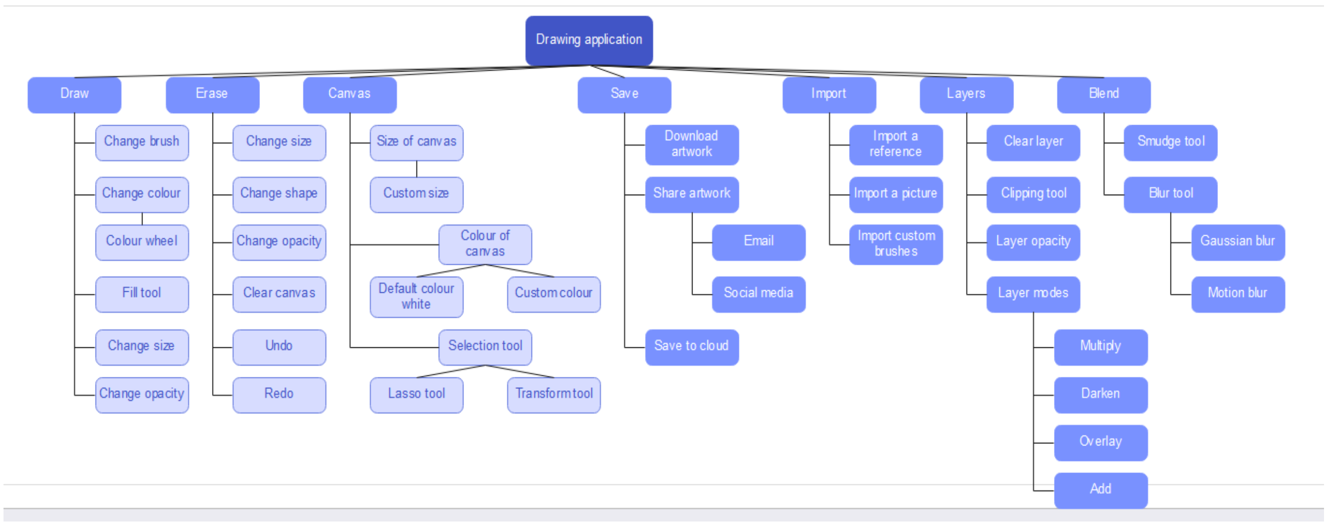
Project: Drawing Application

Project Description:

The project I have chosen is a drawing application similar to MS paint, procreate or ibis paint. It is an application that allows users to create artworks and save or share it. The application allows users to draw, paint, erase, fill, blend etc. The work breakdown structure (WBS) below shows the various features the application offers to users.

Work breakdown structure:



Function modules:

Draw: This module focuses on the functions related to drawing on the canvas like selecting brush, changing brush size, changing colours and changing the opacity of the brush.

Erase: This module classifies the functions related to erasing the strokes on the canvas. These functions mainly include changing shape and size of the eraser tool. Other functions of the module are changing opacity, undo, redo and clear canvas.

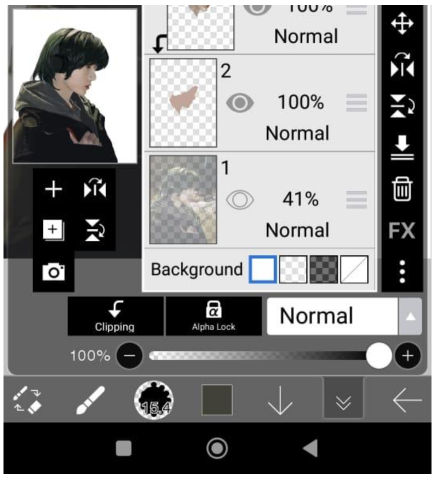
Canvas: This module contains all the functions related to the canvas like changing its dimensions, the default colour of the canvas, changing its colour and the selection tool. The selection can be further broken down into two subtasks – the lasso tool and the transform tool. The lasso tool can be used to select an area of the canvas as the user wishes and the transform tool can be used to transform the selected part of the artwork by changing the size or distorting it.

Save: This module comprises of all the functions related to saving and sharing the artwork including exporting it. The artwork can be saved to the device, cloud or be shared on social media or via email.

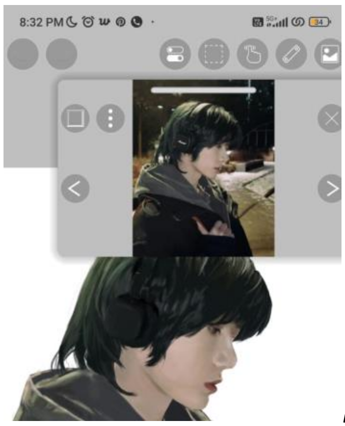
Import: This module deals with all the functions related to importing tools and pictures that will help with the artwork as per the user's wishes. One can import custom brushes, a reference or even a picture.

Layers: This is a module that is extremely helpful for digital artists as layers provide the ability to draw in layers, stacking one on top of the other. This feature is useful when a user wants to erase only certain parts of the drawing or be organized. The subtasks of this module include the clipping tool which allows users to clip the upper layer to the layer below it, changing layer opacity, clearing a layer and layer modes. The layer modes can be further divided into subtasks consisting of different layer types like multiply, add etc.

Example software (Ibis Paint X):



- *Layers and layer modes, clipping tool*

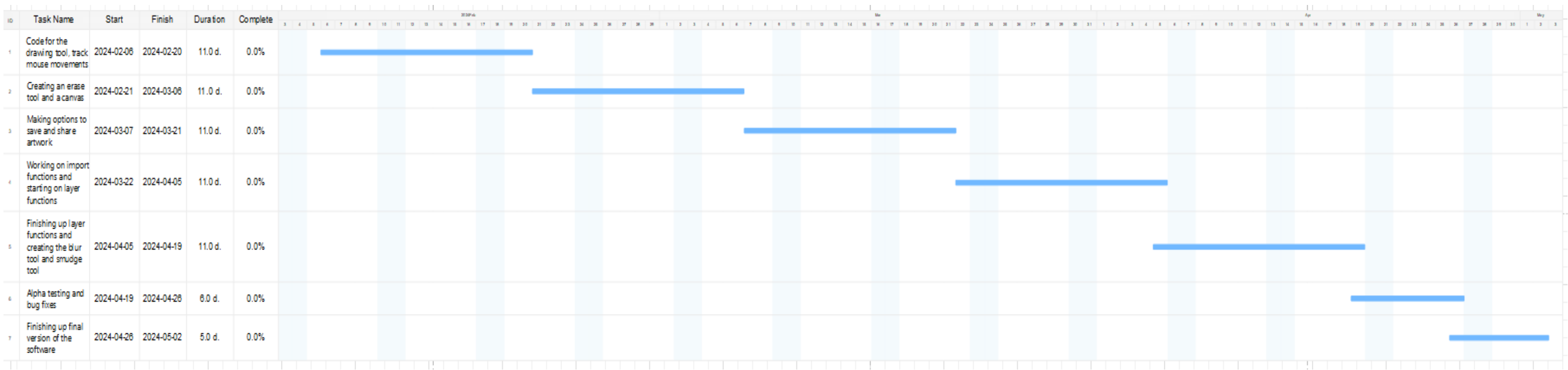


Reference window

Blend: This module focuses on the required tasks of blending and smudging which are useful for digital artists. The blur tool can be further divided into subtasks consisting of various blending modes like gaussian blur and motion blur.

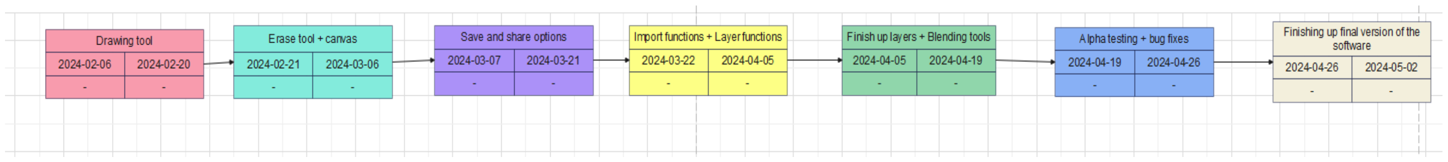
Expected deliverable and timeline using (Gant Chart and PERT chart):

Gantt chart:



ID	Task Name	Start	Finish	Duration	Complete
1	Code for the drawing tool, track mouse movements	2024-02-06	2024-02-20	11.0 d.	0.0%
2	Creating an erase tool and a canvas	2024-02-21	2024-03-06	11.0 d.	0.0%
3	Making options to save and share artwork	2024-03-07	2024-03-21	11.0 d.	0.0%
4	Working on import functions and starting on layer functions	2024-03-22	2024-04-05	11.0 d.	0.0%
5	Finishing up layer functions and creating the blur tool and smudge tool	2024-04-05	2024-04-19	11.0 d.	0.0%
6	Alpha testing and bug fixes	2024-04-19	2024-04-26	6.0 d.	0.0%
7	Finishing up final version of the software	2024-04-26	2024-05-02	5.0 d.	0.0%

Pert chart:



The functional and non-functional requirements:

Functional requirements:

- Draw on a canvas
- Erase lines on the canvas including undo and redo functions
- Clear the canvas.
- Save and share the artwork.
- Blend and smudge lines on the canvas.
- Should allow import of brushes, pictures and references.
- Should allow the customization of certain tools and colours.

Non-functional requirements:

Reliability:

- Adding a number of layers must not make the application lag or glitch.
- Application must not crash if the artwork is too complex.
- Save the artwork even if user hasn't explicitly asked to save the artwork before exiting.

Usability:

- Has to have a user-friendly interface with easy-to-understand icons.
- The tools and available functions should not clutter the user interface.
- The canvas should be clear and there has to be zoom in and zoom out options.

Portability and Flexibility:

- The application should work the same way on a phone or a tablet.
- The application should be compatible with different styluses including a mouse or even the user's finger on the phone.

Scalability:

- The application should allow the user to increase the number of layers without causing the application to lag or crash.
- The application should allow the user to stack brush strokes on top of one another without causing any kind of lag.
- The application should allow the user to use big canvas sizes without any issues like lagging, crashing etc.

Modifiability:

- The application should be easy to modify in case of new user requirements or requests.

Constraints:

- The canvas size can't be set to something extremely large which would result in overloading the application. A maximum and minimum limit to the dimensions can help solve this problem.
- The number of layers a user can stack on top of one another can't be too many as it might make the application lag or crash. A limit to the number of layers has to be specified.

Choosing the Process model required for my project:

My project requirements are subject to change as the drawing application may require new features according to the customer requirements. The customers may also want some modifications and changes based on their experience with the software. Hence using an incremental model for this project would be helpful.

In the **incremental model**, at any time, the plan is made just for the next increment and not for any kind of long-term plan. Therefore, it is easier to modify the version as per the need of the customer.

