Image Processing Tool (ImageCraft)

5653630

January 29, 2025

Contents

1 Introduction

This report outlines the design, development, testing and evaluation of the Image Processing Software(ImageCraft), built using Python and libraries such as Tkinter, CustomTkinter, and PIL for both GUI and image processing.

2 Design

To effectively plan and implement my software, I developed a structure diagram to picture and break down the problem into manageable sections. This approach ensures a systematic development process, since we focus on each component before progressing to the next. Additionally, it provides a high-level overview of the program's functionality, reducing complexity and allowing it to be used as a reference point when planning the actual implementation.

2.1 Structure Diagram

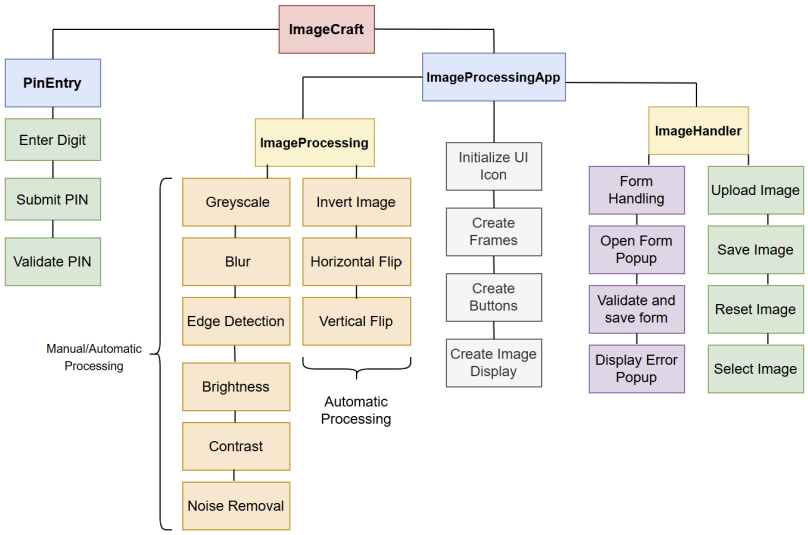

Figure 1: Structure Diagram

2.2 Colour Classification

Colour	Meaning (Function/Class)	Purpose
Blue	Main Class (Application)	Represents the main components of the application, managing core functionalities.
Green	Core Functions	Handles essential actions like uploading, saving, and resetting images.
Yellow	Subclasses	Divides program functionality between two classes, improving code encapsulation and modularity
Purple	Form Handling Functions	Manages input forms and validation for image metadata.
Grey	GUI Setup Functions	Sets up user interface components such as frames and buttons, for asthetics
Orange	Image Processing Functions	Provides image processing features like grayscale, blur, and contrast adjustments.

Table 1: Colour classification for the structure diagram

3 Flowcharts

As part of the design of my system, I made the flow charts shown in Figure 1,2 and 3. These outline the key logical flow and functionality of the PinEntry, ImageProcessing, and ImageHandler classes, which form the core logic of my program.

3.1 Assumptions

1. **Focus on Core Logic:**
   The flowcharts illustrate the fundamental logical flow and functionality of the `PinEntry`, `ImageProcessing`, and `ImageHandler` classes. They do not cover every minor implementation detail, instead they focus on the essential operations and decision-making processes, the users will have to take when using the software
2. **Common Use Cases:**
   They are based on the most typical user interactions and expected error scenarios. Edge cases and rare scenarios have been omitted to maintain clarity and conciseness.
3. **Class Independence Representation:**
   To avoid unnecessary complexity and enhance readability, the `ImageProcessing` and `ImageHandler` classes have been represented as separate entities, even though during implementation I will aim to ensure they are dynamically integrated .

3.2 Entry PIN Flowchart

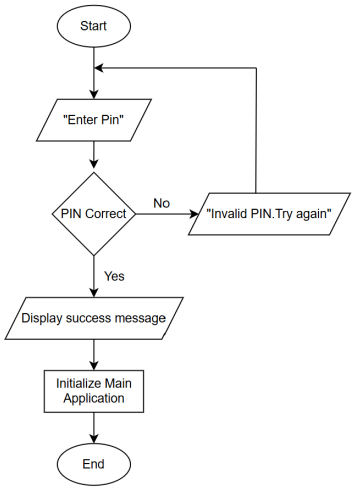

Figure 2: Entry PIN Flowchart

After the PIN is successfully authenticated, the **ImageProcessingApp** are initialised. This sets up the main GUI of the program,creating essential UI components such as frames, buttons, and image display areas. More importantly, it initialises the **ImageProcessing** and **ImageHandler** classes, enabling the user to make use of them by uploading, processing, and managing images within the application all at once, with the entire goal of this algorithm is to provide a simple yet effective security measure for user verification.

