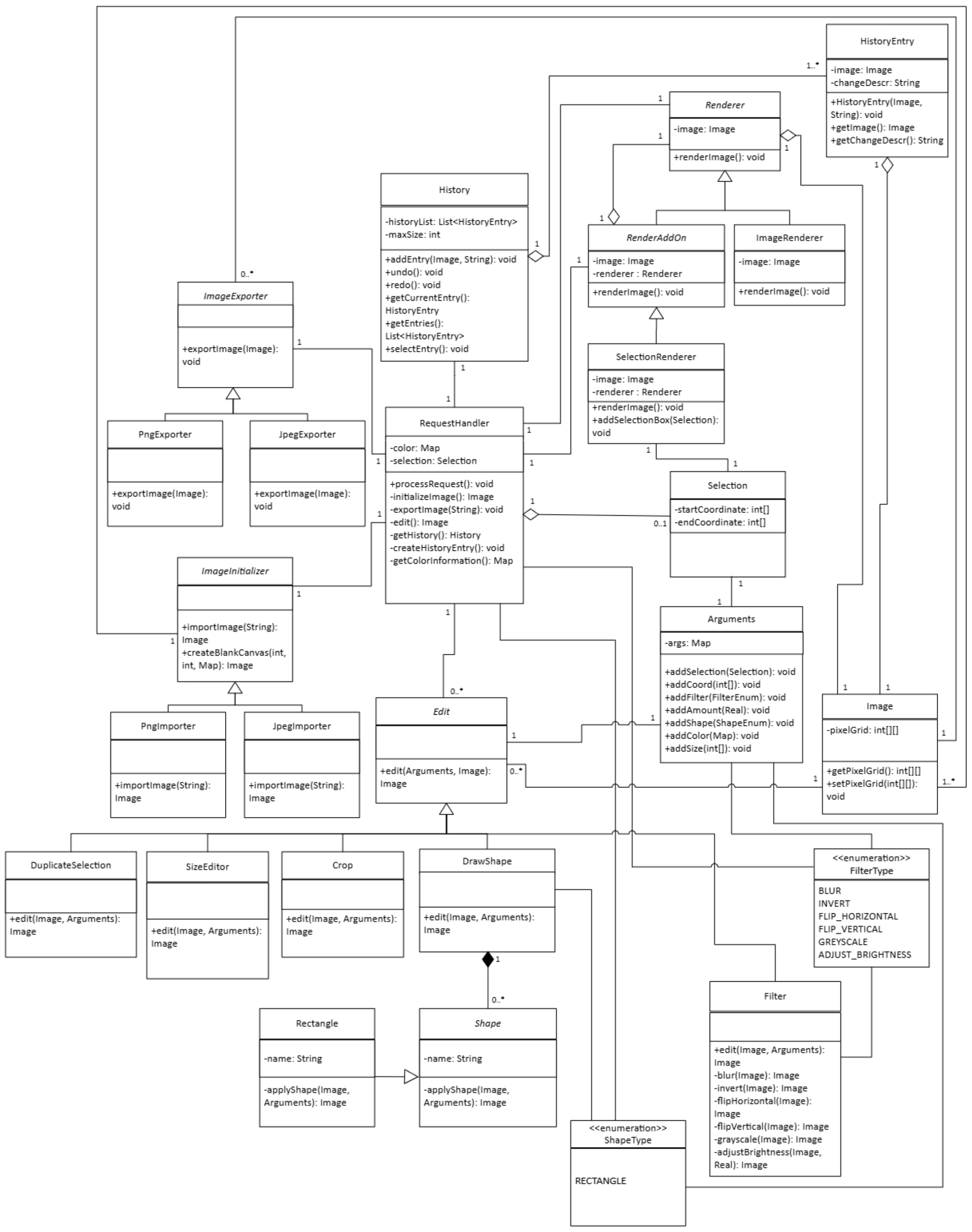

HistoryEntry
-image: Image
-changeDescr: String
+HistoryEntry(Image, String): void
+getImage(): Image
+getChangeDescr(): String
Renderer
-image: Image
+renderImage(): void
History
-historyList: List<HistoryEntry>
-maxSize: int
+addEntry(Image, String): void
+undo(): void
+redo(): void
+getCurrentEntry(): HistoryEntry
+getEntries(): List<HistoryEntry>
+selectEntry(): void
RenderAddOn
-image: Image
-renderer : Renderer
+renderImage(): void
ImageRenderer
-image: Image
+renderImage(): void
ImageExporter
+exportImage(Image): void
SelectionRenderer
-image: Image
-renderer : Renderer
+renderImage(): void
+addSelectionBox(Selection): void
PngExporter
+exportImage(Image): void
JpegExporter
+exportImage(Image): void
RequestHandler
-color: Map
-selection: Selection
+processRequest(): void
-initializeImage(): Image
-exportImage(String): void
-edit(): Image
-getHistory(): History
-createHistoryEntry(): void
-getColorInformation(): Map
Selection
-startCoordinate: int[]
-endCoordinate: int[]
ImageInitializer
+importImage(String): Image
+createBlankCanvas(int, int, Map): Image
Arguments
-args: Map
+addSelection(Selection): void