The core features are developed first. Here, the core features are to draw and erase. Once the core features are fully developed, then these are refined to increase levels of capabilities by adding new functions in Successive versions. As each successive version of the software is constructed and delivered, now the feedback of the Customer is to be taken and these are then incorporated into the next version. Each version of the software has more additional features than the previous ones.

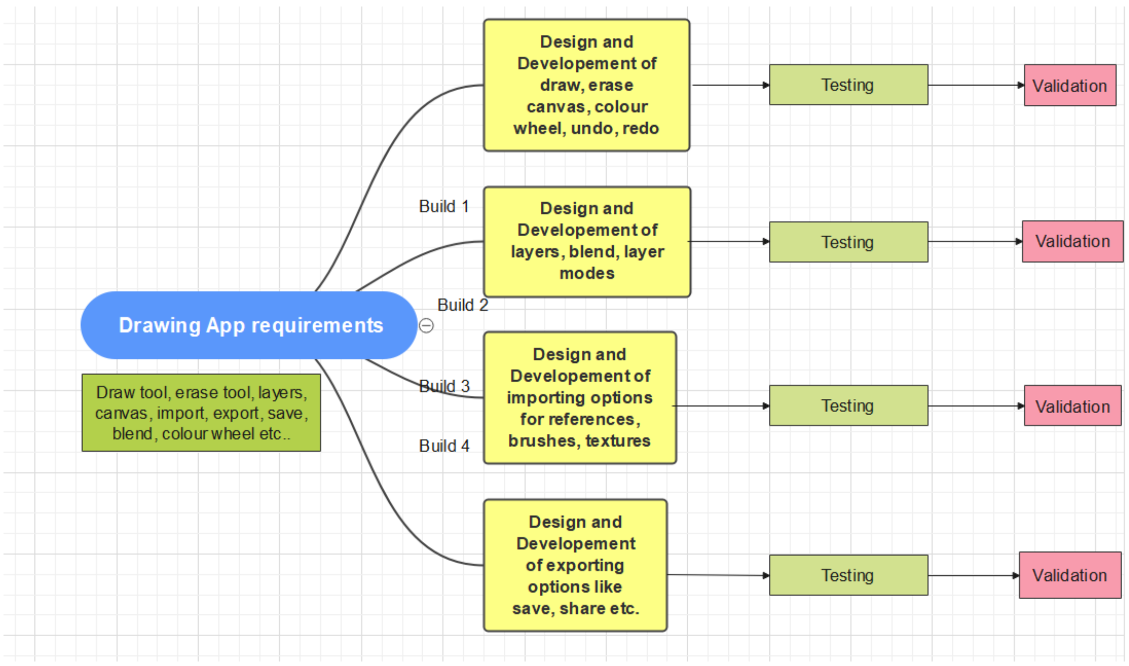
If we decide to use a **waterfall model**, we can have a systematic plan for the software. But for this the customer has to give clear requirements before the plan is drafted. If the requirements are not subject to huge changes or additions, we can use the waterfall model for the software. There is detailed documentation in each step which is useful for the customer's legal reasons. The waterfall model will ensure that everything will proceed in a systematic manner according to the plan.

We could also use the **agile model** for this project. The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance. The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements. Intensive testing and building prototypes give the customer something to build on while giving requirements for changes.

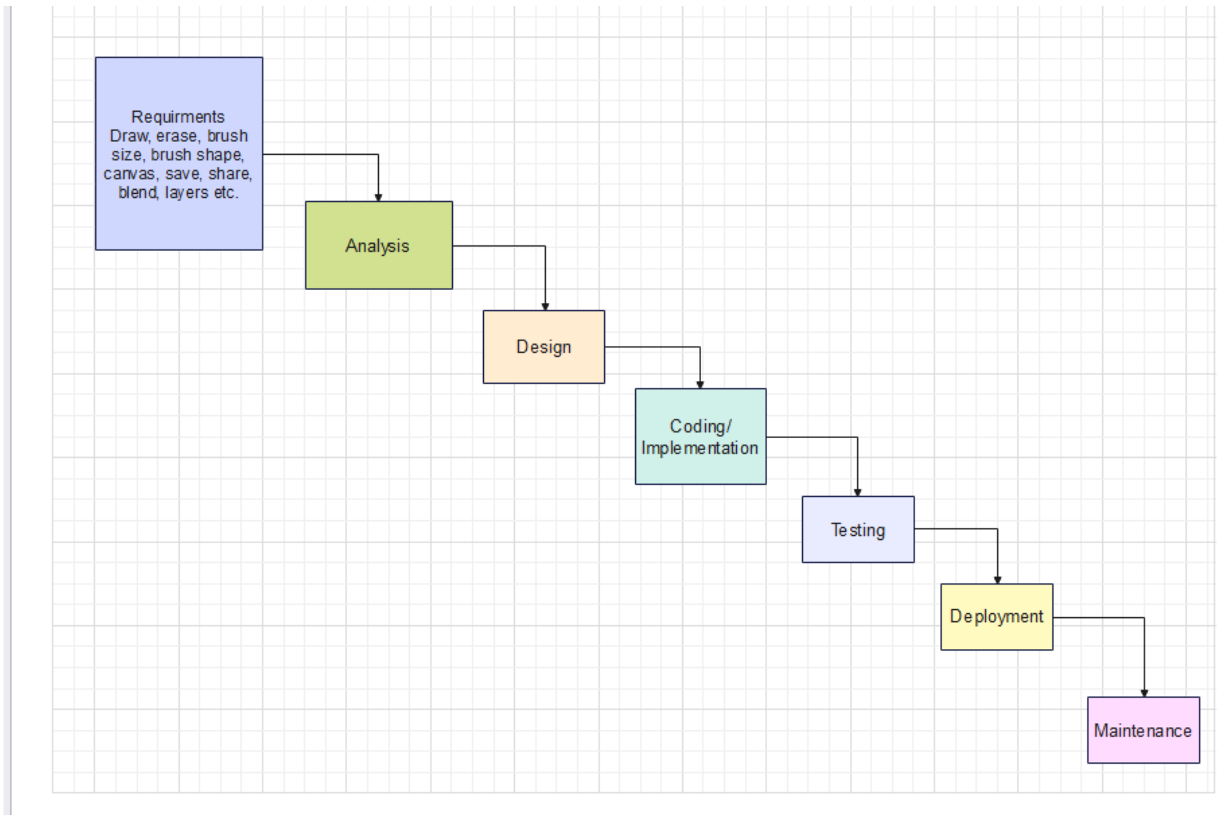
Hence, we can use a mixture of incremental model and waterfall model. The waterfall model can be used for the individual increments to maximize efficiency. Or we could also use an agile model if the task needs to be ready at a sooner date.

Diagrams for process models:

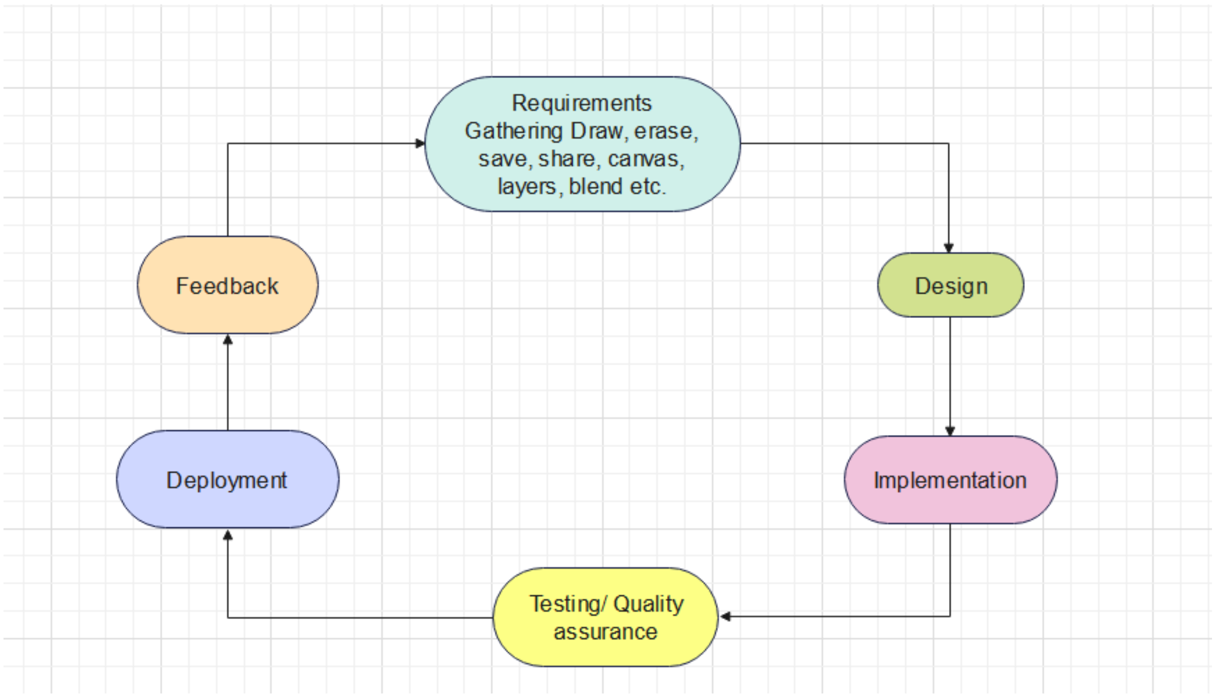
Incremental:


Drawing App requirements
Draw tool, erase tool, layers, canvas, import, export, save, blend, colour wheel etc..
Build 1
Build 2
Build 3
Build 4
Design and Developement of draw, erase canvas, colour wheel, undo, redo
Design and Developement of layers, blend, layer modes
Design and Developement of importing options for references, brushes, textures
Design and Developement of exporting options like save, share etc.
Testing
Testing
Testing
Testing
Validation
Validation
Validation
Validation

Waterfall:


Requirments Draw, erase, brush size, brush shape, canvas, save, share, blend, layers etc.
Analysis
Design
Coding/ Implementation
Testing
Deployment
Maintenance

Agile:



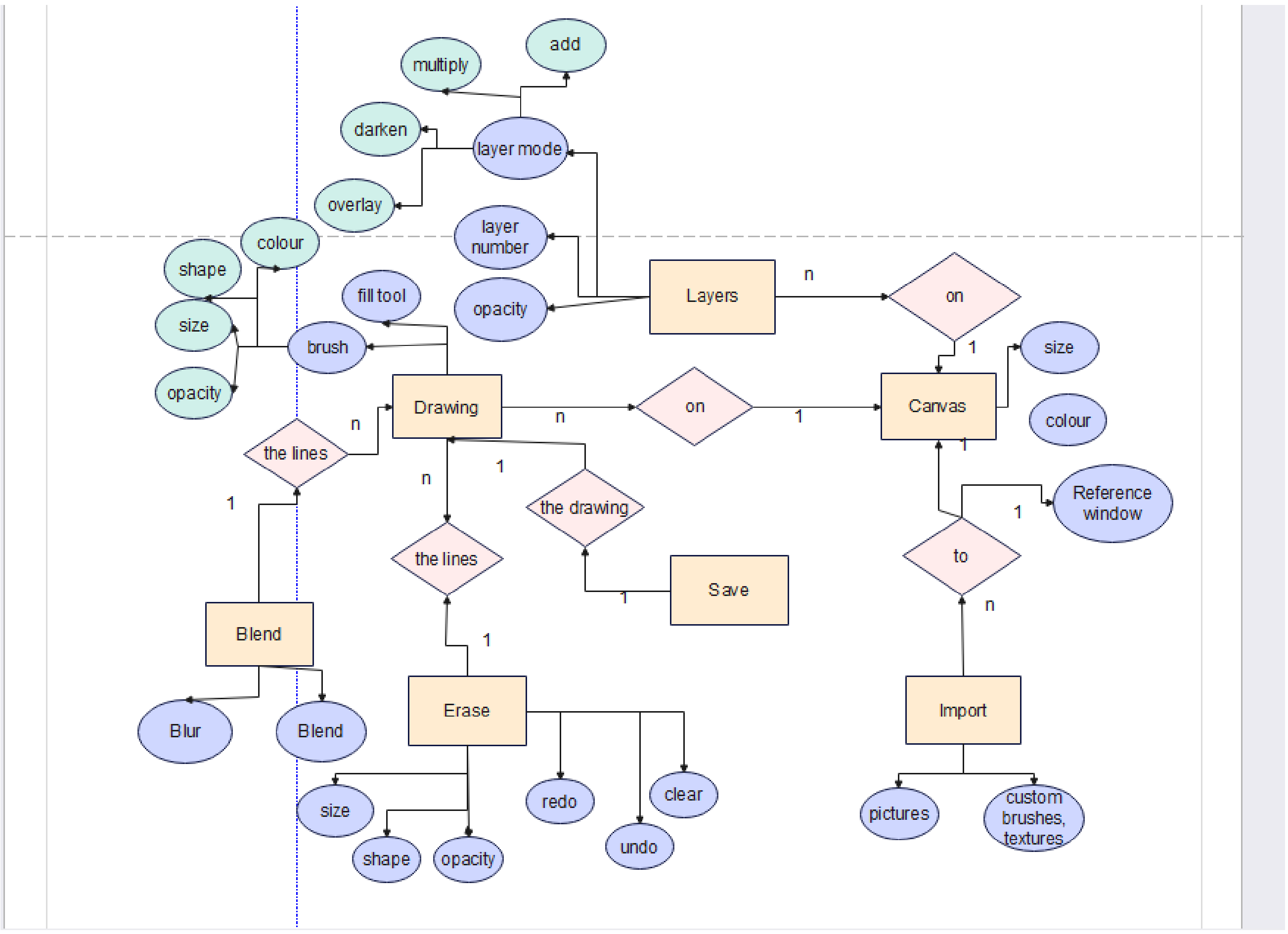
Requirements Gathering Draw, erase, save, share, canvas, layers, blend etc.
Design
Implementation
Testing/ Quality assurance
Deployment
Feedback

ER Diagram:

Entities: Drawing, Blend, Erase, Save, Import, Canvas, Layers

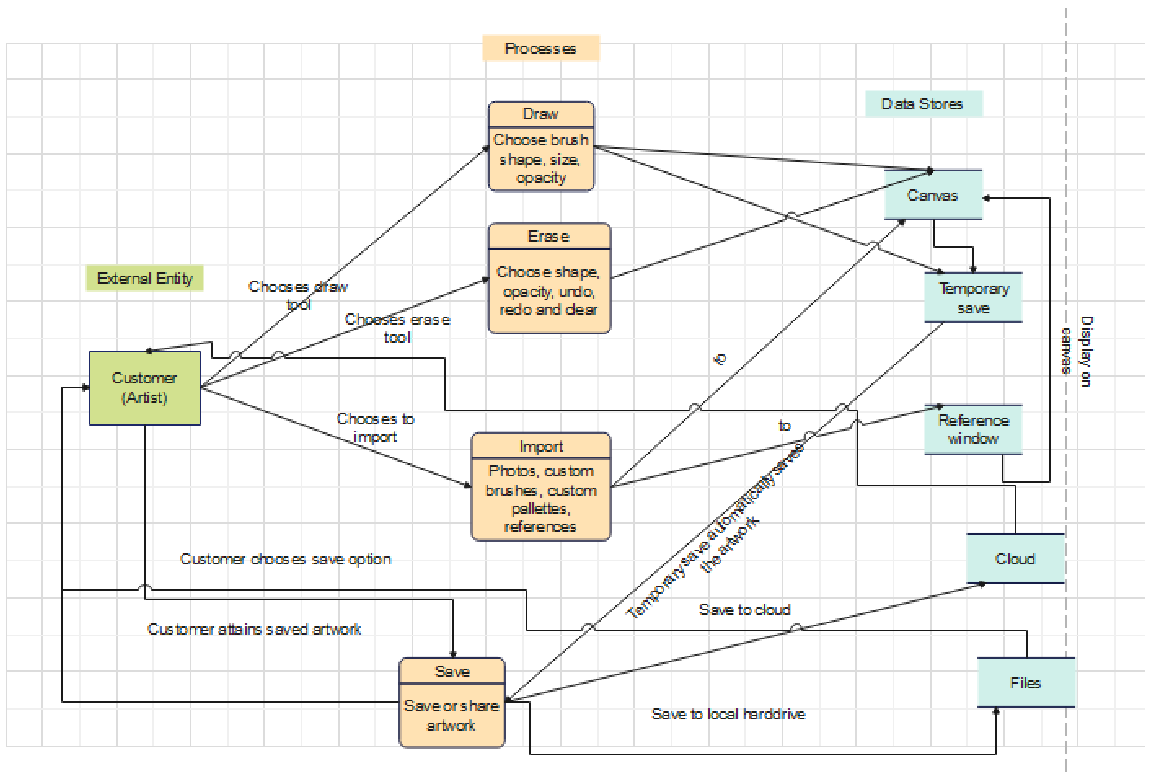
Relations: Erasing *the lines* of the drawing (1 to many), Blending *the lines* of the drawing (1 to many), Save *the* drawing (one to one), Import to canvas (many to one), Layers *on* canvas (many to one), Drawing *on* canvas (many to one),

Attributes are represented by the circles connected to the entities or relations.


add
multiply
darken
layer mode
overlay
layer number
colour
shape
fill tool
opacity
Layers
n
on
size
brush
1
size
opacity
Drawing
n
on
1
Canvas
colour
n
the lines
1
n
1
the drawing
Reference window
1
to
the lines
Save
1
n
Blend
1
Blur
Blend
Erase
Import
size
redo
clear
pictures
custom brushes, textures
shape
opacity
undo

Data flow diagram:

The data flow diagram for my project, drawing application is given below.



The **main external entity** here is the **Artist or the user.**

<u>The main processes are:</u> Draw (choose brush, brush shape, brush size and opacity. Draw lines on the canvas with the chosen brush settings), erase (Erase tool shape, size and opacity selection, undo and redo strokes and clear canvas), Import (Import photos, references, custom brushes, palettes etc. to the console and canvas), save (Save or share the artwork to the desired location.)

<u>The data stores are:</u>

Canvas - The drawn strokes are stored on a stack here.

Temporary save – The drawing is incrementally stored here, also on a stack.

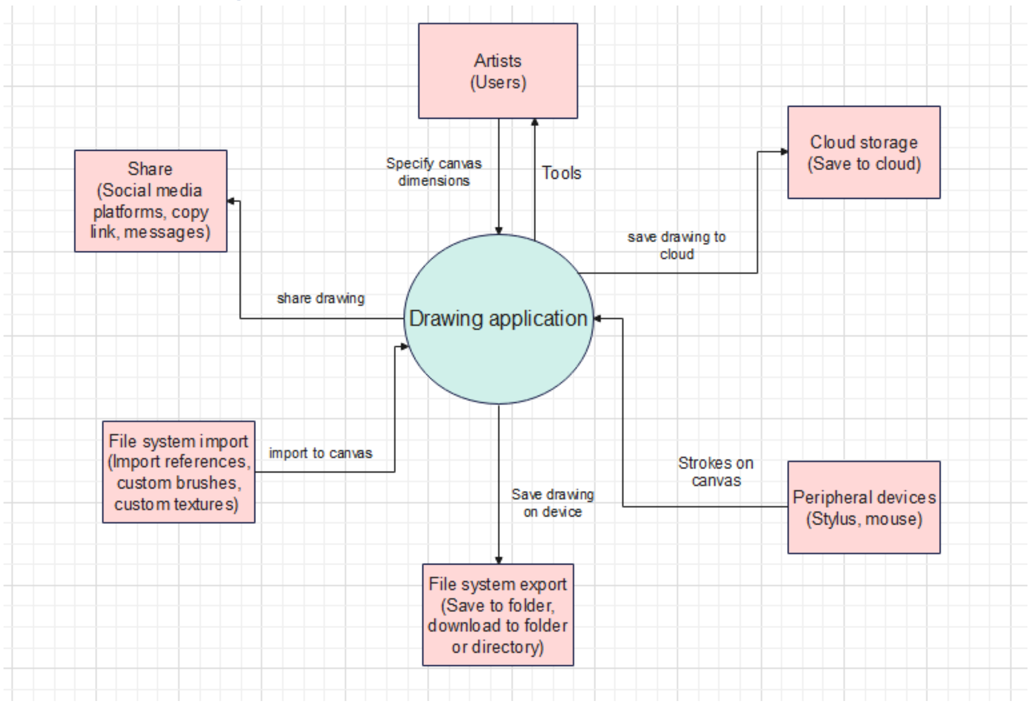
Reference window – The reference photos are stored and displayed here.

Cloud – The drawing is saved on a cloud.

Files – The drawing can be saved on the user's local storage device in a directory.

The paths followed by the data is represented in the diagram above.

Context flow diagram:



Artists (Users)
Specify canvas dimensions
Tools
Share (Social media platforms, copy link, messages)
Cloud storage (Save to cloud)
save drawing to cloud
share drawing
Drawing application
File system import (Import references, custom brushes, custom textures)
import to canvas
Save drawing on device
Strokes on canvas
Peripheral devices (Stylus, mouse)
File system export (Save to folder, download to folder or directory)

The context flow diagram for the drawing application is given above. The diagram shows the various external entities and how they interact with the application.

The external entities here are:

The Artist/ user: Specify canvas dimensions, select the tools from the console, draw with the selected brush tool on the canvas, erase strokes with the selected erase tool, add and delete layers, choose layer modes, blend strokes on canvas, decide to save or share the artwork etc.

Peripheral devices: These include the platform and devices the artist is using to draw, i.e., a stylus or a mouse or even just the user's finger. These are used to make strokes on the canvas.