3.3 Image Handler Flowchart

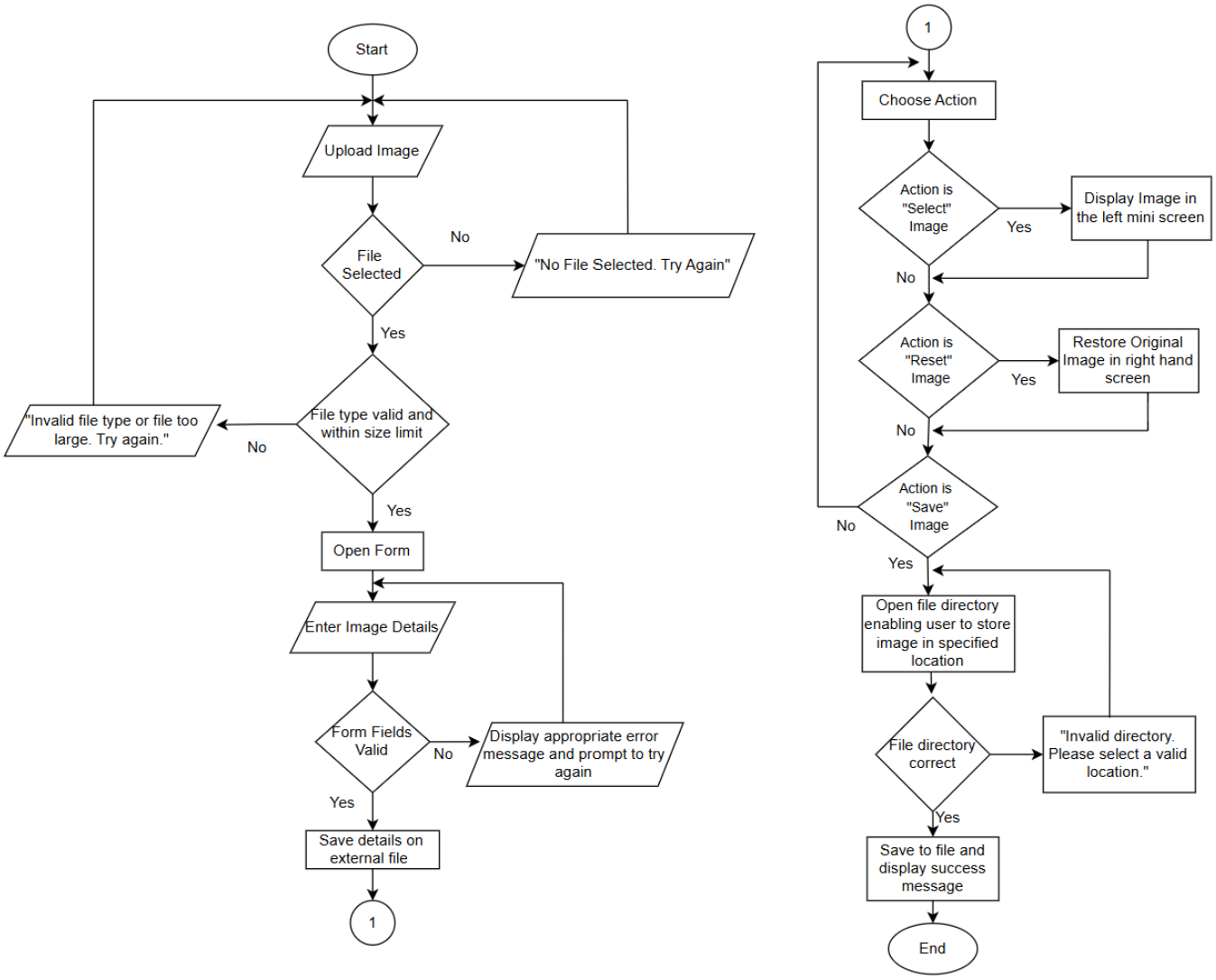

Figure 3: Image Handler Flowchart

Before accessing additional functionalities like **Reset, Select,** and **Save**, the user must first successfully complete the image upload and form handling checks. This is to ensure the software is not used for a purpose it was never built for e.g processing videos. In the actual implementation, the program will allow dynamic access after validation, but for the sake of simplicity here we have showed the transition in a simplified fashion. This simplified representation has the additional benefit of showing a structured flow for the program, enabling the software to be tailored and used in the future for more complex uses such as batch processing.