+addCoord(int[]): void
+addFilter(FilterEnum): void
+addAmount(Real): void
+addShape(ShapeEnum): void
+addColor(Map): void
+addSize(int[]): void
Image
-pixelGrid: int[][]
+getPixelGrid(): int[][]
+setPixelGrid(int[][]): void
PngImporter
+importImage(String): Image
JpegImporter
+importImage(String): Image
Edit
+edit(Arguments, Image): Image
DuplicateSelection
+edit(Image, Arguments): Image
SizeEditor
+edit(Image, Arguments): Image
Crop
+edit(Image, Arguments): Image
DrawShape
+edit(Image, Arguments): Image
<<enumeration>> FilterType
BLUR
INVERT
FLIP_HORIZONTAL
FLIP_VERTICAL
GREYSCALE
ADJUST_BRIGHTNESS
Rectangle
-name: String
-applyShape(Image, Arguments): Image
Shape
-name: String
-applyShape(Image, Arguments): Image
Filter
+edit(Image, Arguments): Image
-blur(Image): Image
-invert(Image): Image
-flipHorizontal(Image): Image
-flipVertical(Image): Image
-grayscale(Image): Image
-adjustBrightness(Image, Real): Image
<<enumeration>> ShapeType
RECTANGLE
1
1..*
0..*
0..1

Figure 13: Updated SIMPLE Class Diagram

4.4.2.2 Sequence Diagrams

The sequence diagram shown in Figure 14 explains the messages and objects involved in importing an image (Use case 2: Import Image).