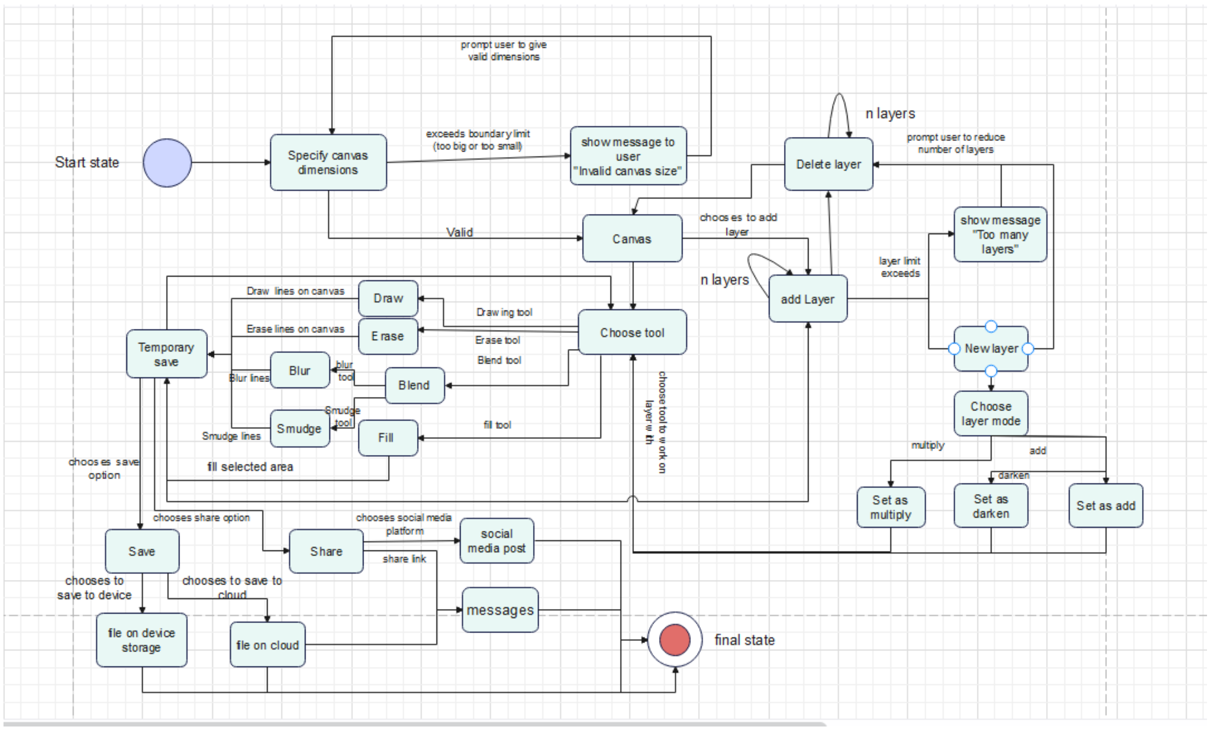
File system import: Import references, custom brushes, custom brushes or custom textures from the user's local storage devices.

File system export: Export the drawing to save it as a file on the user's local device storage file directory.

Cloud store: Export the drawing to the application-maintained cloud or any other cloud application.

Share: Share artwork to social media or copy link or even share to direct messages.

State transition diagram:

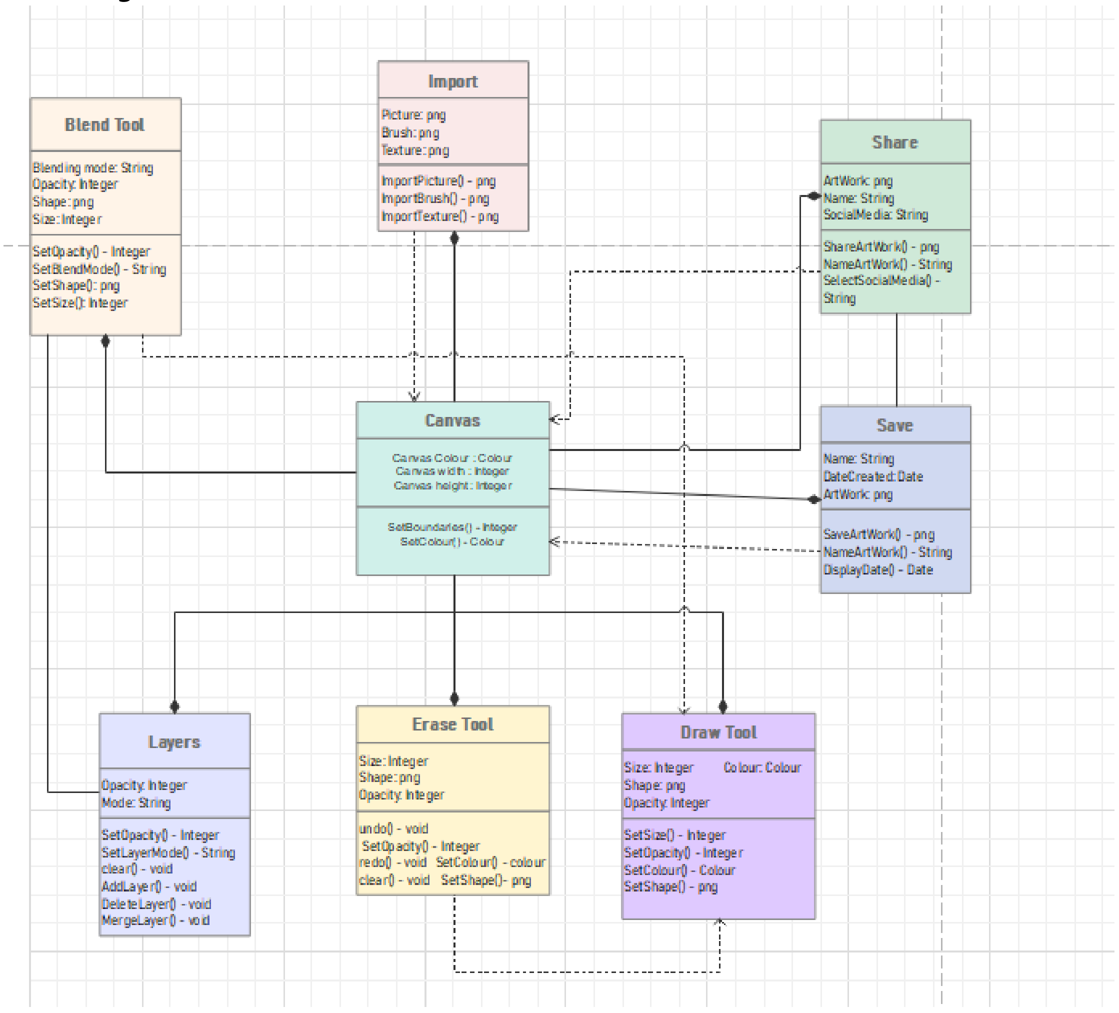


Above is the state transition diagram for the drawing application.

- The start state starts with asking the user to specify the canvas dimensions. If the dimensions exceed the boundaries, then prompt the user to specify valid size of canvas.
- After the canvas is created, the next states are choosing a tool or adding a layer.
- User chooses desired tool, i.e., brush, erase, blend, fill etc at the choose tool state.
- User draws, erases, blends or fills according to their desired drawing.
- The next state is a temporary save which uses a stack to keep track of the drawn strokes on the canvas.
- The user can choose to draw more on this temporary save or decide to save or share this as the final artwork.
- The user can save it as a file on their device storage or as a file on the cloud.
- The user can share the artwork as a social media post or as a link in direct messages.
- The other state you can go to from the canvas state is choosing to add a layer. If the number of layers exceeds the limit, prompt the user to use lesser layers.
- The user can delete layers. User can delete up to n layers which will result in the canvas.
- One can keep adding layers up to n layers. Each layer enters a new layer state.
- The next state is to choose the layer mode.
- This state will go to the choose tool state so the user can make strokes on the layers.
- This will go to the temporary save as well to be saved or shared according to the user's preferences.

<u>Object Oriented Design</u>

a. Class diagram:



Here, the classes are:

Canvas, Layers, erase tool, draw tool, blend tool, save tool, share tool, import

The attributes for the classes are:

Canvas: Canvas colour, canvas width (Integer), canvas height (Integer)

Layers: Opacity (Integer), Mode (String)

Erase Tool: Size (Integer), Shape (png), Opacity (integer)

Draw Tool: Size (integer), Shape (png), Opacity (integer)

Import:

Share:

Save:

The operations are given as follows below the attributes.

The classes are connected as follows:

Canvas and Draw Tool, Erase Tool, Blend Tool, Save, Share and Layers:

Use composition to represent the relationship between Canvas and these classes as they are tightly integrated and cannot exist without the canvas. This implies that these classes are part of the canvas and are owned by it.

Blend Tool and Layers:

Use association to represent the relationship between Blend Tool and Layers as the blend tool interacts with layers but does not own or compose them.

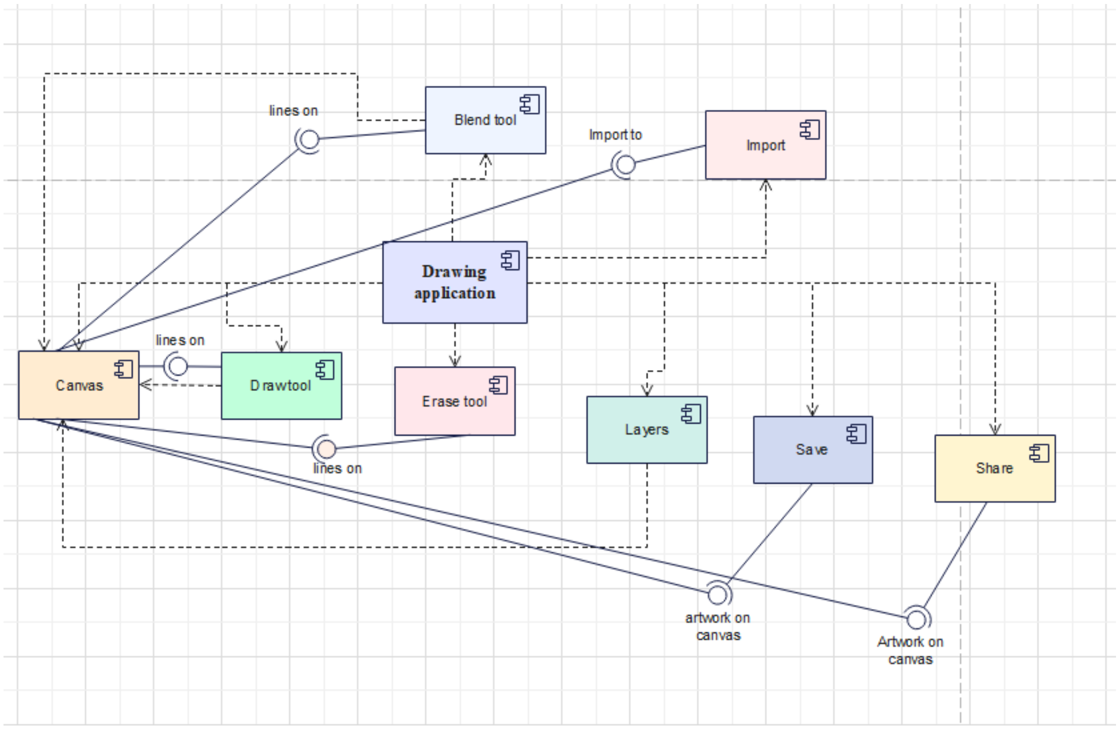
Save and Share:

Use association to represent the relationship between Save and Share as they are related but do not imply ownership or composition.