3.4 Image Processing Flowchart

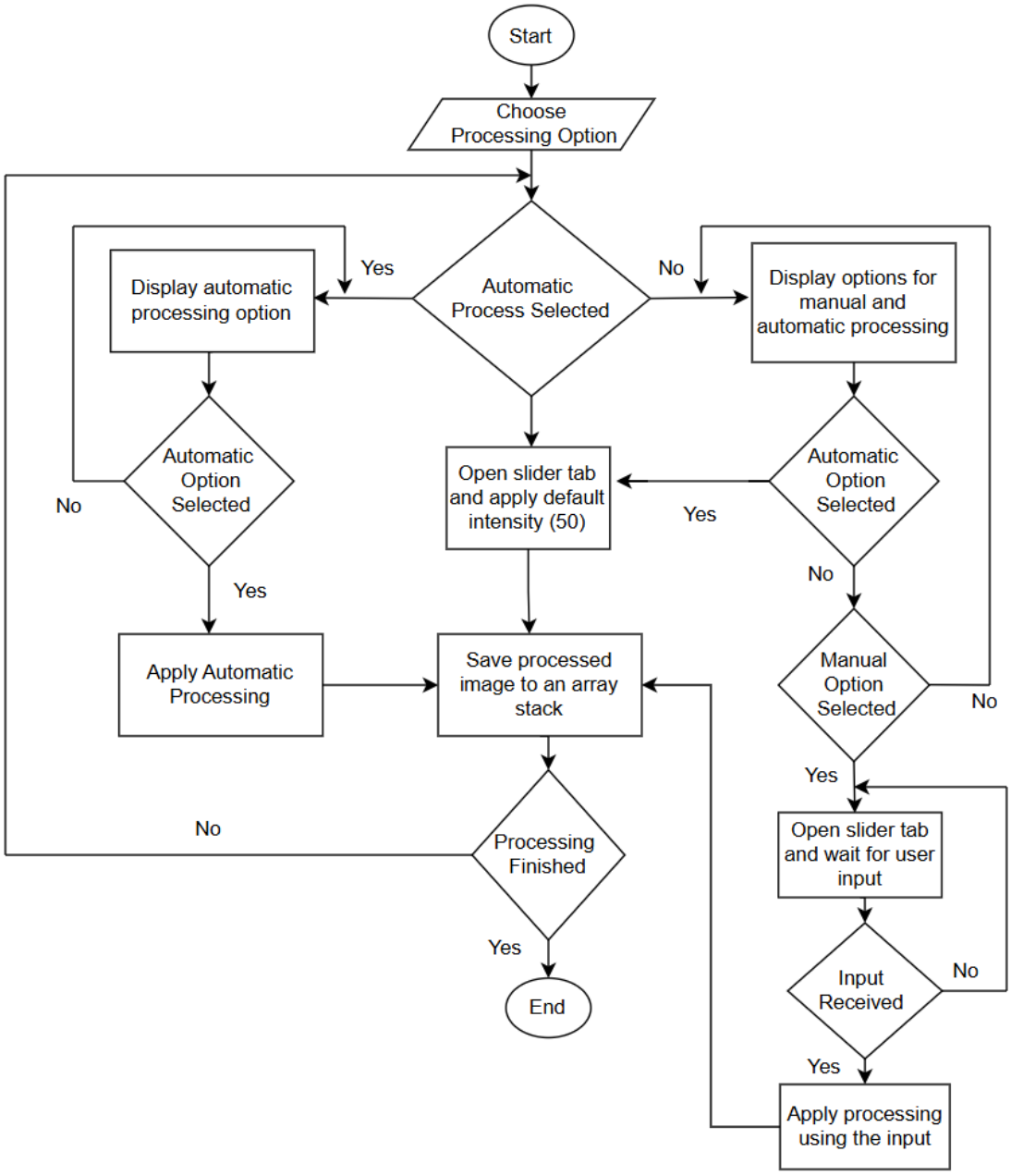

Figure 4: Image Processing Flowchart

The flowchart above simplifies the actual code implementation I am to have, since currently its focusing solely on image processing functionalities.In reality, the user will be able to use both Image Handler and Image Processing functionalities simultaneously. For example, the user will be able to apply inversion and brightness before downloading the processed image. This is because I will ensure the 2 classes are dynamically integrated with each other, enabling the user to smoothly transition from the subroutine of one class to the subroutine of the other class.

