**BAHRIA UNIVERSITY KARACHI CAMPUS DEPARTMENT OF ARTIFICIAL INTELLIGENCE**

# 

**PROJECT REPORT**

**COURSE NAME: PROGRAMMING FOR ARTIFICIAL INTELLIGENCE**

**COURSE CODE: AIL-201**

**Group Members:**

**MAIMOONA SABIR  
(02-136212-047)**

**MUHAMMAD ANAS**

**(02-136212-033)**

**TEAM PROFILE**

|  |  |
| --- | --- |
| MAIMOONA SABIR | COLLAGE MAKER, VIDEOEDITOR |
| MUHAMMAD ANAS | IMAGE EDITING, DOCUMENTATION |

Contents

[1](#_Toc138156303)

[Abstract 5](#_Toc138156304)

[CHAPTER 1: COMPREHENSIVE INSTRUCTION AND PROBLEM STATEMENT 5](#_Toc138156305)

[1.1 Introduction 5](#_Toc138156306)

[1.2 Objective 5](#_Toc138156307)

[1.3 Goals 5](#_Toc138156308)

[1.4 Scope 6](#_Toc138156309)

[1.5 Problem Statement 6](#_Toc138156310)

[Chapter 2: Literature Review 6](#_Toc138156311)

[2.1 Photo Editing Applications 2.2.1 Adobe Photoshop: 6](#_Toc138156312)

[2.2.2 Canva: 6](#_Toc138156313)

[2.2.3 Snapseed: 6](#_Toc138156314)

[2.3 Video Editing Applications 2.3.1 Adobe Premiere Pro: Adobe 7](#_Toc138156315)

[2.3.2 iMovie: 7](#_Toc138156316)

[2.3.3 Filmora: 7](#_Toc138156317)

[2.4 Collage-Making Applications 2.4.1 Canva: 7](#_Toc138156318)

[2.4.2 Adobe Spark: 7](#_Toc138156319)

[2.4.3 Pic Collage: 7](#_Toc138156320)

[2.5 Summary 7](#_Toc138156321)

[Chapter 3: Methodology 7](#_Toc138156322)

[3.2 Use Cases 7](#_Toc138156323)

[3.2.1 Photo Editing Use Case: 8](#_Toc138156324)

[3.2.2 Video Editing Use Case: 8](#_Toc138156325)

[3.2.3 Collage-Making Use Case: 8](#_Toc138156326)

[Chapter 04 : Code Snippet and Analytical/Comparative Discussion 8](#_Toc138156327)

[DISCUSSION 8](#_Toc138156328)

[SOURCE CODE 9](#_Toc138156329)

[SCREENSHOT 67](#_Toc138156330)

[Chapter 5: Conclusion and Future Enhancements 70](#_Toc138156331)

[User Interface Enhancements: 70](#_Toc138156332)

[Advanced Filter Options: 70](#_Toc138156333)

[Video Editing Features: 71](#_Toc138156334)

[Export and Saving: 71](#_Toc138156335)

[Video Effects and Animations: 71](#_Toc138156336)

[Timeline and Keyframe Editing: 71](#_Toc138156337)

[Batch Processing and Automation: 71](#_Toc138156338)

[Integration with Online Services: 71](#_Toc138156339)

[Chapter 06 : References 71](#_Toc138156340)

# Abstract

Pixlr is a versatile and user-friendly application that provides a comprehensive set of tools for editing photos, creating videos, and making collages. It offers a range of features and functionalities that cater to both beginners and more advanced users.When it comes to photo editing, Pixlr offers a wide array of tools and effects to enhance and modify images. Users can adjust brightness, contrast, and saturation, as well as apply filters and creative overlays to add unique styles and moods to their photos. The application also includes features for cropping, resizing, and rotating images, allowing users to customize their compositions.

In addition to photo editing, Pixlr enables users to create and edit videos. Users can import their video clips into the application and apply various effects, transitions, and filters to enhance the visual quality. They can also trim, cut, and rearrange clips to create a cohesive narrative or storyline. Pixlr supports different video formats and resolutions, making it suitable for a range of projects. Another notable feature of Pixlr is its collage-making capabilities. Users can select multiple photos and arrange them in a grid or custom layout to create visually appealing collages. The application provides templates, backgrounds, and stickers to add creative elements to the collages, allowing users to personalize their compositions. Users can also adjust the spacing, borders, and sizes of the photos to achieve the desired arrangement.

Pixlr's user interface is designed to be intuitive and accessible. It offers a range of tools and features that can be easily accessed and utilized through a simple menu system. Whether users are new to photo editing or have experience with other editing applications, Pixlr aims to provide a smooth and user-friendly experience.

# CHAPTER 1: COMPREHENSIVE INSTRUCTION AND PROBLEM STATEMENT

* 1. Introduction

The purpose of this chapter is to provide a comprehensive instruction and problem statement for the Pixlr application. This chapter will outline the main objectives, goals, and scope of the project, as well as introduce the problem that the application aims to solve. Additionally, it will provide an overview of the structure and organization of the subsequent chapters.

* 1. Objective

The objective of the Pixlr application is to provide users with a versatile and user-friendly platform for editing photos, creating videos, and making collages. The application aims to offer a wide range of tools and features that cater to both beginners and more advanced users, allowing them to enhance and customize their visual content with ease.

## 1.3 Goals

The primary goals of the Pixlr application are as follows: 1.3.1 Provide a comprehensive set of tools for photo editing, video editing, and collage-making. 1.3.2 Ensure a user-friendly and intuitive interface that is accessible to users of all skill levels. 1.3.3 Offer a diverse range of effects, filters, and creative overlays to enhance and modify photos. 1.3.4 Support various video formats and resolutions, allowing users to edit and create videos. 1.3.5 Provide templates, backgrounds, and stickers for creating visually appealing collages. 1.3.6 Allow users to customize compositions by adjusting spacing, borders, and sizes of photos.

## 1.4 Scope

The scope of the Pixlr application includes the following aspects: 1.4.1 Photo Editing: Users can modify and enhance photos by adjusting brightness, contrast, saturation, applying filters, and adding creative overlays. 1.4.2 Video Editing: Users can import video clips, apply effects, transitions, and filters, trim and rearrange clips, and export the edited videos. 1.4.3 Collage Making: Users can select multiple photos, arrange them in grids or custom layouts, and add templates, backgrounds, stickers, and adjust spacing and sizes.

## 1.5 Problem Statement

The problem that the Pixlr application addresses is the need for a comprehensive and user-friendly platform for editing photos, creating videos, and making collages. Many existing applications offer limited functionality or are too complex for beginners. Pixlr aims to bridge this gap by providing a versatile and accessible solution that caters to users of all skill levels.

# Chapter 2: Literature Review

This chapter provides a comprehensive review of the existing literature and related applications in the field of photo and video editing. The purpose of this review is to gain insights into the current state of the art, identify the strengths and limitations of existing solutions, and inform the development of the Pixlr application.

2.1 Photo Editing Applications 2.2.1 Adobe Photoshop: Adobe Photoshop is one of the most widely used photo editing applications, known for its extensive set of tools and advanced features. It offers a wide range of editing options, including adjustments, filters, and layer-based editing. However, it is considered a professional-level software and may be too complex for beginners.

2.2.2 Canva: Canva is a popular online graphic design tool that also provides basic photo editing functionalities. It offers a user-friendly interface and a variety of templates and design elements, making it suitable for beginners. However, its editing capabilities are limited compared to more specialized photo editing applications.

2.2.3 Snapseed: Snapseed is a mobile photo editing application developed by Google. It offers a wide range of editing tools, including filters, adjustments, and selective editing. Snapseed provides a user-friendly interface and is known for its advanced features, such as selective control and non-destructive editing.

2.3 Video Editing Applications 2.3.1 Adobe Premiere Pro: Adobe Premiere Pro is a professional video editing software widely used in the film and television industry. It offers advanced editing tools, effects, and transitions, along with support for various video formats and resolutions. However, its complex interface and steep learning curve make it less accessible for beginners.

2.3.2 iMovie: iMovie is a video editing application developed by Apple for macOS and iOS devices. It provides a user-friendly interface and a range of editing features, including trimming, transitions, and effects. iMovie is popular among beginners and casual video editors due to its simplicity and integration with Apple's ecosystem.

2.3.3 Filmora: Filmora is a video editing software that caters to both beginners and intermediate users. It offers a wide range of editing tools, effects, and transitions, along with a user-friendly interface. Filmora provides a simplified editing experience without compromising on essential features.

2.4 Collage-Making Applications 2.4.1 Canva: Canva also offers collage-making functionalities, allowing users to create visually appealing compositions using templates, backgrounds, and stickers. It provides an intuitive drag-and-drop interface and a wide variety of design elements, making it popular among beginners.

2.4.2 Adobe Spark: Adobe Spark is a web-based tool that allows users to create collages, videos, and web pages. It offers a range of templates and customization options for collage-making. Adobe Spark provides a user-friendly interface and seamless integration with other Adobe Creative Cloud applications.