Draw Tool, Erase Tool, Blend Tool, Save, Share:

Use dependency to represent any dependencies these classes have on other classes.

b. Component Diagram:

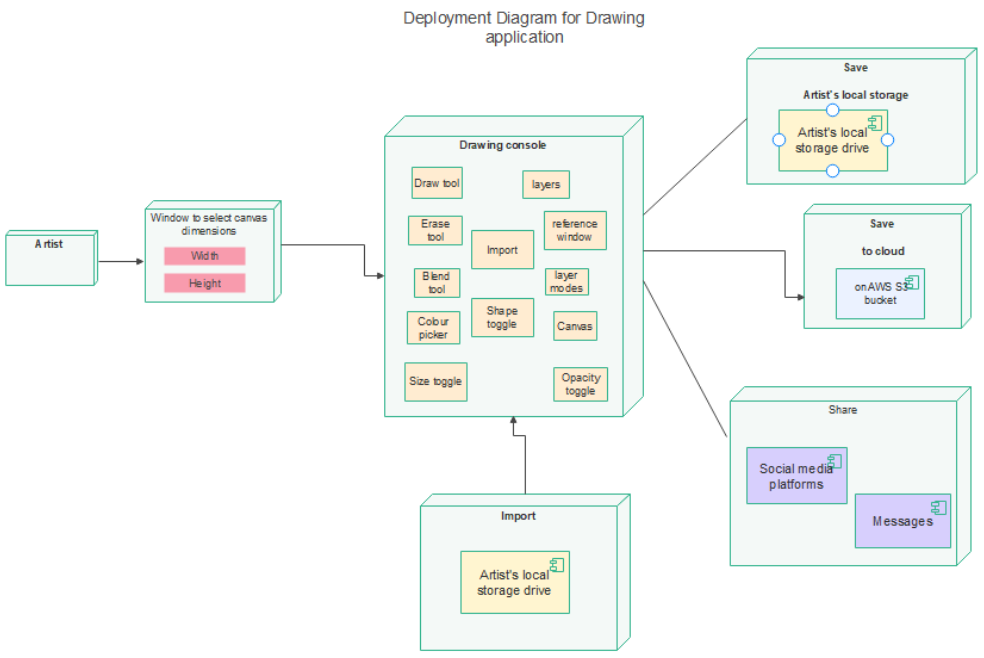


The components are:
Canvas, draw tool, erase tool, layers, save, share, blend tool and import.

- All the tools are dependent on the canvas as they can't exist otherwise.
- The artwork to be saved or shared is provided by the canvas and the interface required is the save and share components.

- The tools, i.e., draw, erase and blend are the provided interfaces that require the interface canvas.
- The import component acts as a provided interface to import the picture to the required interface canvas.

c. *Deployment Diagram:*



Here, the nodes are:

Artist, the window to select canvas dimensions, drawing console, import, save to local storage, save to cloud and share.

The components for the nodes are:

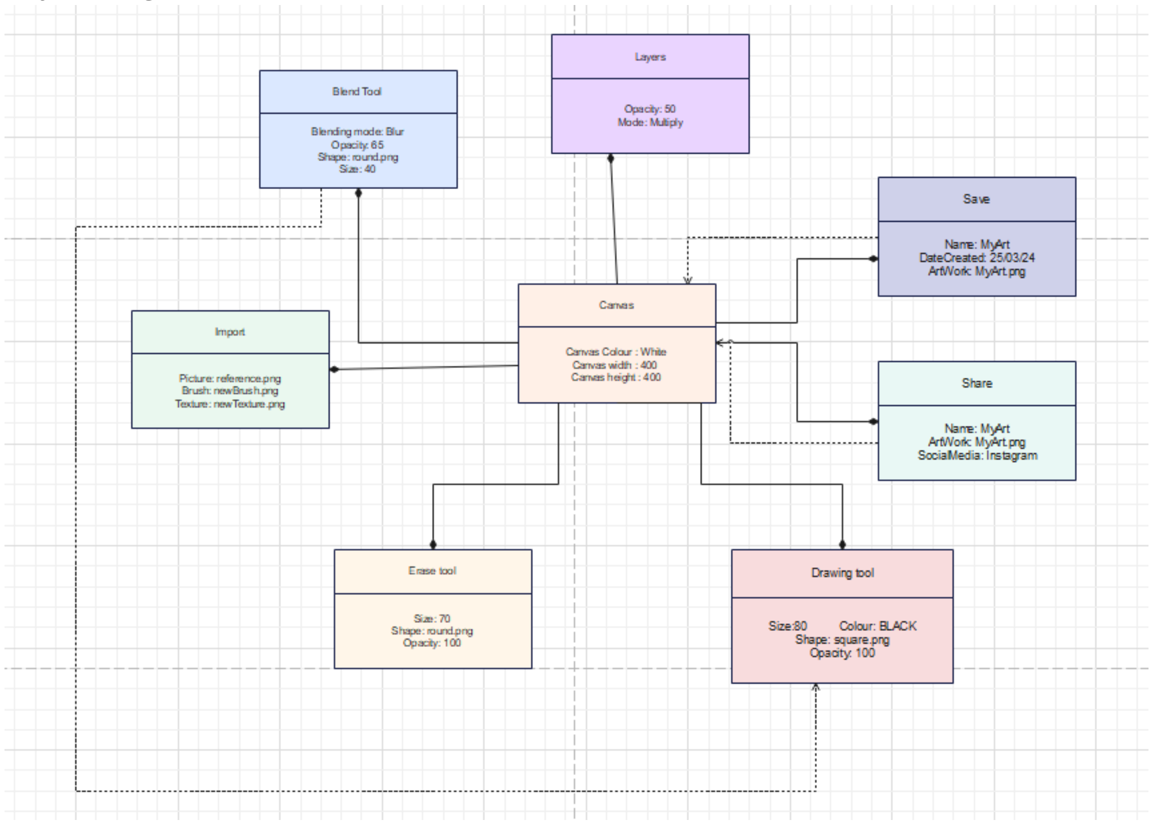
Import: Artist's local storage device.

Share: Social media platforms and messages.

Save: Artist's local storage device and to the cloud – An AWS S3 bucket maintained by the application developers.

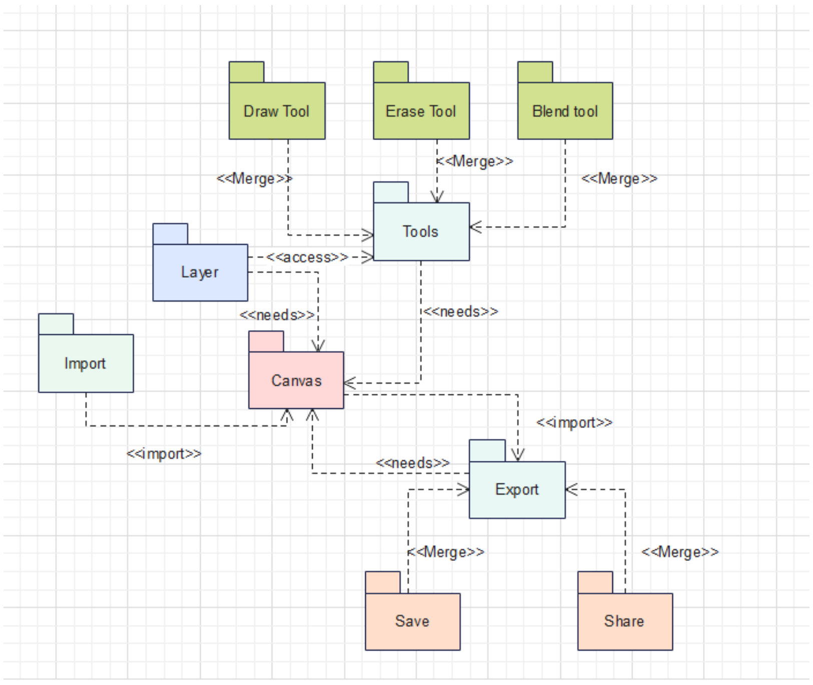
The user is first prompted with a window asking the dimensions for the canvas. If the dimensions are valid, a canvas is created and displayed on the console along with all the tools. After drawing a desired artwork, the artist can save the artwork to their local storage device or to the cloud which is an AWS S3 bucket maintained by the developers. Or the artist can also choose to share the artwork online on their desired social media platforms or through private messages.

d. Object Diagram:



- Here the objects are canvas, import, drawing tool, erase tool, blend tool, save, share.
- Canvas is created based on the dimensions specified by the user, here it is 400 x 400.
- The user can draw with the drawing tool on the canvas and erase the lines made using the erase tool. Here the opacity and size are given.
- The user can import a picture, for example a file called reference.png or newTexture.png.
- The user can name and save the artwork. For example, here the artwork is named as MyArt.png and saved to the local storage. The date it was created is also displayed.
- The user can also share the file MyArt.png to their desired social media platform or messages.

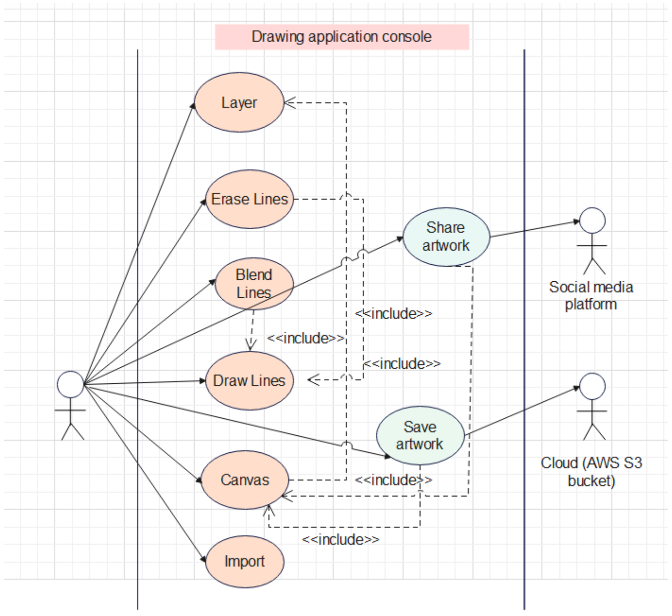
e. *Package Diagram:*



- Here the packages draw tool, erase tool, blend tool which are merged into a package called tools.
- The packages save and share are merged into a package called export.
- The packages tools, export and layer are dependent on the package canvas as they can't exist without the canvas.
- Layers can access the tools without having to import the entire tools package.
- Import package can import the canvas package.