4 Final Product

In this section, I will show the final product of my implemented program alongside core logic and some error handling . I won't go too in depth with the error handling, since this sections main focus involves showing the interface of the different classes alongside the main algorithms associated with them. A more comprehensive set of error handling tests will be carried out in the test section.

4.1 PIN Entry System

Figure 5: Final PIN Entry System Interface

```python
    # PIN entry field
    self.pin_entry = ctk.CTkEntry(self.window, show="*", font=("Helvetica", 16), justify="center", width=150)
    self.pin_entry.pack(pady=10)

def enter_digit(self, digit):
    """Append a digit to the PIN entry field."""
    current_pin = self.pin_entry.get()
    if len(current_pin) < 4:
        self.pin_entry.insert(ctk.END, str(digit))

def clear_pin(self):
    """Clear the PIN entry field."""
    self.pin_entry.delete(0, ctk.END)

def check_pin(self):
    """Validate the entered PIN."""
    entered_pin = self.pin_entry.get()
    if entered_pin == self.pin:
        self.result_label.configure(
            text="PIN Accepted",
            text_color="green",
            font=("Arial", 16, "bold")
        )
        self.window.destroy()  # Close the PIN entry window
        self.root.deiconify()  # Restore the main root window
        self.app_callback()  # Initialize the main application
    else:
        self.result_label.configure(
            text="Invalid PIN",
            text_color="red",
            font=("Arial", 20, "bold")  # Larger and bolder font for error message
        )
        self.clear_pin()
```

Figure 6: PIN Entry System Logic

The final GUI is designed with a clean, modern aesthetic to enhance visual appeal and ensure a user-friendly experience. Users are prompted to enter the correct PIN (set to "1234") to access the ImageCraft software. For security, user input is masked and validated. If the PIN is correct, the PIN entry interface closes, and the ImageCraft GUI launches. Otherwise, an appropriate error message is displayed. This aproach has also the added benefit of keeping the code clean and minimalistic with checks since all the code has to check for is numbers and not letters. There is also the added benefit of not having to worry about length checks since the program only allows for a specefic number of charecters to be inputted. Below we can see the output to an invalid input