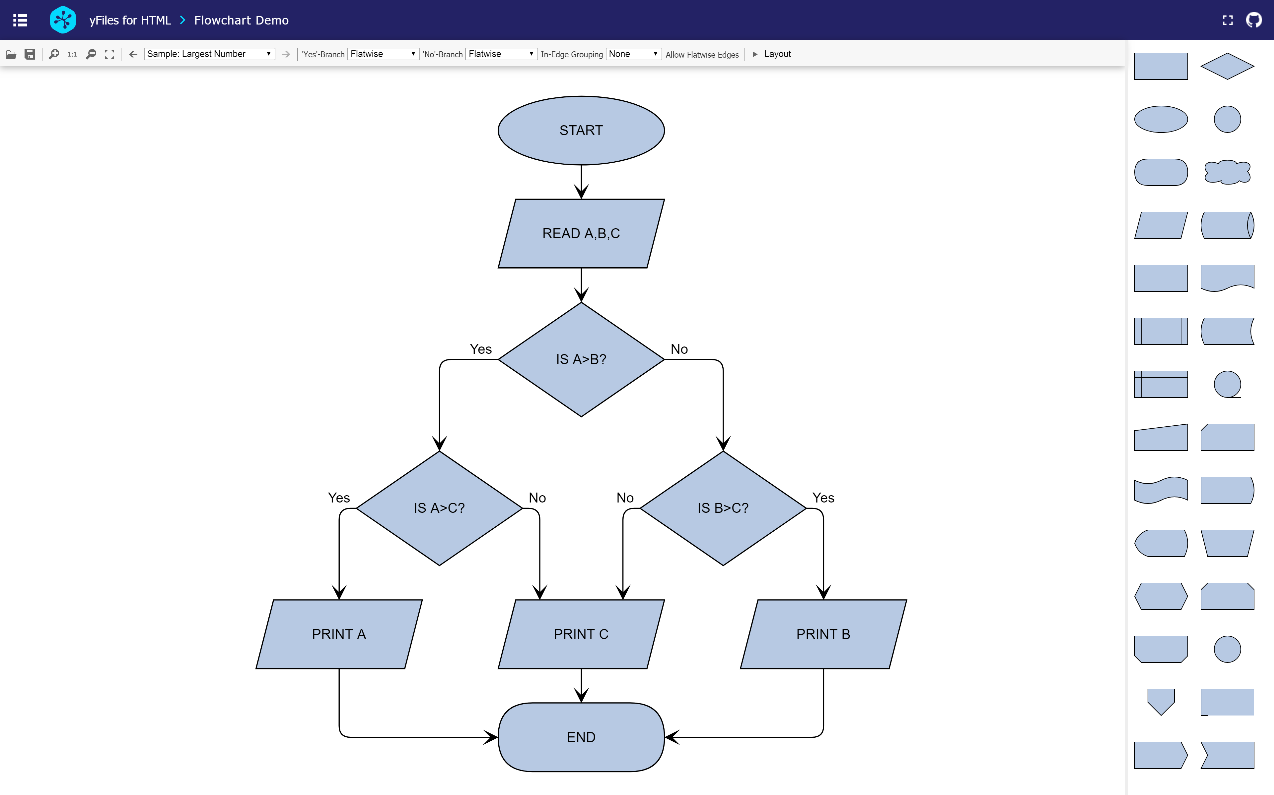
2.4.3 Pic Collage: Pic Collage is a mobile application that specializes in collage-making. It offers a wide variety of templates, stickers, and backgrounds, along with basic editing tools. Pic Collage provides a simple and intuitive interface, making it suitable for users of all skill levels.

## 2.5 Summary

This chapter presented a comprehensive review of existing literature and applications in the fields of photo editing, video editing, and collage-making. It highlighted the strengths and limitations of popular applications such as Adobe Photoshop, Canva, Snapseed, Adobe Premiere Pro, iMovie, Filmora, Canva, Adobe Spark, and Pic Collage. The insights gained from this review will inform the development and design considerations of the Pixlr application in subsequent chapters.

# Chapter 3: Methodology

This chapter outlines the methodology employed in the development of the Pixlr application. It describes the use cases for various actions within the application and presents flowcharts to illustrate the workflow and interactions between different components of the application.



3.2 Use Cases Use cases provide a detailed description of the interactions between users and the application, highlighting the specific actions and functionalities available. The following use cases are identified for the Pixlr application:

## 3.2.1 Photo Editing Use Case:

User selects a photo from their device or captures a new photo using the application's camera feature.

User applies various editing tools such as adjustments (brightness, contrast, saturation), filters, and creative overlays.

User can crop, resize, or rotate the photo to achieve the desired composition.

User saves the edited photo or shares it on social media platforms.

## 3.2.2 Video Editing Use Case:

User imports video clips from their device into the application.

User trims, cuts, and rearranges the clips to create a desired sequence.

User applies effects, transitions, and filters to enhance the visual quality of the video.

User can adjust the video's audio settings, including volume and background music.

User exports the edited video in their preferred format and resolution.

## 3.2.3 Collage-Making Use Case:

User selects multiple photos from their device or chooses from the application's library.

User arranges the photos in a grid or custom layout to create a collage.

User can adjust the spacing, borders, and sizes of the photos for a desired composition.

User adds templates, backgrounds, stickers, and text to enhance the visual appeal of the collage.

User saves the collage or shares it with others.

# Chapter 04 : Code Snippet and Analytical/Comparative Discussion

## DISCUSSION

The code you provided seems to be a editing application built using Tkinter library in Python. It includes features like opening and playing videos, applying filters to the video frames, adding music to the video, undoing and redoing filters, rotating the video, and controlling the playback speed.

However, there are a few issues and missing parts in the code that need to be addressed for the application to work correctly. Here are the main points:

There is a missing import statement for the tkinter module. Add the line import tkinter as tk at the beginning of the code.

The toggle\_playback() function is referenced in the code, but its implementation is missing. You need to define this function to handle the play/pause functionality.

The update\_button\_visibility() function is referenced but not defined. You should provide the implementation for this function to handle the visibility of the play/pause button.

The apply\_filter() function is used to apply the selected filter to the video frame, but its implementation is missing. You need to define this function and include the logic for applying the selected filter.

The add\_music() function is referenced but not defined. You should provide the implementation for this function to handle adding music to the video.

The restart\_video() function is referenced but not defined. You should provide the implementation for this function to handle restarting the video playback.

The undo\_filter() and redo\_filter() functions are referenced but not defined. You should provide the implementation for these functions to handle undoing and redoing the applied filters.

The rotate\_video() function is referenced but not defined. You should provide the implementation for this function to handle rotating the video.

The play\_video() function is defined but seems to be incomplete. You need to review and update the logic inside this function to properly play the video.

There are some missing parts related to GUI elements creation, such as creating the main window, video frame, options frame, and other buttons. Make sure to include the necessary code for creating these elements and arranging them properly in the application window.

## SOURCE CODE

from tkinter import \*

from PIL import ImageTk, Image, ImageFilter, ImageDraw

from tkinter import filedialog

from collections import deque

from PIL import ImageEnhance

import numpy as np

from tkinter import messagebox

import cv2

from PIL import ImageFont

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from tkinter import ttk

from PIL import Image, ImageTk

from tkinter import Tk, Label

from tkinter import HORIZONTAL

from tkinter import Label, Entry, Button

# Create the main window

mains = Tk()

mains.geometry("1200x800")

mains.attributes("-fullscreen", True) # Set fullscreen attribute

mains.title("Image Editor")

style = ttk.Style()

# Load the background image

bg\_image = ImageTk.PhotoImage(Image.open("C:\\Users\Lenovo\Downloads\\Picture2.jpg"))

# Create a label to display the background image

bg\_label = Label(mains, image=bg\_image)

bg\_label.place(x=0, y=0, relwidth=1, relheight=1)

# Function to exit fullscreen mode

def exit\_fullscreen(event):

mains.attributes("-fullscreen", False) # Set fullscreen attribute to False

mains.geometry("1200x800") # Restore the original window size

mains.bind("<Escape>", exit\_fullscreen) # Bind the Escape key to exit fullscreen mode

# Create a panel to display the image

panel = Label(mains, bg="black")

panel.grid(row=0, column=0, rowspan=12, padx=50, pady=50)

# Initialize image variables

img = None

output\_image = None

drawing\_image = Image.new("RGBA", (500, 500))

image\_stack = deque()

undo\_stack = deque()

drawing = False

last\_x, last\_y = 0, 0

def display\_image(image):

global disp\_image # Make disp\_image a global variable

disp\_image = ImageTk.PhotoImage(image)

panel.configure(image=disp\_image)

panel.image = disp\_image

def open\_image():

global img, output\_image, drawing\_image

# Open a file dialog to select an image file

file\_path = filedialog.askopenfilename(filetypes=[("Image Files", "\*.jpg;\*.jpeg;\*.png")])

if file\_path:

# Open the selected image file

img = Image.open(file\_path)

# Resize the image to (500, 500)

img = img.resize((500, 500))

# Create a copy of the original image

output\_image = img.copy()

# Create a new RGBA image for drawing

drawing\_image = Image.new("RGBA", output\_image.size)

# Append a copy of the image to the image stack

image\_stack.append(img.copy())

# Combine the output image and drawing image using alpha compositing

final\_image = Image.alpha\_composite(output\_image.convert("RGBA"), drawing\_image)

# Display the final image

display\_image(final\_image)

# Clear the undo stack when a new image is opened

undo\_stack.clear()

# Show a message box indicating successful image opening

messagebox.showinfo("Image Opened", "Image opened successfully!")

def adjust\_brightness(brightness\_pos):

# Convert brightness\_pos to a float value

brightness\_pos = float(brightness\_pos)

# Access the global output\_image variable

global output\_image

# Check if img exists (an image is selected)

if img:

# Convert the image pixels to a NumPy array

pixels = np.array(img)