Behavioural Diagrams

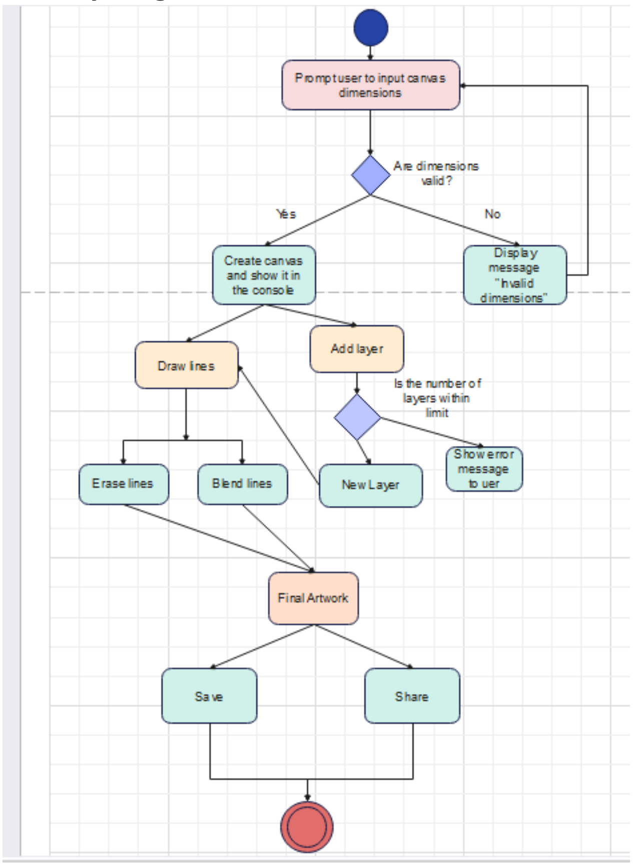
f. Use case diagram:



Here,

- The actors are the users or the artist
- The use cases are bounded by the system “Drawing Application console”.
- The uses cases are erase lines, blend lines, draw lines, canvas, import, save and share.
- The include relationships between the use cases are represented by <<include>> over the dotted lines. The Include Relationship indicates that a use case includes the functionality of another use case. For example, Blending lines includes drawing lines.

g. Activity Diagram:

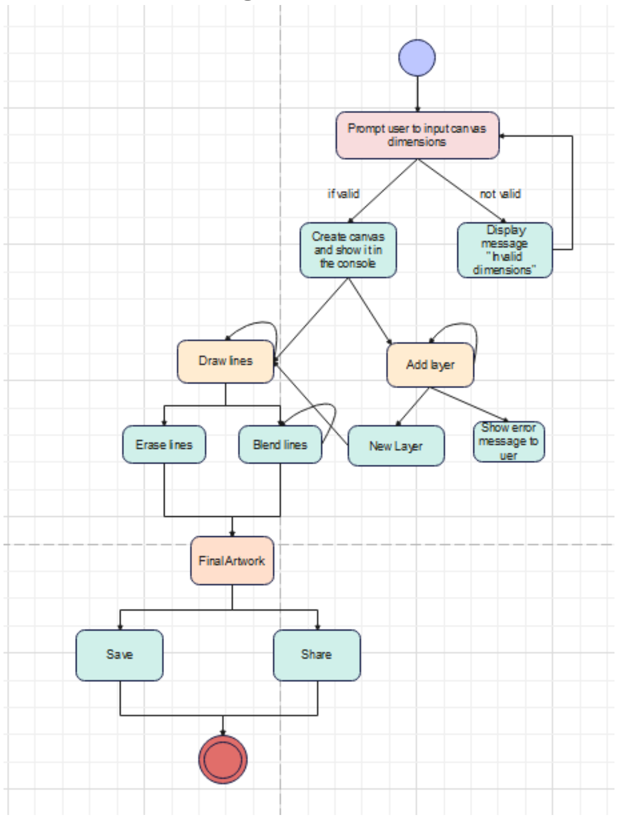


Prompt user to input canvas dimensions
Are dimensions valid?
Yes
No
Create canvas and show it in the console
Display message "Invalid dimensions"
Draw lines
Add layer
Is the number of layers within limit
Erase lines
Blend lines
New Layer
Show error message to uer
Final Artwork
Save
Share

- The initial state is represented as the coloured circle. The first activity is to prompt the user to define canvas dimensions.
- If the dimensions are valid it goes to the next state that is to create a canvas and display it on the console. If the dimensions were not valid then the user is prompted to give valid dimensions.
- The next activity is to draw lines on the canvas or add a layer to the canvas.
- A new layer is created only if it’s within the limit of number of layers, otherwise it shows an error and prompts user to reduce the number of layers.
- The next activity is to blend lines on the canvas or erase them. This part is represented using a fork to represent that the activities can be done simultaneously.

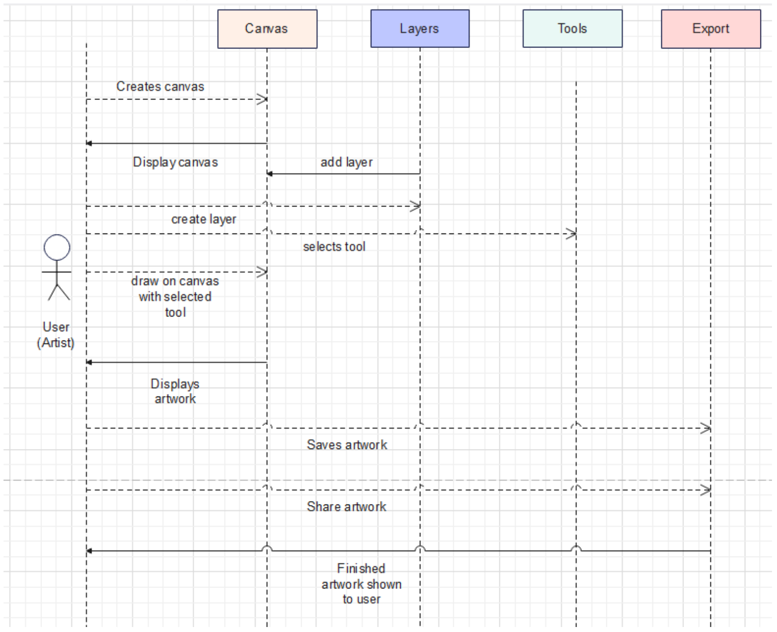
- These activities join to form the final artwork which can then be saved or shared which then goes to the final state which is represented by a coloured circle in another circle.

h. State machine diagram:


Prompt user to input canvas dimensions
if valid
not valid
Create canvas and show it in the console
Display message "Invalid dimensions"
Draw lines
Add layer
Erase lines
Blend lines
New Layer
Show error message to uer
Final Artwork
Save
Share

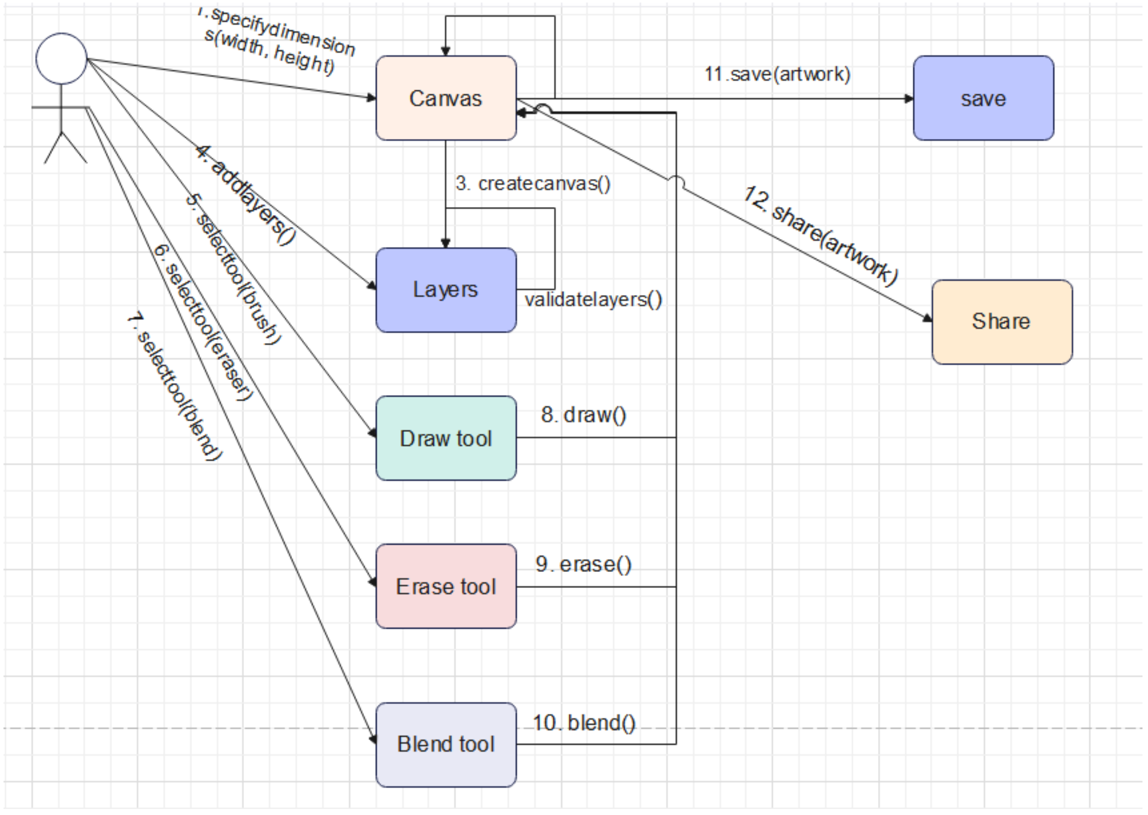
- The initial state is represented as the coloured circle. The first state is to prompt the user to define canvas dimensions.
- If the dimensions are valid it goes to the next state that is to create a canvas and display it on the console. If the dimensions were not valid then the user is prompted to give valid dimensions.
- The next state is to draw lines on the canvas or add a layer to the canvas. These states can be repeated hence there is a self-loop. The layers can be stacked on top of each other until it exceeds the limit.
- A new layer is created only if it's within the limit of number of layers, otherwise it shows an error and prompts user to reduce the number of layers.
- The next activity is to blend lines on the canvas or erase them. This part is represented using a fork to represent that the activities can be done simultaneously. The blend lines state has a self-loop as blended lines can be blended more.
- These states join to form the final artwork which can then be saved or shared which then goes to the final state which is represented by a coloured circle in another circle.

i. Sequence diagram:



- Here the actor is the user/ artist. The lifelines are canvas, tools, export, layers.
- The reply messages are represented using an arrow and the other messages are represented using a dotted arrow.
- The above diagram shows the interactions between the user and the various lifelines in the drawing application.

j. Communication diagram:



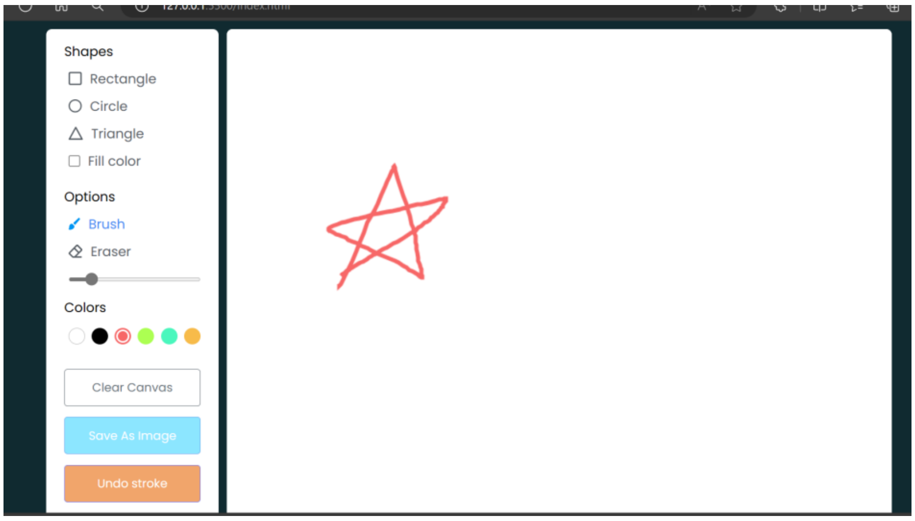
- Objects are represented as rectangles with their names inside.
- Messages between objects are represented by arrows indicating the direction of communication and the message name.
- Each message represents a method call or action performed by one object/component on another object/component.
- The User object sends a specifyDimensions(width, height) message to the Canvas object to specify the dimensions for the new canvas.
- The Canvas object validates the specified dimensions by sending a validateDimensions(width, height) message to itself.
- If the dimensions are valid, the Canvas object proceeds to create the canvas by sending a createCanvas() message to the Layers object.
- User draws on the canvas using the selected tool (draw() message from Draw Tool to Canvas).
- User saves the canvas (saveCanvas() message from Canvas to Save).
- User shares the canvas (shareCanvas() message from Canvas to Share).

Implementation using HTML, CSS and Java Script:

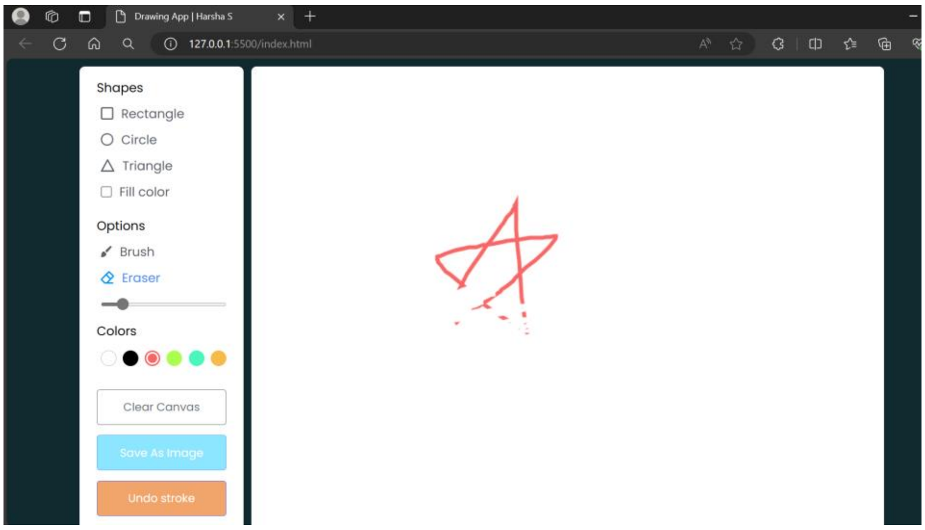
- The features implemented in this first build of the software are the draw tool, erase tool, colour picker, clear canvas, draw shapes, undo and save image.
- There is also a size bar implemented so the user can change the size of the tool.
- An array was used to store snapshots of the drawing after every stroke. When undo function is called, the latest stroke from the array is removed. When the canvas clear function is called, all the snapshots in the array are cleared.
- EventListners were used to make the draw and erase functions.
- The index.html file contains all the text, buttons and icons displayed on the console.
- Pickr library was used in the JavaScript file to implement the colour picker.
- Testing was done after completion and verified as shown below. Newer features can be added incrementally to this build after user validation.

Testing all the features and verifying functionality:

Testing draw tool:

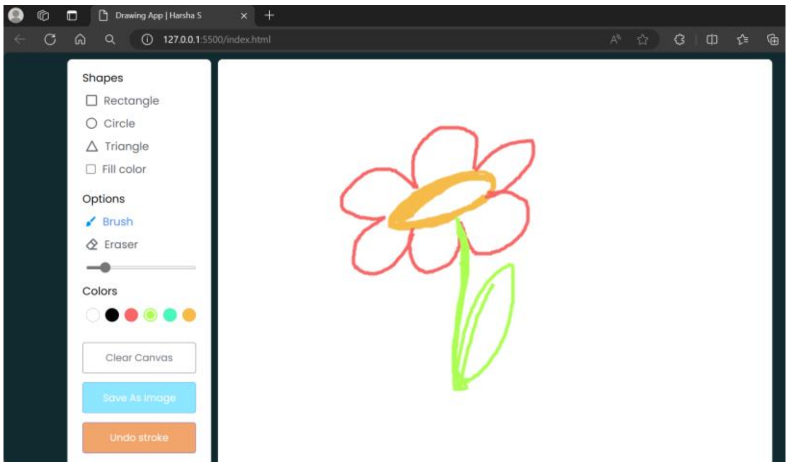


Testing erase tool:

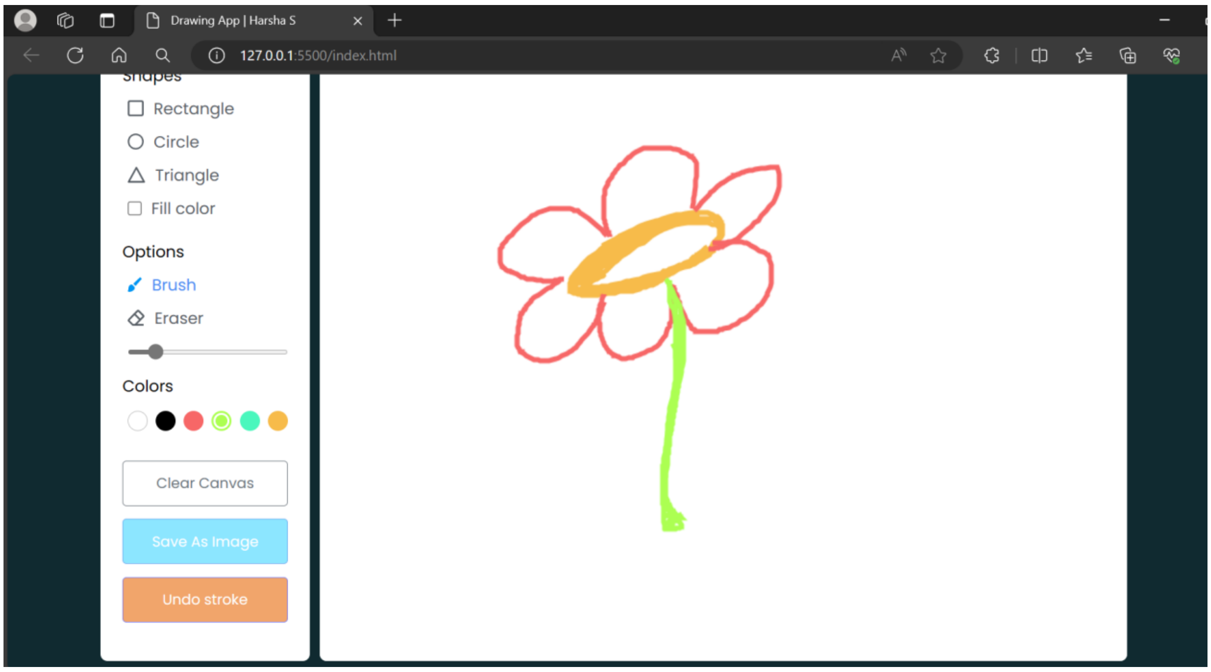


Testing undo:

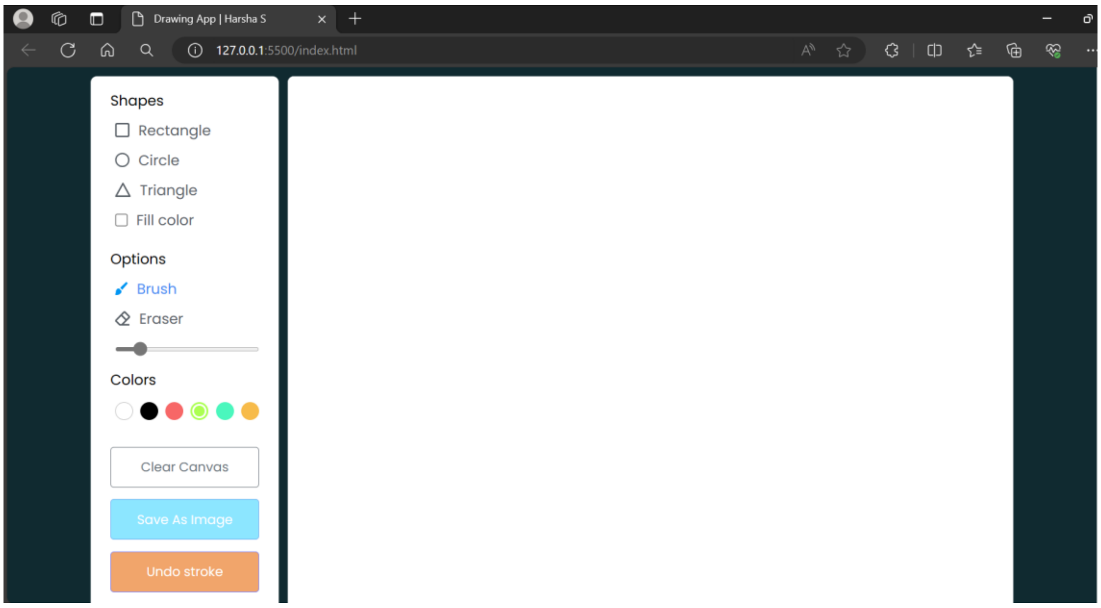
Before:



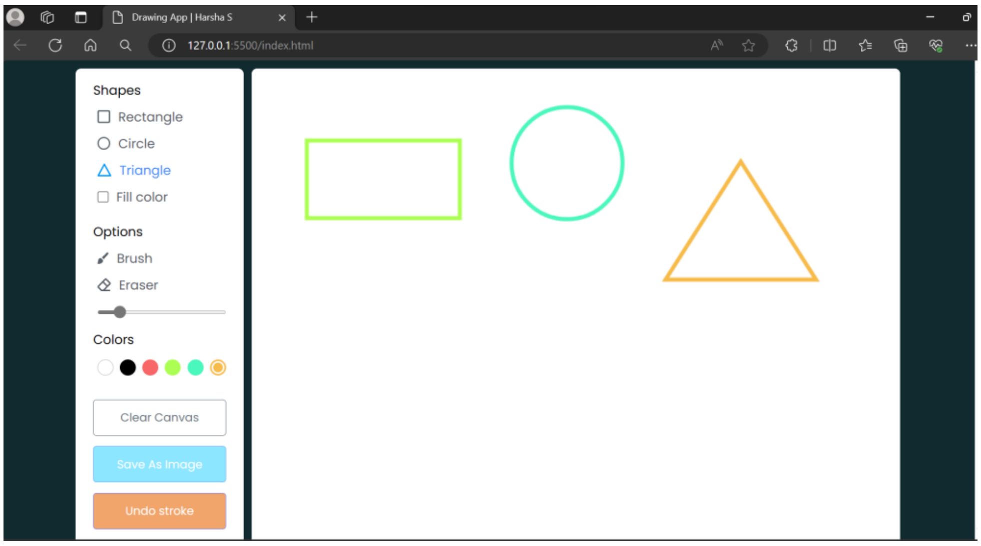
After:



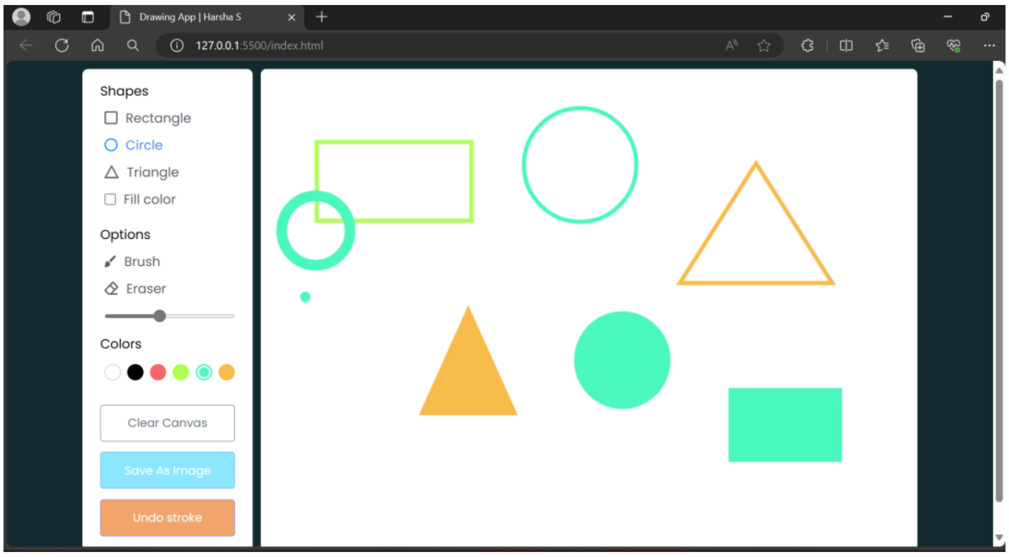
Testing clear canvas:



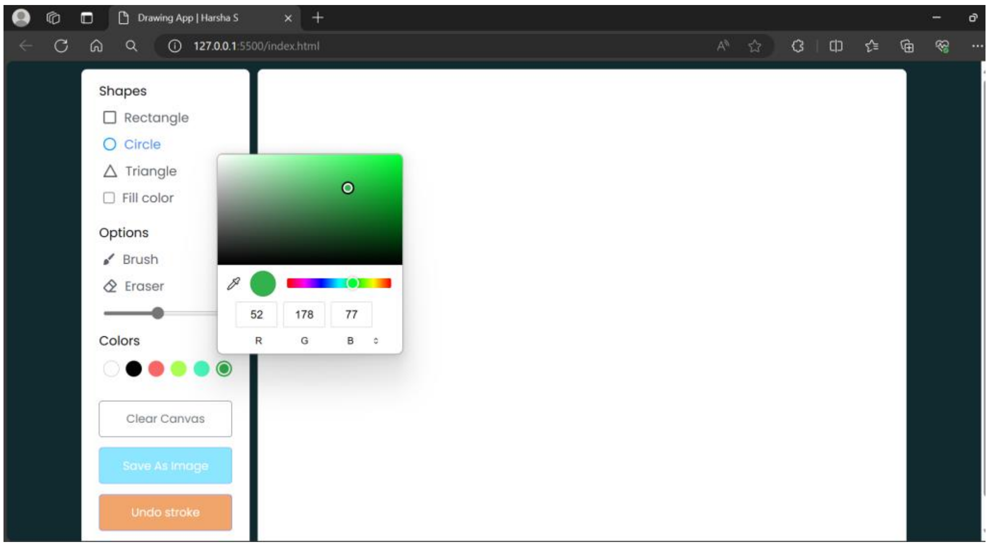
Testing shapes:

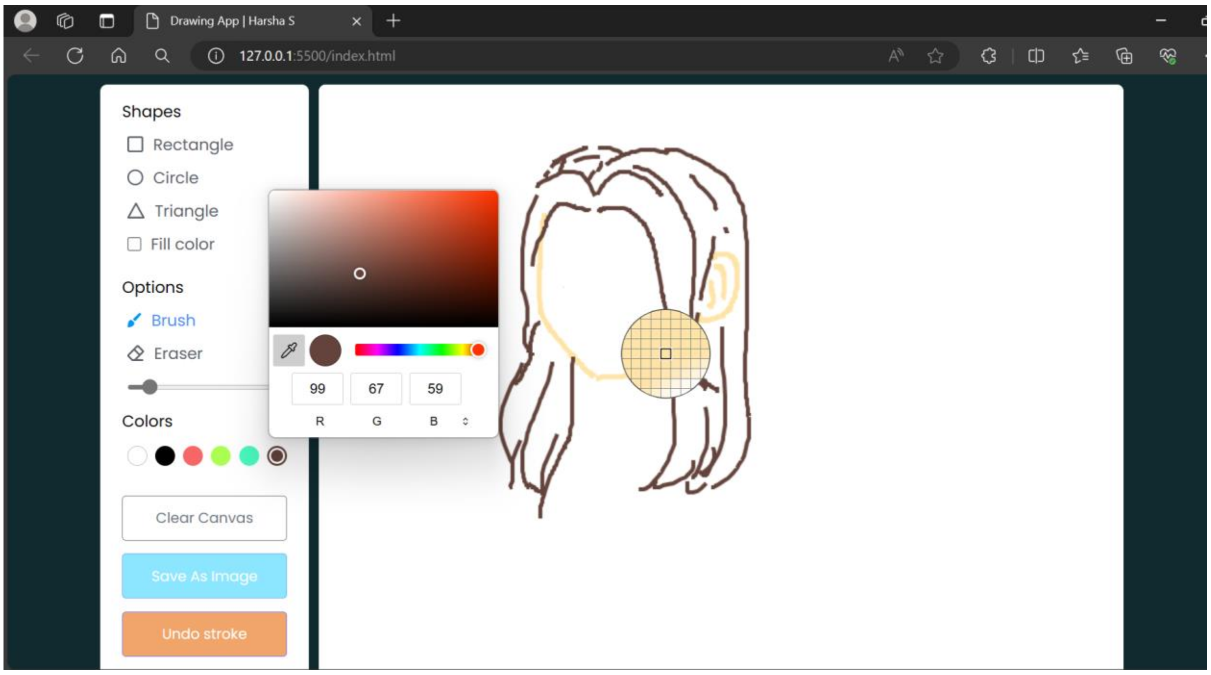


Testing fill:

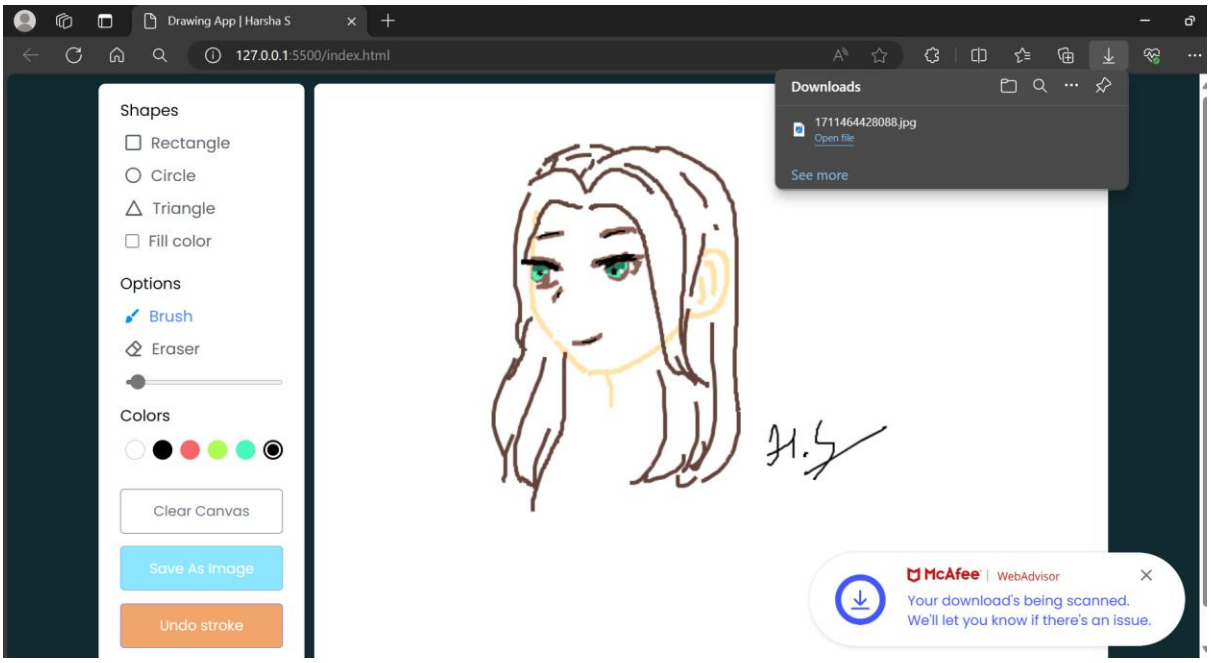


Test for Colour picker:





Testing save:



Conclusion:

Based on all the diagrams and planning that have been done, we have arrived at the first build of the software. This build is going to be given to the user for validation and testing. Based on their input and whether they require any more features or wish to change certain requirements, we can issue a new build that can be incremented on the initial build. This is possible because our project uses the incremental model. Based on the testing results we can say that this project meets the functional requirements of the users. Other artists who have tried out the application have enjoyed using the software and liked how easy to use it was. The features to save the artwork allows them to save their beautiful artworks to their local storage device's download folder. Overall, the customer requirements have been met and customer's are satisfied.

Code files have been committed to my GitHub account:

https://github.com/harsha-likes-to-code/Drawing-app