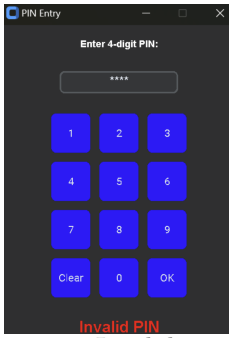

Figure 7: Invalid output

4.2 Form GUI

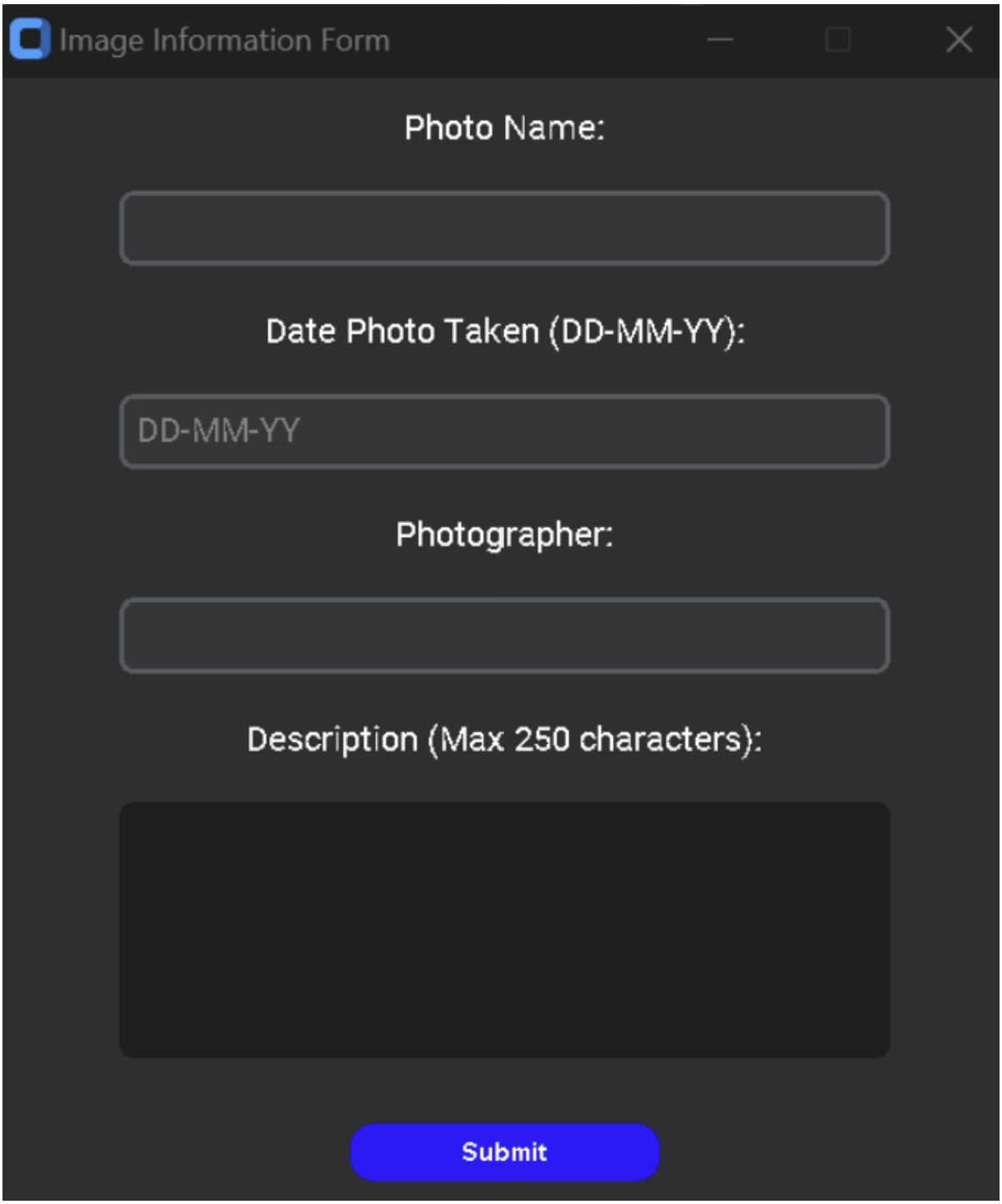

Figure 8: Form Interface

Users typically begin by uploading an image from their computer directory, followed by filling out a form with details about the image. Once the fields are validated, the image is uploaded to the Right Preview panel, ready for processing.

The code below shows some of the validation the program checks for:

```python
def validate_and_save_form(self, photo_name, date_taken, photographer, description, popup):
    """Validate form inputs and save data."""
    errors = []

    # 1. Validate Photo Name (Non-empty, only letters, numbers, spaces, and dashes)
    if not photo_name.strip():
        errors.append("Photo Name cannot be empty.")
    if not re.match(r'^[a-zA-Z0-9 \-]+$', photo_name.strip()):
        errors.append("Photo Name can only contain letters, numbers, spaces, and dashes.")
    if len(photo_name.strip()) > 50:
        errors.append("Photo Name must not exceed 50 characters.")

    # 2. Normalize date format to DD-MM-YY (convert YYYY to YY if entered)
    date_taken = date_taken.strip().replace("/", "-").replace(".", "-")  # Replace separators with "-"

    # Convert 4-digit year (YYYY) to 2-digit (YY) if valid
    match = re.match(r'^(\d{2})-\d{2})-(\d{4})$', date_taken)
    if match:
        day_month = match.group(1)
        year = match.group(2)
        if int(year) < 1900:
            errors.append("Date must be after 01-01-1900.")
        else:
            date_taken = f"{day_month}-{year[-2:]}"  # Convert to two-digit format

    # 3. Validate Date (Format DD-MM-YY, valid calendar date, not in the future, after 1900)
    date_pattern = r"^\d{2}-\d{2}-\d{2}$"  # Expecting DD-MM-YY format
    if not re.match(date_pattern, date_taken):
        errors.append("Date must be in the format DD-MM-YY and contain only numbers.")
    else:
        try:
            entered_date = datetime.strptime(date_taken, "%d-%m-%y")

            # Ensure proper interpretation of past dates (e.g., 45 = 1945)
            current_year = datetime.today().year % 100  # Get last two digits of the current year
            century = 2000 if int(date_taken[-2:]) <= current_year else 1900
            entered_date = entered_date.replace(year=century + int(date_taken[-2:]))

            # Ensure the date is not in the future
            if entered_date > datetime.today():
                errors.append("Date cannot be in the future.")

            # Ensure the date is not before 1900
            min_year = 1900
            if entered_date.year < min_year:
                errors.append(f"Date must be after 01-01-{min_year}.")

        except ValueError:
            errors.append("Invalid date. Ensure the day and month are correct.")
```