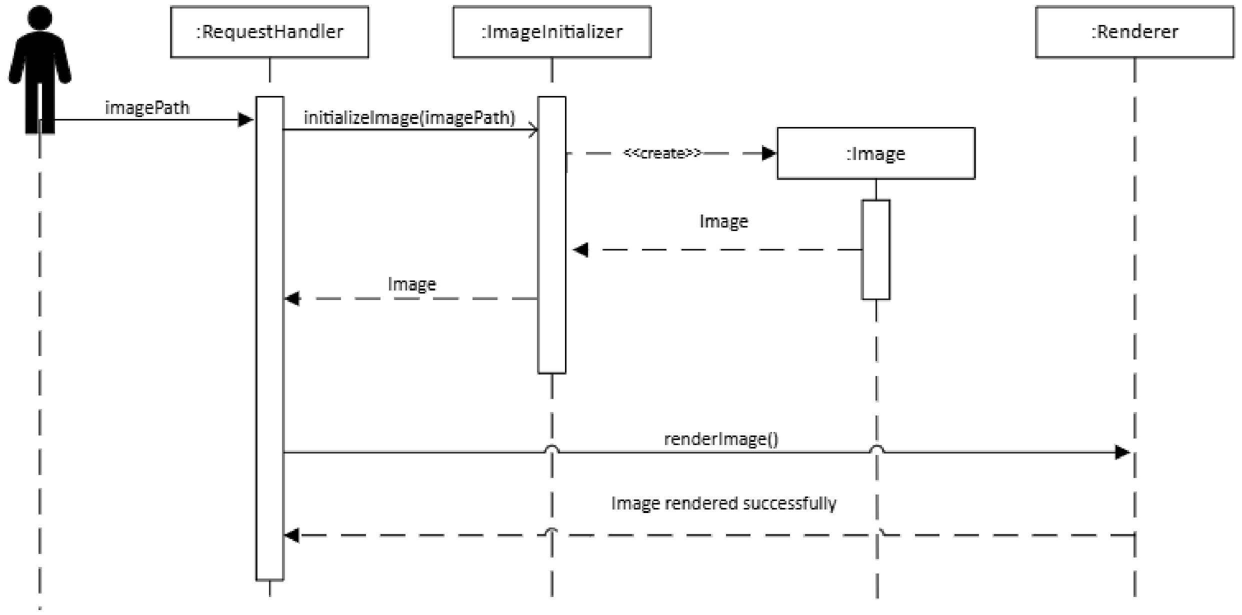


Figure 14: Import Image Sequence Diagram

The sequence diagram shown in Figure 15 explains the messages and objects involved in cropping an image (Use case 6: Crop Image).

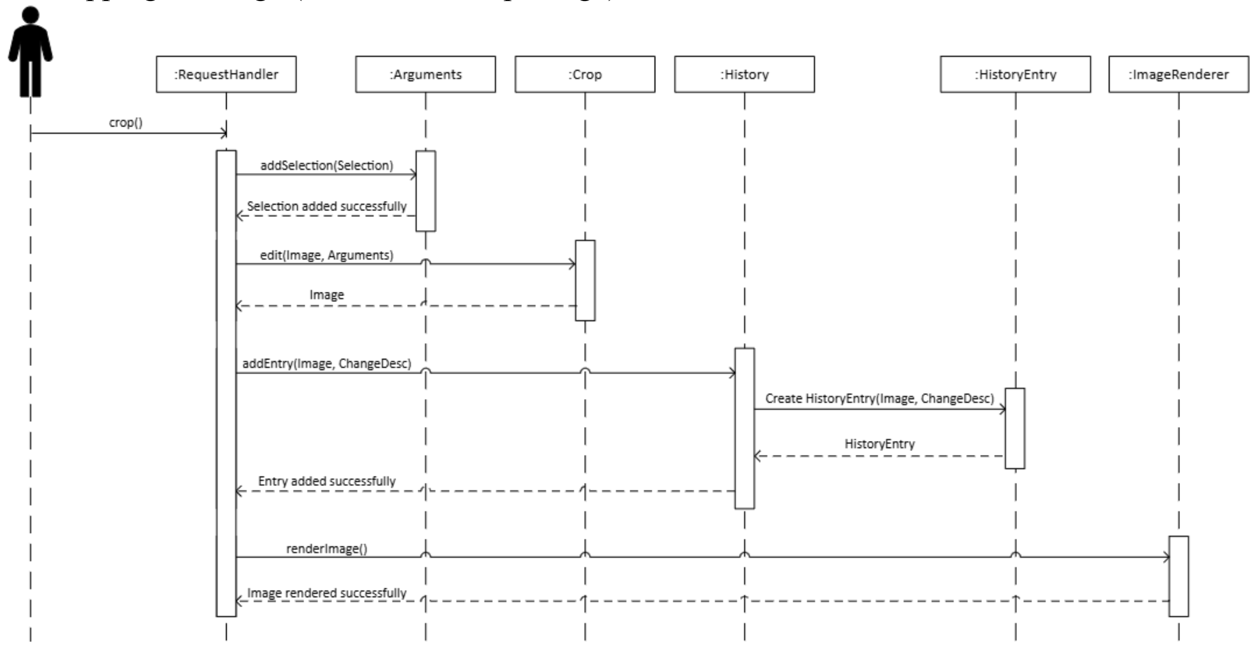


Figure 15: Crop Image Sequence Diagram

The sequence diagram shown in Figure 16 explains the messages and objects involved in applying a filter to an image (Use case 7: Apply Filter).