# Adjust the brightness by multiplying each pixel with brightness\_pos

brightness\_adjusted = pixels \* brightness\_pos

# Clip the adjusted values to ensure they stay within the range of 0 to 255

brightness\_adjusted = np.clip(brightness\_adjusted, 0, 255).astype(np.uint8)

# Create a new image from the adjusted brightness array

output\_image = Image.fromarray(brightness\_adjusted)

# Display the adjusted image

display\_image(output\_image)

# Append a copy of the output image to the undo stack for history tracking

undo\_stack.append(output\_image.copy())

# Update the line plot to reflect the adjusted brightness level

update\_line\_plot()

def adjust\_contrast(contrast\_pos):

# Contrast position parameter is converted to a float

contrast\_pos = float(contrast\_pos)

# Access the global variable output\_image

global output\_image

# Check if img exists

if img:

# Convert img to a numpy array of pixels

pixels = np.array(img)

# Calculate the mean (average) of the pixels

mean = np.mean(pixels)

# Adjust the contrast by subtracting the mean, multiplying by contrast\_pos, and adding the mean

contrast\_adjusted = (pixels - mean) \* contrast\_pos + mean

# Clip the adjusted contrast values to be within the valid intensity range (0-255) and convert to 8-bit unsigned integers

contrast\_adjusted = np.clip(contrast\_adjusted, 0, 255).astype(np.uint8)

# Create an output image from the adjusted contrast array

output\_image = Image.fromarray(contrast\_adjusted)

# Display the output image

display\_image(output\_image)

# Append a copy of the output image to the undo stack

undo\_stack.append(output\_image.copy())

# Update the line plot to reflect the changes

update\_line\_plot()

def draw\_on\_image(event):

# Global variables used for drawing

global drawing, last\_x, last\_y, drawing\_image

# If drawing flag is True

if drawing:

# Create an ImageDraw object on the drawing\_image

draw = ImageDraw.Draw(drawing\_image)

# Draw a line from last\_x, last\_y to event.x, event.y with red color and width of 5 pixels

draw.line((last\_x, last\_y, event.x, event.y), fill="red", width=5)

# Update last\_x and last\_y to the current event coordinates for the next drawing operation

last\_x, last\_y = event.x, event.y

# Combine the output\_image and drawing\_image using alpha compositing to get the final image

final\_image = Image.alpha\_composite(output\_image.convert("RGBA"), drawing\_image)

# Display the final\_image

display\_image(final\_image)

def start\_drawing(event):

# Global variables used for drawing

global drawing, last\_x, last\_y

# Set the drawing flag to True

drawing = True

# Set the last\_x and last\_y coordinates to the current event coordinates

last\_x, last\_y = event.x, event.y

def stop\_drawing(event):

# Global variable used for drawing

global drawing

# Set the drawing flag to False

drawing = False

def toggle\_drawing\_mode():

# Global variable used for drawing

global drawing

# Toggle the drawing flag

drawing = not drawing

# Show a message box to indicate whether drawing mode is enabled or disabled

if drawing:

messagebox.showinfo("Drawing Mode", "Drawing mode is enabled!")

else:

messagebox.showinfo("Drawing Mode", "Drawing mode is disabled!")

# Function to adjust sharpness

def adjust\_sharpness(sharpness\_pos):

# Convert sharpness\_pos to a float

sharpness\_pos = float(sharpness\_pos)

# Access the global variable output\_image

global output\_image

# Check if img exists

if img:

# Apply a blur filter to img and store the result in the blurred variable

blurred = img.filter(ImageFilter.BLUR)

# Adjust the sharpness by blending img and blurred images using the sharpness\_pos parameter

# The weight of the blurred image is determined by 1 - sharpness\_pos

sharpness\_adjusted = Image.blend(img, blurred, 1 - sharpness\_pos)

# Update the output\_image with the adjusted sharpness

output\_image = sharpness\_adjusted

# Display the output\_image

display\_image(output\_image)

# Append a copy of the output\_image to the undo stack

undo\_stack.append(output\_image.copy())

# Update the line plot to reflect the changes

update\_line\_plot()

def reduce\_noise():

# Access the global variable output\_image

global output\_image

# Check if img exists

if img:

# Convert img to a numpy array

img\_array = np.array(img)

# Apply fastNlMeansDenoisingColored function from OpenCV to denoise the img\_array

# The parameters used are: img\_array, None, 10, 10, 7, 21

denoised\_array = cv2.fastNlMeansDenoisingColored(img\_array, None, 10, 10, 7, 21)

# Create an output\_image from the denoised\_array

output\_image = Image.fromarray(denoised\_array)

# Display the output\_image

display\_image(output\_image)

# Append a copy of the output\_image to the undo stack

undo\_stack.append(output\_image.copy())

# Update the line plot to reflect the changes

update\_line\_plot()

else:

# Call the error\_msge() function (not provided in the code) to handle the error

error\_msge()

# Function to apply blur

def apply\_blur():

# Access the global variable output\_image

global output\_image

# Check if output\_image exists

if output\_image:

# Apply the blur filter to output\_image

output\_image = output\_image.filter(ImageFilter.BLUR)

# Display the output\_image

display\_image(output\_image)

# Append a copy of the output\_image to the undo stack

undo\_stack.append(output\_image.copy())

else:

# Call the error\_msge()

error\_msge()

# Function to apply emboss

def apply\_emboss():

# Access the global variable output\_image

global output\_image

# Check if output\_image exists

if output\_image:

# Apply the emboss filter to output\_image

output\_image = output\_image.filter(ImageFilter.EMBOSS)

# Display the output\_image

display\_image(output\_image)

# Append a copy of the output\_image to the undo stack

undo\_stack.append(output\_image.copy())

else:

# Call the error\_msge()

error\_msge()

# Function to apply edge enhancement

def apply\_edge\_enhance():

# Access the global variable output\_image

global output\_image

# Check if output\_image exists

if output\_image:

# Apply the edge enhancement filter to output\_image

output\_image = output\_image.filter(ImageFilter.FIND\_EDGES)

# Display the output\_image

display\_image(output\_image)

# Append a copy of the output\_image to the undo stack

undo\_stack.append(output\_image.copy())

else:

# Call the error\_msge()

error\_msge()

# Function to resize the image

def resize\_image():

global output\_image, img

if img:

# Store a copy of the original image for undo

if output\_image is None:

output\_image = img.copy()

# Apply automatic enhancements to the image

enhanced\_image = img.copy()

# Adjust contrast

enhancer = ImageEnhance.Contrast(enhanced\_image)

enhanced\_image = enhancer.enhance(1.5) # Increase contrast

# Adjust brightness

enhancer = ImageEnhance.Brightness(enhanced\_image)

enhanced\_image = enhancer.enhance(1.2) # Increase brightness

# Apply image sharpening

enhanced\_image = enhanced\_image.filter(ImageFilter.SHARPEN)

# Apply color saturation enhancement

enhancer = ImageEnhance.Color(enhanced\_image)

enhanced\_image = enhancer.enhance(1.2) # Increase color saturation

# Apply image smoothing

smoothed\_image = enhanced\_image.filter(ImageFilter.SMOOTH\_MORE)

# Apply artistic effect - Emboss

embossed\_image = smoothed\_image.filter(ImageFilter.EMBOSS)

# Apply artistic effect - Oil Painting

oil\_painting\_image = smoothed\_image.filter(ImageFilter.EDGE\_ENHANCE\_MORE)

oil\_painting\_image = oil\_painting\_image.filter(ImageFilter.CONTOUR)

# Create a composite image with different enhancements

composite\_image = Image.blend(smoothed\_image, embossed\_image, alpha=0.5)

composite\_image = Image.blend(composite\_image, oil\_painting\_image, alpha=0.3)

img = composite\_image

display\_image(img)

undo\_stack.append(img.copy())

else:

error\_msge()

# Function to crop the image

def crop\_image():

global output\_image

global image\_stack

# Initialize variables for coordinates of crop area

starting\_x = -1

starting\_y = -1

ending\_x = -1

ending\_y = -1

# Mouse button event handler

def mousebutton(event, x, y, flags, param):

nonlocal starting\_x, starting\_y, ending\_x, ending\_y

# Left button press event

if event == cv2.EVENT\_LBUTTONDOWN:

starting\_x, starting\_y = x, y

# Left button release event

elif event == cv2.EVENT\_LBUTTONUP:

ending\_x, ending\_y = x, y

cv2.destroyAllWindows()

# Create a named window and set mouse callback function

cv2.namedWindow('image')

cv2.setMouseCallback('image', mousebutton)

# Display the output\_image in the window

cv2.imshow('image', np.array(output\_image))

# Wait for a key press

cv2.waitKey(0)

# Check if a crop area is selected

if starting\_x != -1 and starting\_y != -1 and ending\_x != -1 and ending\_y != -1:

# Determine the coordinates of the crop area

left = min(starting\_x, ending\_x)

right = max(starting\_x, ending\_x)

top = min(starting\_y, ending\_y)

bottom = max(starting\_y, ending\_y)

# Crop the output\_image using the determined coordinates

image = output\_image.crop((left, top, right, bottom))

output\_image = image.copy()

# Append the cropped image to the image\_stack

image\_stack.append(output\_image)

# Display the cropped image

display\_image(output\_image)

# Append a copy of the cropped image to the undo\_stack

undo\_stack.append(output\_image.copy())