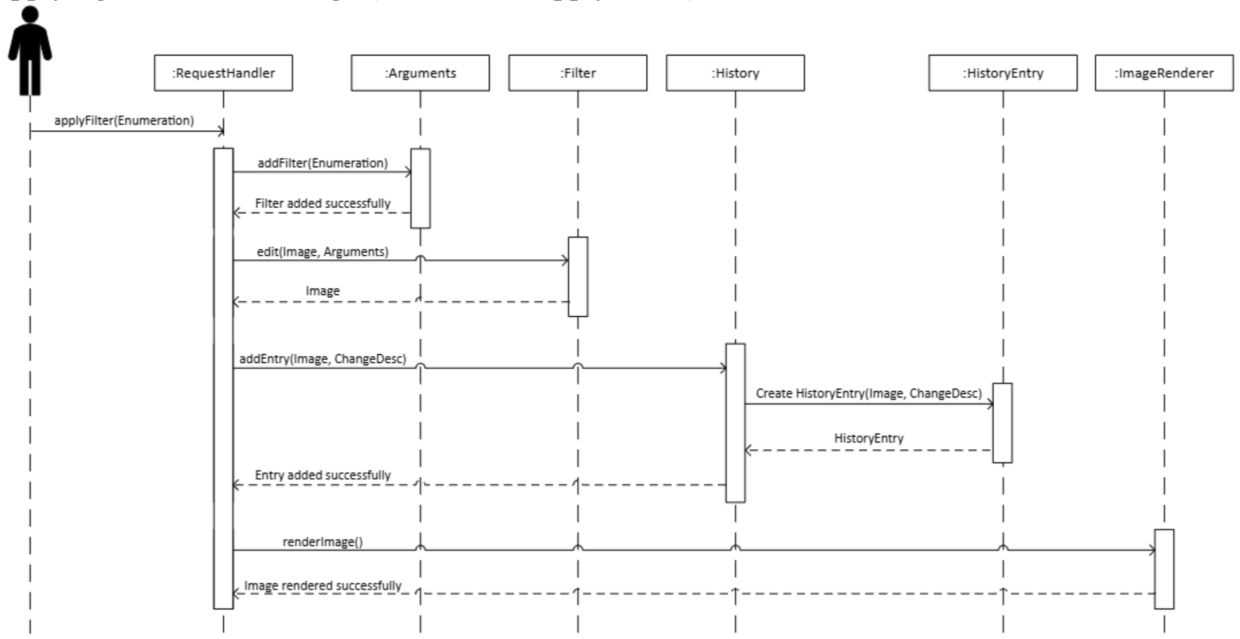


Figure 16: Apply Filter Sequence Diagram

The sequence diagram shown in Figure 17 explains the messages and objects involved in identifying a color from an image (Use case 10: Identify Color from Image).