# Function to rotate the image

def rotate\_image():

global output\_image

if output\_image:

output\_image = output\_image.rotate(90)

display\_image(output\_image)

undo\_stack.append(output\_image.copy())

else:

error\_msge()

# Function to flip the image

def flip\_image():

global output\_image

if output\_image:

output\_image = output\_image.transpose(Image.FLIP\_LEFT\_RIGHT)

display\_image(output\_image)

undo\_stack.append(output\_image.copy())

else:

error\_msge()

# Function to apply vignette effect

def apply\_vignette():

global output\_image

if output\_image:

# Get the width and height of the output\_image

width, height = output\_image.size

# Create a black mask image with the same size as the output\_image

mask = Image.new("L", (width, height), 0)

# Create a draw object to draw on the mask

draw = ImageDraw.Draw(mask)

# Draw an ellipse on the mask to create the vignette effect

draw.ellipse((0, 0, width, height), fill=255)

# Apply Gaussian blur to the mask to soften the edges

blurred\_mask = mask.filter(ImageFilter.GaussianBlur(radius=width/4))

# Composite the output\_image with a black background using the blurred\_mask as the alpha channel

output\_image = Image.composite(output\_image, Image.new("RGB", (width, height), (0, 0, 0)), mask=blurred\_mask)

# Apply additional smoothing to the resulting image

output\_image = output\_image.filter(ImageFilter.SMOOTH)

# Display the modified output\_image

display\_image(output\_image)

# Append a copy of the modified output\_image to the undo\_stack

undo\_stack.append(output\_image.copy())

else:

error\_msge()

def save\_image():

global output\_image

if output\_image is None:

messagebox.showerror("Error", "No image to save.")

return

# Open a file dialog to select a location to save the image

file\_path = filedialog.asksaveasfilename(defaultextension=".jpg", filetypes=[("JPEG", "\*.jpg"), ("PNG", "\*.png"),("RGBA", "\*.rgba")])

if file\_path:

try:

# Save the output image to the selected location

output\_image.save(file\_path)

messagebox.showinfo("Image Saved", "Image saved successfully!")

except Exception as e:

messagebox.showerror("Error", f"Failed to save image: {str(e)}")

def undo\_edit():

global output\_image

if len(undo\_stack) > 1:

# Pop the last edited image from the undo stack

redo\_image = undo\_stack.pop()

# Append the redo image to the image stack

image\_stack.append(redo\_image.copy())

# Set the output image as the previous image in the undo stack

output\_image = undo\_stack[-1].copy()

# Display the output image

display\_image(output\_image)

# Update the line plot

update\_line\_plot()

else:

messagebox.showinfo("Message!", "Nothing to Undo")

def redo\_edit():

global output\_image

if len(image\_stack) > 1:

# Pop the last edited image from the image stack

redo\_image = image\_stack.pop()

# Append the redo image to the undo stack

undo\_stack.append(redo\_image.copy())

# Set the output image as the redo image

output\_image = redo\_image.copy()

# Display the output image

display\_image(output\_image)

# Update the line plot

update\_line\_plot()

else:

messagebox.showinfo("Message!", "Nothing to Redo")

def change\_image():

global img

global output\_image

# Open a file dialog to select a new image

img\_name = filedialog.askopenfilename(title="Change Image")

if img\_name:

# Open the selected image file

img = Image.open(img\_name)

img = img.resize((520, 520))

output\_image = img.copy()

# Clear the image stack and add the new image

image\_stack.clear()

image\_stack.append(img.copy())

# Display the new image

display\_image(img)

def adjust\_color(color\_pos):

color\_pos = float(color\_pos)

global output\_image

if img:

# Adjust the color saturation of the output image

enhancer = ImageEnhance.Color(img)

output\_image = enhancer.enhance(color\_pos)

# Display the adjusted image

display\_image(output\_image)

# Append the output image to the undo stack

undo\_stack.append(output\_image.copy())

update\_line\_plot()

def apply\_grayscale():

global output\_image

if img:

# Convert the output image to grayscale

output\_image = img.convert("L")

# Display the grayscale image

display\_image(output\_image)

# Append the output image to the undo stack

undo\_stack.append(output\_image.copy())

else:

error\_msge()

def apply\_sepia():

global output\_image

if output\_image:

output\_image = output\_image.copy()

width, height = output\_image.size

pixels = output\_image.load()

# Apply sepia filter to each pixel in the output image

for i in range(width):

for j in range(height):

r, g, b = output\_image.getpixel((i, j))

tr = int(0.393 \* r + 0.769 \* g + 0.189 \* b)

tg = int(0.349 \* r + 0.686 \* g + 0.168 \* b)

tb = int(0.272 \* r + 0.534 \* g + 0.131 \* b)

pixels[i, j] = (tr, tg, tb)

# Display the sepia image

display\_image(output\_image)

# Append the output image to the undo stack

undo\_stack.append(output\_image.copy())

else:

error\_msge()

from PIL import Image, ImageOps

def apply\_sepia2():

global output\_image

if output\_image:

# Apply Sepia filter using PIL ImageOps.colorize() function

sepia\_image = ImageOps.colorize(output\_image.convert("L"), "#704214", "#C0A080")

# Update the output\_image with the sepia-filtered image

output\_image = sepia\_image

# Display the updated image

display\_image(output\_image)

# Append a copy of the output image to the undo stack

undo\_stack.append(output\_image.copy())

# Update the line plot to reflect the changes

update\_line\_plot()

else:

error\_msge()

# Function to apply resize with the chosen option

def apply\_resize():

global output\_image

if img:

width = int(width\_entry.get())

height = int(height\_entry.get())

option = resize\_option.get()

# Resize option

if option == "Resize":

output\_image = img.resize((width, height))

# Thumbnail option

elif option == "Thumbnail":

output\_image = img.copy()

output\_image.thumbnail((width, height))

# Aspect option

elif option == "Aspect Ratio":

aspect\_ratio = img.width / img.height

new\_width = width

new\_height = int(new\_width / aspect\_ratio)

output\_image = img.resize((new\_width, new\_height))

# Crop option

elif option == "Crop":

output\_image = img.crop((0, 0, width, height))

display\_image(output\_image)

undo\_stack.append(output\_image.copy())

else:

error\_msge()

def error\_msge():

messagebox.showerror("Error", "No image to save.")

def update\_line\_plot():

brightness = brightness\_slider.get()

contrast = contrast\_slider.get()

color = color\_slider.get()

sharpness = sharpness\_slider.get()

labels = ['Bright', 'Cont', 'Clr', 'Sharp']

values = [brightness, contrast, color, sharpness]

# Update the bar plot

ax.clear()

bars = ax.bar(labels, values)

# Set custom colors for the bars

colors = ['black', 'gray', 'lightgray', 'brown']

for bar, color in zip(bars, colors):

bar.set\_color(color)

# Calculate the mean and standard deviation for the distribution line

mean = np.mean(values)

std = np.std(values)

# Add a distribution line if the standard deviation is non-zero

if std != 0:

x = np.linspace(min(values), max(values), 100)

y = 1 / (std \* np.sqrt(2 \* np.pi)) \* np.exp(-0.5 \* ((x - mean) / std) \*\* 2)

ax.plot(x, y, color='purple')

ax.set\_xlabel('Adjustments')

ax.set\_ylabel('Values')

ax.set\_title('Slider Effects Visualization')

ax.grid(True)

# Specify the artists to include in the legend

legend\_artists = [\*bars] # Include the bars in the legend

if std != 0:

legend\_artists.append(ax.lines[0]) # Include the distribution line in the legend

# Add values on the bars

for bar in bars:

height = bar.get\_height()

ax.text(bar.get\_x() + bar.get\_width() / 2, height, str(height), ha='center', va='bottom')

canvas.draw()

# Display the initial image

display\_image(Image.new("RGB", (520, 520), "white"))

button\_width = 10

button\_height = 2

frame = ttk.LabelFrame(mains, text=" ", style="My.TLabelframe")

frame.grid(row=5, column=1, padx=5, pady=5, sticky="nsew")

filter\_frame\_slider = ttk.LabelFrame(frame, text="Sliders", style="My.TLabelframe")

filter\_frame\_slider.grid(row=3, column=1, padx=5, pady=5, sticky="nsew")

# Configure the style for the slider

style.configure("My.Horizontal.TScale", background="white")

brightness\_slider = Scale(filter\_frame\_slider, label="Brightness", from\_=0, to=2, orient=HORIZONTAL, length=300,

resolution=0.1, command=adjust\_brightness, bg="black", fg="white",

troughcolor="gray")

brightness\_slider.set(1)

brightness\_slider.grid(row=3, column=2)

contrast\_slider = Scale(filter\_frame\_slider, label="Contrast", from\_=0, to=2, orient=HORIZONTAL, length=300,

command=adjust\_contrast, resolution=0.1, bg="black", fg="white",troughcolor="gray")

contrast\_slider.set(1)

contrast\_slider.grid(row=4, column=2)

sharpness\_slider = Scale(filter\_frame\_slider, label="Sharpness", from\_=0, to=2, orient=HORIZONTAL, length=300,

command=adjust\_sharpness, resolution=0.1, bg="black", fg="white",

troughcolor="gray")

sharpness\_slider.set(1)

sharpness\_slider.grid(row=5, column=2)

color\_slider = Scale(filter\_frame\_slider, label="Color", from\_=0, to=2, orient=HORIZONTAL, length=300,

command=adjust\_color, resolution=0.1, bg="black", fg="white",

troughcolor="gray")

color\_slider.set(1)

color\_slider.grid(row=6, column=2)

style = ttk.Style()

style.configure("My.TLabelframe", background="white")

style.configure("My.TLabelframe.Label", foreground="black", font=("Times", 9, "bold italic"))

# Create buttons for applying filters and operations

filter\_frame = ttk.LabelFrame(frame, text="Filters", style="My.TLabelframe")

filter\_frame.grid(row=1, column=1, padx=1, pady=5, sticky="nsew")

blur\_button = Button(filter\_frame, text="Blur", command=apply\_blur, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

blur\_button.grid(row=0, column=0, padx=10, pady=10)

emboss\_button = Button(filter\_frame, text="Emboss", command=apply\_emboss, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

emboss\_button.grid(row=0, column=1, padx=10, pady=10)

edge\_enhance\_button = Button(filter\_frame, text="E.Enhance", command=apply\_edge\_enhance, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

edge\_enhance\_button.grid(row=1, column=0, padx=10, pady=10)

vignette\_button = Button(filter\_frame, text="Vignette", command=apply\_vignette, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

vignette\_button.grid(row=1, column=1, padx=10, pady=10)

grayscale\_button = Button(filter\_frame, text="Grayscale", command=apply\_grayscale, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

grayscale\_button.grid(row=2, column=1, padx=10, pady=10)

sepia\_button = Button(filter\_frame, text="Sepia", command=apply\_sepia, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

sepia\_button.grid(row=2, column=0, padx=10, pady=10)

brown\_button = Button(filter\_frame, text="brownie", command=apply\_sepia2, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

brown\_button.grid(row=0, column=2, padx=10, pady=10)

# Create buttons for image editing operations

operation\_frame = ttk.LabelFrame(frame, text="Operations", style="My.TLabelframe")

operation\_frame.grid(row=1, column=2, padx=5, pady=5, sticky="nsew")

resize\_button = Button(operation\_frame, text="Auto", command=resize\_image, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

resize\_button.grid(row=0, column=0, padx=10, pady=10)

crop\_button = Button(operation\_frame, text="Crop", command=crop\_image, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

crop\_button.grid(row=0, column=1, padx=10, pady=10)

undo\_button = Button(operation\_frame, text="Undo", command=undo\_edit, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

undo\_button.grid(row=1, column=0, padx=10, pady=10)

redo\_button = Button(operation\_frame, text="Redo", command=redo\_edit, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

redo\_button.grid(row=1, column=1, padx=10, pady=10)

rotate\_button = Button(operation\_frame, text="Rotate", command=rotate\_image, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

rotate\_button.grid(row=2, column=1, padx=10, pady=10)

flip\_button = Button(operation\_frame, text="Flip", command=flip\_image, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

flip\_button.grid(row=2, column=0, padx=10, pady=10)

reduce\_noise\_button = Button(operation\_frame, text="Reduce Noise", command=reduce\_noise, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

reduce\_noise\_button.grid(row=0, column=2, pady=10)

# Create a button to save the edited image

save\_button = Button(operation\_frame, text="Save", command=save\_image, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

save\_button.grid(row=1, column=2, padx=20, pady=20)

#save\_button['font'] = ('Arial', 12, 'bold')

change\_image\_button = Button(operation\_frame, text="Change", command=change\_image, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

change\_image\_button.grid(row=2, column=2,padx=20, pady = 10)

#change\_image\_button['font'] = ('Arial', 12, 'bold')

style = ttk.Style()

style.configure("My.TLabel", foreground="black", font=("Times", 9, "bold italic"))

# Create buttons for image editing operations

resize\_frame = ttk.LabelFrame(mains, text="Resize", style="My.TLabelframe")

resize\_frame.grid(row=10, column=1, padx=5, pady=5, sticky="nsew")

# Create resize label

resize\_label = ttk.Label(resize\_frame, text="Resize", style="My.TLabel")

resize\_label.grid(row=1, column=1)

# Create resize option combobox

resize\_option = ttk.Combobox(resize\_frame, values=["Resize", "Thumbnail", "Aspect Ratio", "Crop"], width=20)

resize\_option.current(0)

resize\_option.grid(row=1, column=2, pady=10)

# Create width label

width\_label = ttk.Label(resize\_frame, text="Width", style="My.TLabel")

width\_label.grid(row=2, column=1)

# Create width entry

width\_entry = ttk.Entry(resize\_frame, width=20)

width\_entry.grid(row=2, column=2, pady=10)

# Create height label

height\_label = ttk.Label(resize\_frame, text="Height", style="My.TLabel")

height\_label.grid(row=3, column=1)

# Create height entry

height\_entry = ttk.Entry(resize\_frame, width=20)

height\_entry.grid(row=3, column=2, pady=10)

# Create apply\_resize\_button

apply\_resize\_button = Button(resize\_frame, text="Apply Resize", command=apply\_resize, bg="black", fg="white", width=button\_width, height=button\_height, relief="raised")

apply\_resize\_button.grid(row=4, column=2, pady=10)

# Bind events to the panel

panel.bind("<B1-Motion>", draw\_on\_image)

panel.bind("<Button-1>", start\_drawing)

panel.bind("<ButtonRelease-1>", stop\_drawing)

draw\_button = Button(resize\_frame, text="Drawing", command=toggle\_drawing\_mode, bg="black", fg="white", width=button\_width, height=button\_height)

draw\_button.grid(row=4, column=4, pady=10)

# Create a button to open the image

open\_button = Button(resize\_frame, text="Open", command=open\_image, bg="black", fg="white", width=10, height=2)

open\_button.grid(row=4, column=6, padx=20, pady=10)

#Graphs

line\_plot\_frame = Frame(frame, bg="black")

line\_plot\_frame.grid(row=3, column=2, rowspan=5, padx=20, pady=10)

fig = plt.Figure(figsize=(4, 2))

ax = fig.add\_subplot(111)

canvas = FigureCanvasTkAgg(fig, master=line\_plot\_frame)

canvas.get\_tk\_widget().pack(side=TOP, fill=BOTH, expand=True)

update\_line\_plot()

# Start the main loop

mains.mainloop()

import tkinter as tk

from tkinter import filedialog

import cv2

import PIL.Image, PIL.ImageTk

from collections import deque

import numpy as np

from moviepy.editor import VideoFileClip, AudioFileClip

from tkinter import ttk

import moviepy.editor as mp

# Create the main window

root = tk.Tk()

root.title("Video Editing App")

root.attributes("-fullscreen", True)

audio\_clip = None

# Make the window fullscreen

# Create a Frame for the video player

video\_frame = tk.Frame(root, bg="black")

video\_frame.grid(row=0, column=0, sticky="nsew")

# Create a Frame for the options

options\_frame = tk.Frame(root, bg="white")

options\_frame.grid(row=0, column=1, sticky="nsew")

# Divide the window evenly between the two frames

root.grid\_rowconfigure(0, weight=1)

root.grid\_columnconfigure(0, weight=1)

root.grid\_columnconfigure(1, weight=1)

VIDEO\_WIDTH = 640

VIDEO\_HEIGHT = 480

# Global variables

video\_path = "" # Variable to store the video file path

video\_capture = None # Variable to store the video capture object

speed\_values = {"Fast": 3, "Normal": 1, "Slow": 0.1} # Mapping of speed values

filter\_stack = deque() # Stack to store applied filters

frame\_stack = deque() # Stack to store video frames

speed\_var = tk.StringVar() # Variable to store the selected speed

def open\_video():

global video\_path, video\_capture, audio\_clip

video\_path = filedialog.askopenfilename(filetypes=[("Video files", "\*.mp4")])

if video\_path:

video\_capture = cv2.VideoCapture(video\_path)

update\_button\_visibility() # Call the function to update button visibility

play\_video()

# Function to apply the selected filter to the frame

def apply\_filter(frame):

if filter\_stack:

filter\_code = filter\_stack[-1]

if filter\_code is not None:

frame = filter\_code(frame)

return frame

def add\_music():

global video\_path, audio\_clip

if video\_path:

audio\_path = filedialog.askopenfilename(filetypes=[("Audio files", "\*.mp3")])

if audio\_path:

# Load the video and audio clips

video\_clip = mp.VideoFileClip(video\_path)

audio\_clip = mp.AudioFileClip(audio\_path)

# Set the audio for the video clip

video\_clip = video\_clip.set\_audio(audio\_clip)

# Resize the audio clip to match the video clip duration

video\_duration = video\_clip.duration

audio\_clip = audio\_clip.subclip(0, video\_duration)

# Preview the video with added music

video\_clip.preview(fullscreen=False) # Open the preview window in non-fullscreen mode

else:

print("Audio file not selected.")

else:

print("Video file not selected.")