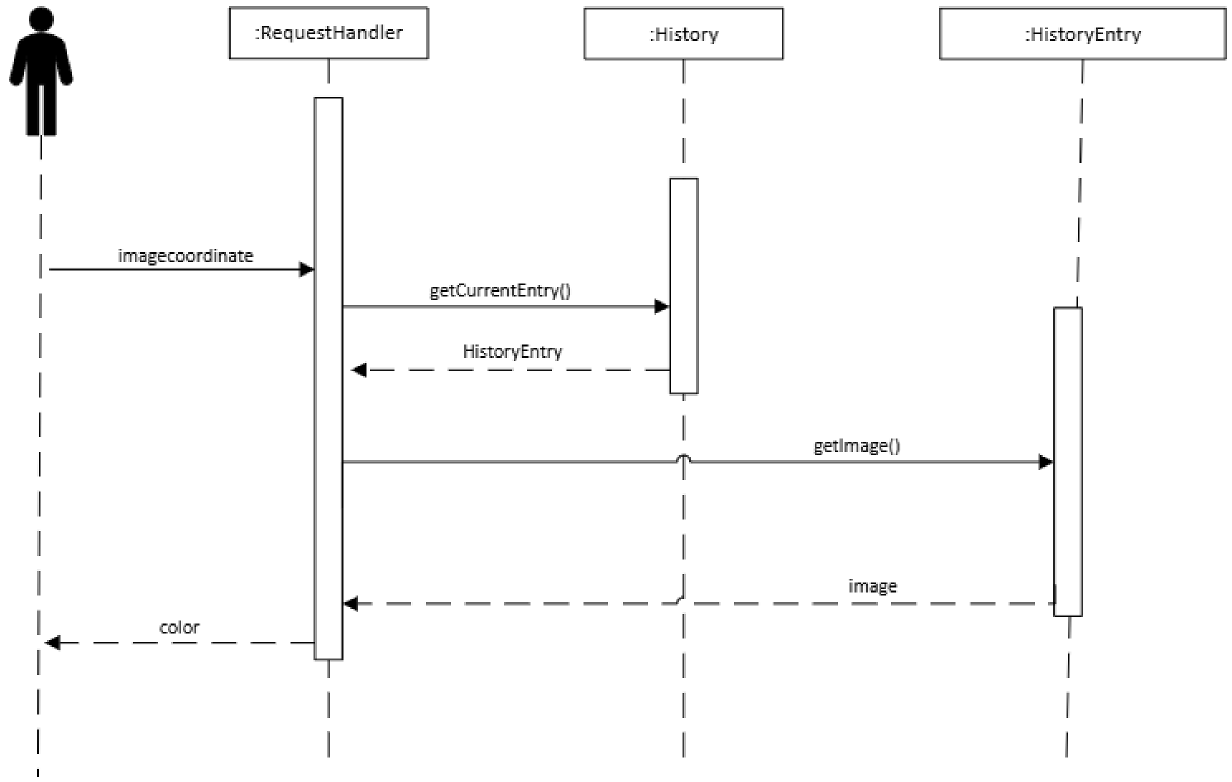
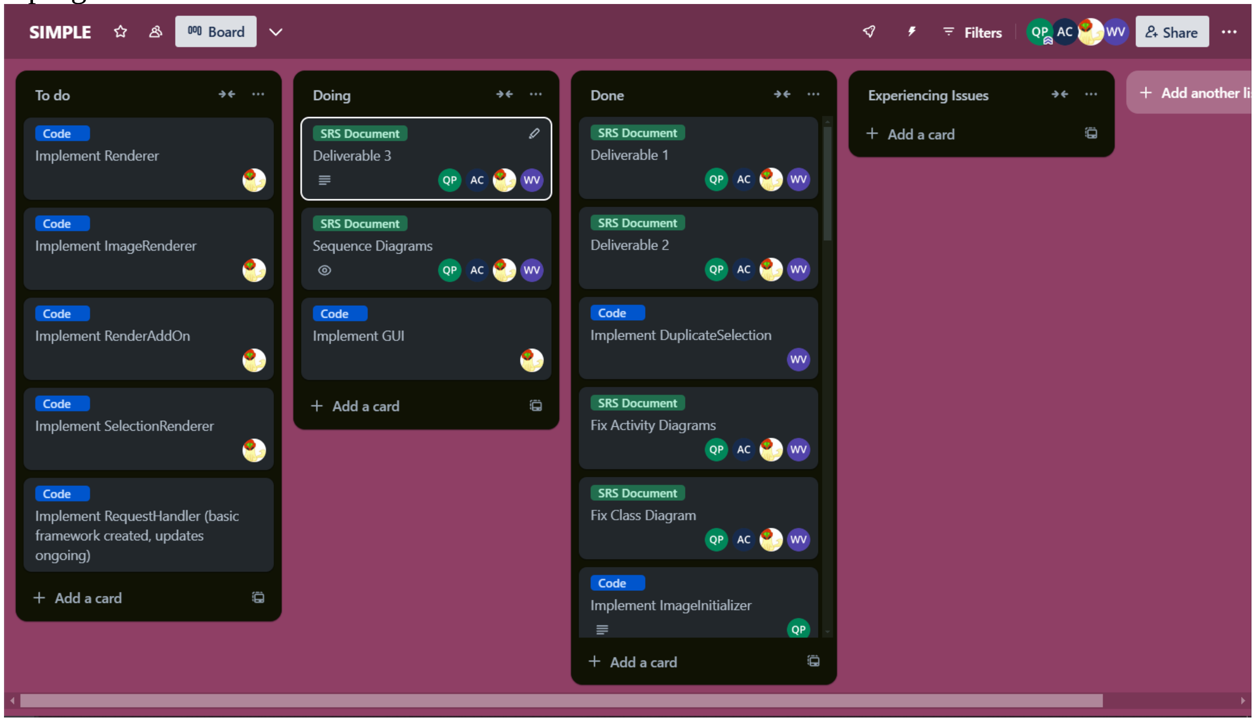


Figure 17: Identify Color from Image Sequence Diagram

4.4.3 Development

4.4.3.1 Description

We have continued to make progress in implementing the code for the classes according to the class diagram in Figure 13. Implementation for and updates to many of these classes can now be found in our GitHub repository. Our Kanban board has also been updated following our progress.