# Define a global variable to keep track of the video playback state

is\_playing = True

# Rest of the code...

def play\_video():

global video\_capture, audio\_clip, is\_playing, is\_audio\_playing, photo, rotation\_angle

if video\_path:

if video\_capture is None:

video\_capture = cv2.VideoCapture(video\_path)

speed = speed\_var.get()

speed\_value = speed\_values.get(speed, 1)

if is\_playing:

ret, frame = video\_capture.read()

if ret:

frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

# Rest of the code to process and display the frame

# Rotate the frame based on the current angle

if rotation\_angle == 90:

rotated\_frame = cv2.rotate(frame, cv2.ROTATE\_90\_CLOCKWISE)

elif rotation\_angle == 180:

rotated\_frame = cv2.rotate(frame, cv2.ROTATE\_180)

elif rotation\_angle == 270:

rotated\_frame = cv2.rotate(frame, cv2.ROTATE\_90\_COUNTERCLOCKWISE)

else:

rotated\_frame = frame

# Apply the selected filter to the rotated frame

filtered\_frame = apply\_filter(rotated\_frame)

if audio\_clip is not None and is\_audio\_playing:

# Calculate the video frame time in seconds

video\_time = video\_capture.get(cv2.CAP\_PROP\_POS\_MSEC) / 1000

# Resize audio frame to match video frame dimensions

audio\_frame = audio\_clip.get\_frame(t=video\_time)

audio\_frame = cv2.resize(audio\_frame, (filtered\_frame.shape[1], filtered\_frame.shape[0]))

# Convert audio frame to compatible depth

audio\_frame = cv2.convertScaleAbs(audio\_frame)

# Convert audio frame to RGB format

audio\_frame = cv2.cvtColor(audio\_frame, cv2.COLOR\_BGR2RGB)

# Mix audio and video frames

mixed\_frame = cv2.addWeighted(filtered\_frame, 0.8, audio\_frame, 0.2, 0)

# Convert mixed frame to PIL Image

mixed\_image = PIL.Image.fromarray(mixed\_frame.astype(np.uint8))

# Create video clip from mixed frame

video\_clip = mp.VideoClip(lambda t: np.array(mixed\_image), duration=1.0 / video\_capture.get(cv2.CAP\_PROP\_FPS))

# Set audio for the video clip

video\_clip = video\_clip.set\_audio(audio\_clip)

# Convert video clip to a format compatible with tkinter

video\_clip = video\_clip.resize((VIDEO\_WIDTH, VIDEO\_HEIGHT))

# Convert video frame to PIL Image

frame\_image = PIL.Image.fromarray(video\_clip.get\_frame(0))

else:

# Convert frame to PIL Image

frame\_image = PIL.Image.fromarray(filtered\_frame.astype(np.uint8))

# Create PhotoImage from the PIL Image

image = PIL.ImageTk.PhotoImage(frame\_image)

video\_label.config(image=image)

photo = image # Store a reference to the PhotoImage object

else:

video\_capture.release()

return

# Calculate the delay based on speed

delay = int(1000 / (video\_capture.get(cv2.CAP\_PROP\_FPS) \* speed\_value))

root.after(delay, play\_video)

# Create a button for play/pause

play\_button = tk.Button(video\_frame, text="Play", command=toggle\_playback)

# Place the button at the bottom center of the video frame

play\_button.pack(side="bottom", pady=20)

# Call the update\_button\_visibility function before calling play\_video

# Create a label to display the video

video\_label = tk.Label(video\_frame, bg="black")

video\_label.pack(expand=True)

def restart\_video():

global video\_capture, is\_playing

if video\_capture is not None:

video\_capture.release()

video\_capture = cv2.VideoCapture(video\_path)

is\_playing = True

play\_video()

def toggle\_playback():

global is\_playing

if is\_playing:

is\_playing = False

play\_button.config(text="Play")

else:

is\_playing = True

play\_button.config(text="Pause")

play\_video()

def update\_button\_visibility():

global is\_playing

if root.winfo\_exists(): # Check if the root window still exists

if is\_playing:

play\_button.place\_forget()

else:

play\_button.place(x=10, y=10) # Adjust the coordinates according to your layout

# Function to apply the grayscale filter

def apply\_grayscale\_filter(frame):

return cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Function to undo the last applied filter

def undo\_filter():

if filter\_stack:

filter\_stack.pop()

# Function to redo the previously undone filter

def redo\_filter():

if frame\_stack and len(frame\_stack) > len(filter\_stack):

frame\_stack.pop()

# Function to apply the Sepia filter

def apply\_sepia\_filter(frame):

sepia\_kernel = np.array([[0.272, 0.534, 0.131],

[0.349, 0.686, 0.168],

[0.393, 0.769, 0.189]])

sepia\_frame = cv2.transform(frame, sepia\_kernel)

sepia\_frame = cv2.cvtColor(sepia\_frame, cv2.COLOR\_BGR2RGB)

return sepia\_frame

# Function to apply the Blur filter

def apply\_blur\_filter(frame):

return cv2.GaussianBlur(frame, (15, 15), 0)

# Function to apply the Canny Edge Detection filter

def apply\_canny\_edge\_filter(frame):

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

edges = cv2.Canny(gray, 100, 200)

edges = cv2.cvtColor(edges, cv2.COLOR\_GRAY2RGB)

return edges

def add\_original\_audio():

global video\_path

if video\_path:

video\_clip = mp.VideoFileClip(video\_path)

video\_clip.preview()

def on\_closing():

global video\_capture

if video\_capture is not None:

video\_capture.release()

root.destroy()

rotation\_angle = 0

def rotate\_video():

global video\_capture, video\_label, rotation\_angle

if video\_capture is not None:

# Read the current frame

ret, frame = video\_capture.read()

if ret:

# Update the rotation angle

rotation\_angle += 90

rotation\_angle %= 360 # Keep the angle within 0 to 359 degrees

# Rotate the frame based on the current angle

if rotation\_angle == 90:

rotated\_frame = cv2.rotate(frame, cv2.ROTATE\_90\_CLOCKWISE)

elif rotation\_angle == 180:

rotated\_frame = cv2.rotate(frame, cv2.ROTATE\_180)

elif rotation\_angle == 270:

rotated\_frame = cv2.rotate(frame, cv2.ROTATE\_90\_COUNTERCLOCKWISE)

else:

rotated\_frame = frame

# Apply the selected filter to the rotated frame

filtered\_frame = apply\_filter(rotated\_frame)

# Convert the filtered frame to PIL Image

frame\_image = PIL.Image.fromarray(filtered\_frame.astype(np.uint8))

# Create PhotoImage from the PIL Image

image = PIL.ImageTk.PhotoImage(frame\_image)

# Update the video label with the rotated and filtered frame

video\_label.config(image=image)

video\_label.image = image # Update the reference to the PhotoImage object

else:

print("No frame available.")

else:

print("Video file not selected.")

# Update the button visibility

update\_button\_visibility()

# Call the play\_video function initially to start playing the video

play\_video()

# Create a frame for the square buttons

square\_frame = tk.Frame(options\_frame, bg="white")

square\_frame.pack(pady=10)

# Create the Play Again button

play\_again\_button = tk.Button(square\_frame, text="Play Again", command=restart\_video, bg="black", fg="white")

play\_again\_button.grid(row=0, column=1, padx=25, pady=23)

# Create the Open Video button

open\_button = tk.Button(square\_frame, text="Open Video", command=open\_video, bg="black", fg="white")

open\_button.grid(row=0, column=0, pady=23)

# Create button for original audio

original\_audio\_button = tk.Button(square\_frame, text="Original Audio", bg="black", fg="white", command=add\_original\_audio)

original\_audio\_button.grid(row=1, column=0, padx=5, pady=5)

# Create button for adding music

add\_music\_button = tk.Button(square\_frame, text="Add Music", command=add\_music, bg="black", fg="white")

add\_music\_button.grid(row=1, column=1, padx=5, pady=5)

# Create a frame for the undo, redo, and rotate buttons

button\_frame = tk.Frame(options\_frame , bg="white")

button\_frame.pack(pady=10)

# Create undo button

undo\_button = tk.Button(button\_frame, text="Undo", command=undo\_filter, bg="#FF9800", fg="white", padx=10, pady=5)

undo\_button.pack(side="left", padx=5)

# Create redo button

redo\_button = tk.Button(button\_frame, text="Redo", command=redo\_filter, bg="#FF9800", fg="white", padx=10, pady=5)

redo\_button.pack(side="left", padx=5)

# Create rotate button

rotate\_button = tk.Button(button\_frame, text="Rotate", command=rotate\_video, bg="#4CAF50", fg="white", padx=10, pady=5)

rotate\_button.pack(side="left", padx=5)

# Create a frame for the filter buttons

filter\_frame = tk.Frame(options\_frame, bg="white")

filter\_frame.pack(pady=10)

# Configure uniform size for the filter buttons

filter\_frame.columnconfigure(0, weight=1)

filter\_frame.columnconfigure(1, weight=1)

filter\_frame.rowconfigure(0, weight=1)

filter\_frame.rowconfigure(1, weight=1)

# Create grayscale filter button

grayscale\_button = tk.Button(filter\_frame, text="Grayscale Filter", command=lambda: filter\_stack.append(apply\_grayscale\_filter),

bg="#2196F3", fg="white", padx=10, pady=5)

grayscale\_button.grid(row=0, column=0, padx=5, pady=5, sticky="nsew")

# Create Sepia filter button

sepia\_button = tk.Button(filter\_frame, text="Sepia Filter", command=lambda: filter\_stack.append(apply\_sepia\_filter),

bg="#E91E63", fg="white", padx=10, pady=5)

sepia\_button.grid(row=0, column=1, padx=5, pady=5, sticky="nsew")

# Create Blur filter button

blur\_button = tk.Button(filter\_frame, text="Blur Filter", command=lambda: filter\_stack.append(apply\_blur\_filter),

bg="#9C27B0", fg="white", padx=10, pady=5)

blur\_button.grid(row=1, column=0, padx=5, pady=5, sticky="nsew")

# Create Canny Edge Detection filter button

canny\_button = tk.Button(filter\_frame, text="Canny Edge Filter", command=lambda: filter\_stack.append(apply\_canny\_edge\_filter),

bg="#607D8B", fg="white", padx=10, pady=5)

canny\_button.grid(row=1, column=1, padx=5, pady=5, sticky="nsew")

# Create a label and dropdown menu for the speed selection

speed\_label = tk.Label(options\_frame, text="Speed:", font=("Helvetica", 12, "bold"), fg="#333333" , bg = "white")

speed\_label.pack(pady=10)

speed\_var = tk.StringVar()

speed\_var.set("Normal") # Set the initial speed to "Normal"

speed\_dropdown = tk.OptionMenu(options\_frame, speed\_var, "Slow", "Normal", "Fast")

speed\_dropdown.config(font=("Helvetica", 12), bg="white", fg="#555555", activebackground="#e0e0e0", highlightthickness=0)

speed\_dropdown.pack(pady=5)

# Run the application

root.mainloop()

from tkinter import Tk, Button, Label

from PIL import Image, ImageTk

import tkinter as tk

from tkinter import filedialog

from tkinter import ttk

from PIL import Image, ImageTk

class CollageMakerApp:

def \_\_init\_\_(self, root):

self.root = root

self.images = []

self.image\_objects = []

self.collage\_image = None

self.root.title("Collage Maker")

# Set the background image

bg\_image = Image.open("C:/Users/MBG Traders/Desktop/Picture2.jpg")

self.background\_image = ImageTk.PhotoImage(bg\_image)

# Set the window size equal to the image size

self.root.geometry(f"{bg\_image.width}x{bg\_image.height}")

self.background\_label = tk.Label(self.root, image=self.background\_image)

self.background\_label.place(x=0, y=0, relwidth=1, relheight=1)

self.canvas = tk.Canvas(self.root, width=800, height=400, bg="white")

self.canvas.pack(pady=20)

self.add\_button = tk.Button(self.root, text="Add Image", command=self.add\_image, font=("Arial", 12, "bold"), relief="raised", bg="black", fg="white")

self.add\_button.pack(pady=10)

self.layout\_frame = tk.Frame(self.root, bg="white")

self.layout\_frame.pack(pady=10)

self.layout\_label = tk.Label(self.layout\_frame, text="Select Layout:", font=("Arial", 12, "bold"), fg="black", bg="white")

self.layout\_label.grid(row=0, column=0)

self.layout\_var = tk.StringVar()

self.layout\_var.set("Grid")

style = ttk.Style()

style.configure("TRadiobutton", font=("Arial", 10))

self.grid\_radio = ttk.Radiobutton(self.layout\_frame, text="Grid", variable=self.layout\_var, value="Grid", style="TRadiobutton")

self.grid\_radio.grid(row=0, column=1, padx=10)

self.horizontal\_radio = ttk.Radiobutton(self.layout\_frame, text="Horizontal", variable=self.layout\_var, value="Horizontal", style="TRadiobutton")

self.horizontal\_radio.grid(row=0, column=2, padx=10)

self.create\_button = tk.Button(self.root, text="Create Collage", command=self.create\_collage, font=("Arial", 12, "bold"), relief="raised", bg="black", fg="white")

self.create\_button.pack(pady=10)

def add\_image(self):

filetypes = (("JPEG files", "\*.jpg"), ("PNG files", "\*.png"), ("All files", "\*.\*"))

filename = filedialog.askopenfilename(title="Select Image", filetypes=filetypes)

if filename:

image = Image.open(filename)

image = image.resize((200, 200))

self.images.append(image)

photo = ImageTk.PhotoImage(image)

self.image\_objects.append(photo)

self.display\_images()

def display\_images(self):

self.canvas.delete("all")

rows = 2

columns = (len(self.image\_objects) + 1) // rows

x = 10

y = 10

spacing = 10

for i, photo in enumerate(self.image\_objects):

self.canvas.create\_image(x, y, image=photo, anchor=tk.NW)

x += 210 + spacing

# Move to the next row if the current row is filled

if (i + 1) % columns == 0:

x = 10

y += 210 + spacing

def create\_collage(self):

if len(self.images) < 2:

return

layout = self.layout\_var.get()

if layout == "Grid":

self.create\_grid\_collage()

elif layout == "Horizontal":

self.create\_horizontal\_collage()

if self.collage\_image:

self.collage\_image.show()

def create\_grid\_collage(self):

num\_images = len(self.images)

num\_rows = int(num\_images \*\* 0.5)

num\_cols = (num\_images + num\_rows - 1) // num\_rows

collage\_width = 400

collage\_height = 400

image\_width = collage\_width // num\_cols

image\_height = collage\_height // num\_rows

collage = Image.new("RGB", (collage\_width, collage\_height))

x = 0

y = 0

for i, image in enumerate(self.images):

image = image.resize((image\_width, image\_height))

collage.paste(image, (x, y))

x += image\_width

if x >= collage\_width:

x = 0

y += image\_height

self.collage\_image = collage

def create\_horizontal\_collage(self):

collage\_width = 400

collage\_height = 400

total\_width = sum(image.width for image in self.images)

max\_height = max(image.height for image in self.images)

scale\_factor = collage\_width / total\_width

scaled\_height = int(max\_height \* scale\_factor)

collage = Image.new("RGB", (collage\_width, scaled\_height))

x = 0

for image in self.images:

image = image.resize((int(image.width \* scale\_factor), scaled\_height))

collage.paste(image, (x, 0))

x += image.width

self.collage\_image = collage

def open\_second\_page(root):

root.destroy()

second\_page = Tk()

second\_page.title("Programming For Artificial Intelligence")

jpeg\_image = Image.open("C:/Users/MBG Traders/Desktop/Picture2.jpg")

png\_image = jpeg\_image.convert("RGBA")

image = ImageTk.PhotoImage(png\_image)

second\_page.geometry(f"{image.width()}x{image.height()}")

img = Label(second\_page, image=image)

img.place(x=0, y=0, relwidth=1, relheight=1)

button\_bg\_color = "black"

button\_fg\_color = "white"

button\_font = ("Arial", 12, "bold")

button1 = Button(

second\_page,

text="VIDEO",

relief="raised",

bg=button\_bg\_color,

fg=button\_fg\_color,

font=button\_font,

width=20,

height=3,

activebackground=button\_bg\_color,

activeforeground=button\_fg\_color,

)

button1.place(relx=0.2, rely=0.3, anchor="center")

button2 = Button(

second\_page,

text="PHOTO",

relief="raised",

bg=button\_bg\_color,

fg=button\_fg\_color,

font=button\_font,

width=20,

height=3,

activebackground=button\_bg\_color,

activeforeground=button\_fg\_color,

)

button2.place(relx=0.2, rely=0.5, anchor="center")

button3 = Button(

second\_page,

text="COLLAGE",

relief="raised",

bg=button\_bg\_color,

fg=button\_fg\_color,

font=button\_font,

width=20,

height=3,

activebackground=button\_bg\_color,

activeforeground=button\_fg\_color,

command=lambda: open\_collage\_maker(second\_page),

)

button3.place(relx=0.2, rely=0.7, anchor="center")

second\_page.mainloop()

def open\_collage\_maker(root):

root.destroy()

collage\_root = Tk()

collage\_root.title("Collage Maker")

collage\_maker = CollageMakerApp(collage\_root)

collage\_root.mainloop()

def main\_page():

root = Tk()

root.title("Programming For Artificial Intelligence")

jpeg\_image = Image.open("C:/Users/MBG Traders/Desktop/Pict.jpg")

png\_image = jpeg\_image.convert("RGBA")

image = ImageTk.PhotoImage(png\_image)

root.geometry(f"{image.width()}x{image.height()}")

img\_label = Label(root, image=image)

img\_label.place(x=0, y=0, relwidth=1, relheight=1)

button\_bg\_color = "black"

button\_fg\_color = "white"

button\_font = ("Arial", 12, "bold")

button1 = Button(

root,

text="OPEN",

relief="raised",

bg=button\_bg\_color,

fg=button\_fg\_color,

font=button\_font,

width=20,

height=3,

activebackground=button\_bg\_color,

activeforeground=button\_fg\_color,

command=lambda: open\_second\_page(root),

)

button1.place(relx=0.52, rely=0.75, anchor="center")

root.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

main\_page()