To do
- Code — Implement Renderer
- Code — Implement ImageRenderer
- Code — Implement RenderAddOn
- Code — Implement SelectionRenderer
- Code — Implement RequestHandler (basic framework created, updates ongoing)
- Add a card

Doing
- SRS Document — Deliverable 3
- SRS Document — Sequence Diagrams
- Code — Implement GUI
- Add a card

Done
- SRS Document — Deliverable 1
- SRS Document — Deliverable 2
- Code — Implement DuplicateSelection
- SRS Document — Fix Activity Diagrams
- SRS Document — Fix Class Diagram
- Code — Implement ImageInitializer
- Add a card

Experiencing Issues
- Add a card

Add another li

Figure 18: Updated SIMPLE Kanban Board

4.5 Deliverable 4

4.5.1 Description

For this deliverable, we fixed the Sequence Diagrams, updated them in the document, and updated the Class Diagram according to the feedback we received on Deliverable 3. We have also completed development on implementing the classes as shown in the Class Diagram. Our GitHub repository reflects the updates and new implementations we have made since Deliverable 3.

4.5.2 Design

4.5.2.1 Class Diagram

The updated class diagram is shown in Figure 19 below.

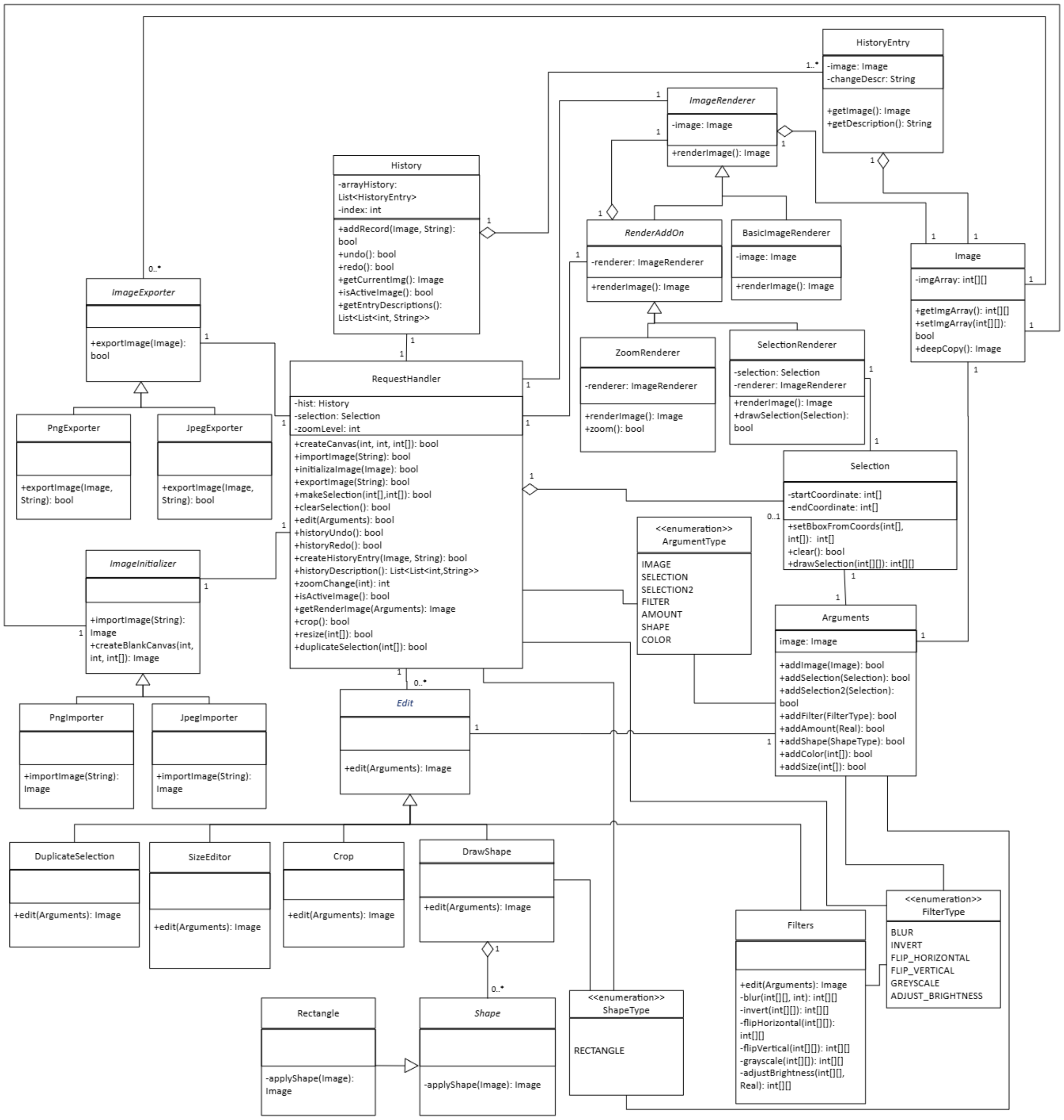
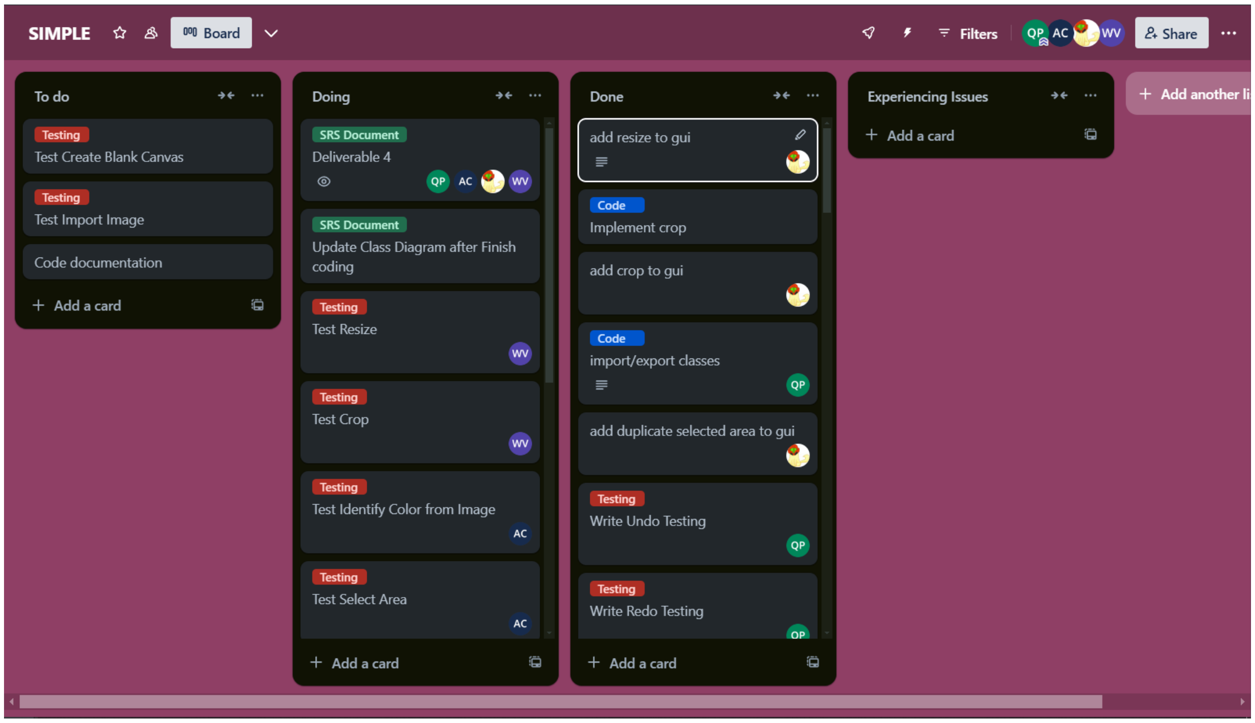


Figure 19: Updated SIMPLE Class Diagram

4.5.3 Development

4.5.3.1 Description

We have completed implementing the code for the classes according to the class diagram in Figure 19. Implementation for and updates to these classes can now be found in our GitHub repository. Our Kanban board has also been updated following our progress



To do
- Testing — Test Create Blank Canvas
- Testing — Test Import Image
- Code documentation
- Add a card

Doing
- SRS Document — Deliverable 4 — QP AC WV
- SRS Document — Update Class Diagram after Finish coding
- Testing — Test Resize — WV
- Testing — Test Crop — WV
- Testing — Test Identify Color from Image — AC
- Testing — Test Select Area — AC
- Add a card

Done
- add resize to gui
- Code — Implement crop
- add crop to gui
- Code — import/export classes — QP
- add duplicate selected area to gui
- Testing — Write Undo Testing — QP
- Testing — Write Redo Testing — QP
- Add a card

Experiencing Issues
- Add a card

Figure 20: Updated SIMPLE Kanban Board