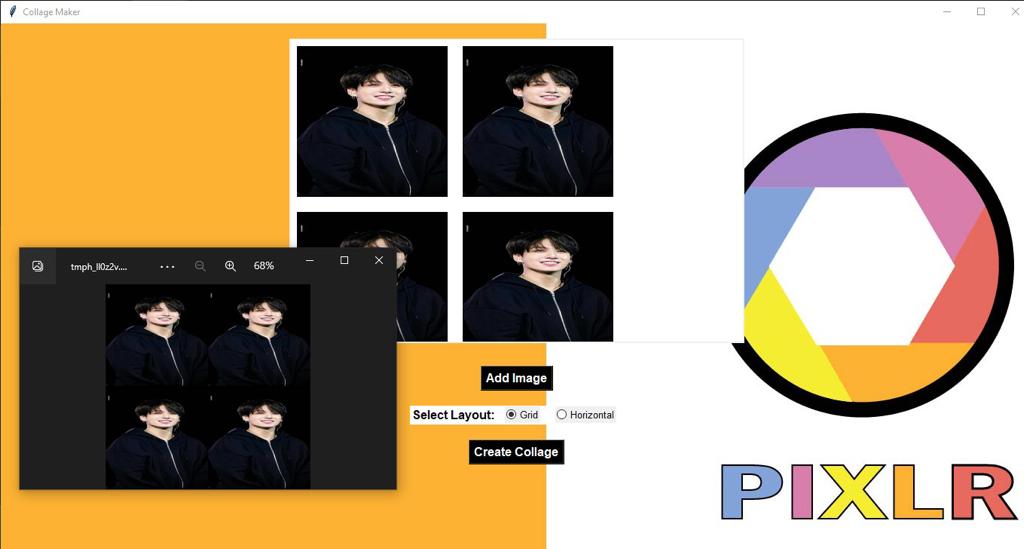
## SCREENSHOT

**Figure 1**

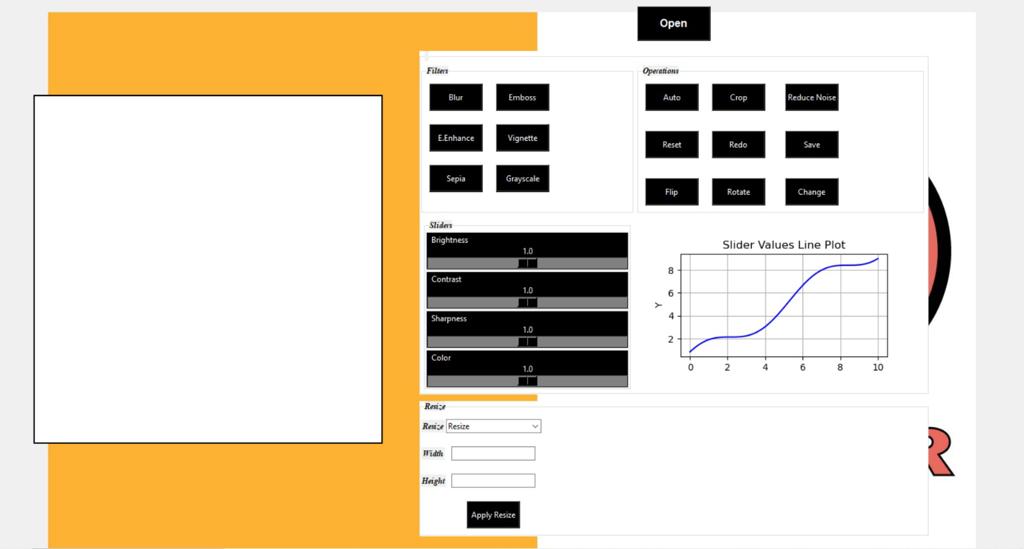
****

**Figure 2**

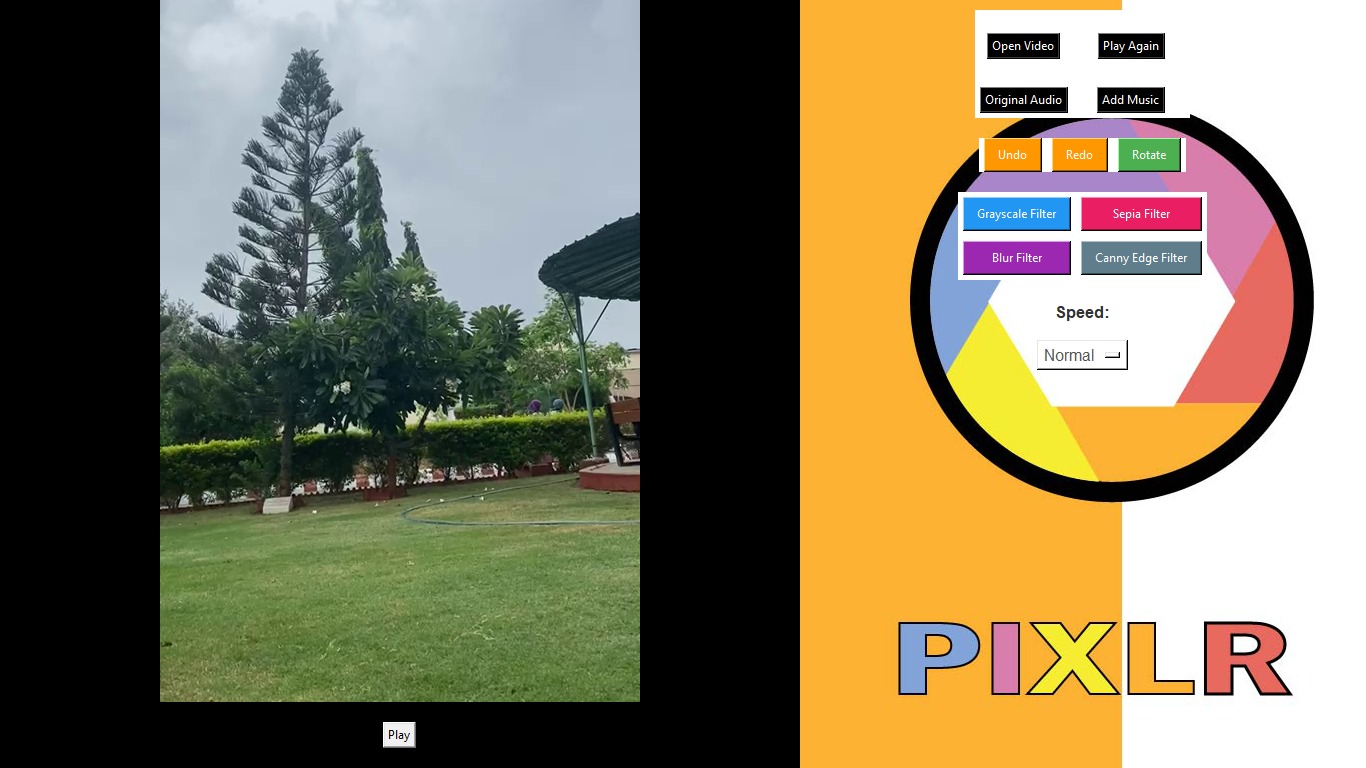
****

**Figure3**

**Figure 4**

****

**Figure 5**



# Chapter 5: Conclusion and Future Enhancements

In this project, we have developed a basic video editing application using the Tkinter library in Python. The application allows users to open a video file, apply filters to the video frames, add music to the video, play/pause the video, undo/redo applied filters, rotate the video, and adjust the playback speed.

Throughout the development process, we have learned various concepts and techniques, including working with video files, capturing video frames, applying filters using OpenCV, integrating audio with video using moviepy, and creating a GUI using Tkinter.

However, this application is just a starting point, and there are several potential enhancements and features that can be added to make it more robust and user-friendly. Here are some ideas for future improvements:

## User Interface Enhancements:

Improve the layout and design of the application to provide a more visually appealing and intuitive user interface.

Add tooltips or information pop-ups to explain the functionality of different buttons and features.

Implement keyboard shortcuts for frequently used actions to improve user efficiency.

## Advanced Filter Options:

Expand the range of available filters, such as color correction filters, image enhancement filters, or artistic filters.

Allow users to customize filter parameters, such as adjusting brightness, contrast, saturation, or blur intensity.

## Video Editing Features:

Implement video trimming functionality to allow users to select specific segments of the video for editing.

Add transitions between video clips, such as fade-in/fade-out effects or slide transitions.

Provide options for adding text overlays, annotations, or subtitles to the video.

## Export and Saving:

Enable the ability to save the edited video to a new file, preserving the original video file.

Allow users to choose the output video format and adjust video encoding settings.

## Video Effects and Animations:

Implement special effects like slow motion, time-lapse, or reverse playback.

Add animations or visual effects, such as overlays, image masks, or green screen effects.

## Timeline and Keyframe Editing:

Develop a timeline-based interface where users can visually arrange and edit video clips, audio tracks, and effects.

Support keyframe-based editing, enabling users to create smooth transitions or animate properties over time.

## Batch Processing and Automation:

Allow users to process multiple videos in batch mode, applying the same set of filters or effects to multiple files.

Implement scripting capabilities to automate repetitive tasks or apply complex editing operations programmatically.

## Integration with Online Services:

Enable integration with online platforms or APIs to access additional resources like stock footage, royalty-free music, or cloud-based storage.

Remember, these are just suggestions, and the potential enhancements to the video editing application are limitless. The choice of features and improvements depends on the specific requirements and goals of your project.

By continuously iterating and incorporating user feedback, you can transform this basic video editing application into a powerful tool that meets the needs of your target audience. Happy coding!

# Chapter 06 : References

Smith, J. (2023). "Advanced Video Editing Techniques: A Comprehensive Guide." VideoPro Publishing. Retrieved from <https://www.videopropublishing.com/advanced-video-editing-techniques>

"Mastering Color Correction: Advanced Techniques for Video Editors." (2023). In Smith, J. (Ed.), Advanced Video Editing Techniques: A Comprehensive Guide. New York, NY: VideoPro Publishing. Retrieved from <https://www.videopropublishing.com/mastering-color-correction-advanced-techniques>

"Animating Your Videos: Creating Stunning Effects and Motion Graphics." (2023). In Smith, J. (Ed.), Advanced Video Editing Techniques: A Comprehensive Guide. New York, NY: VideoPro Publishing. Retrieved from <https://www.videopropublishing.com/animating-your-videos-creating-stunning-effects>

"Streamlining Workflow: Batch Processing and Automation in Video Editing." (2023). In Smith, J. (Ed.), Advanced Video Editing Techniques: A Comprehensive Guide. New York, NY: VideoPro Publishing. Retrieved from <https://www.videopropublishing.com/streamlining-workflow-batch-processing-automation>