Figure 9: First section of checks

```python
# 4. Validate Photographer Name (Non-empty, only letters and spaces)
if not photographer.strip():
    errors.append("Photographer cannot be empty.")
if not re.match(r'^[A-Za-z ]+$', photographer.strip()):
    errors.append("Photographer name should only contain letters and spaces.")

# 5. Check for duplicate photo name
folder_name = "FormData"
file_name = "form_data.txt"
file_path = os.path.join(folder_name, file_name)

if os.path.exists(file_path):
    with open(file_path, "r") as file:
        if f"Photo Name: {photo_name}" in file.read():
            errors.append("A photo with this name has already been submitted.")

# 6. Ensure no fields contain only spaces except description (optional)
required_fields = [photo_name, date_taken, photographer]
if not all(field.strip() for field in required_fields):
    errors.append("Photo Name, Date Taken, and Photographer fields must be filled out correctly.")

# 7. Optional description validation (if provided)
if description.strip():
    if len(description.strip()) > 250:
        errors.append("Description must not exceed 250 characters.")

# Handle errors or save data
if errors:
    self.show_form_error_popup("\n".join(errors))
else:
    # Save data to a file
    if not os.path.exists(folder_name):
        os.makedirs(folder_name)

    with open(file_path, "a") as file:
        file.write(f"Photo Name: {photo_name}\n")
        file.write(f"Date Taken: {date_taken}\n")
        file.write(f"Photographer: {photographer}\n")
        file.write(f"Description: {description if description.strip() else 'N/A'}\n")
        file.write("-" * 40 + "\n")

    # Update the right-hand side preview
    self.display_right_preview()

    messagebox.showinfo("Success", "Form submitted successfully!")
    popup.destroy()
```

Figure 10: Second section of checks

Error Testing 1: when photo field is validated wrong

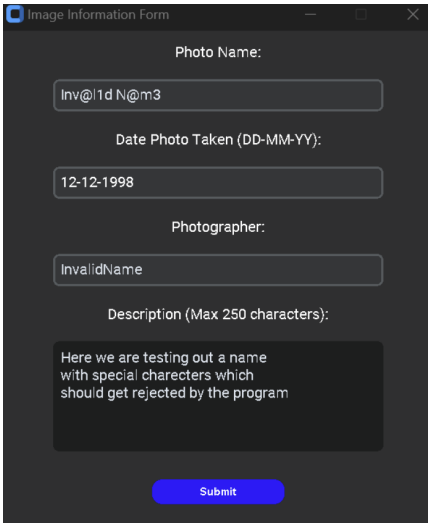

Figure 11: Invalid Photo Name Input

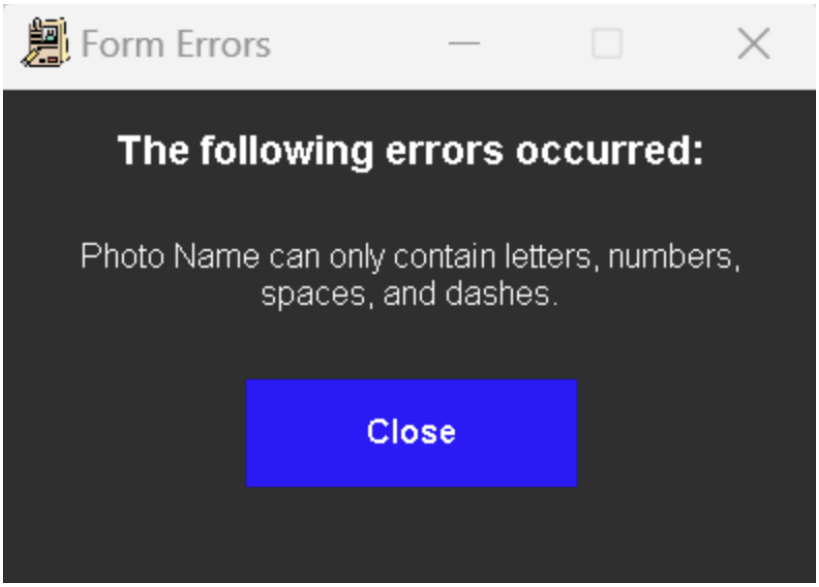

Figure 12: Invalid Photo Name Output

Error Testing 2: when date input is wrong

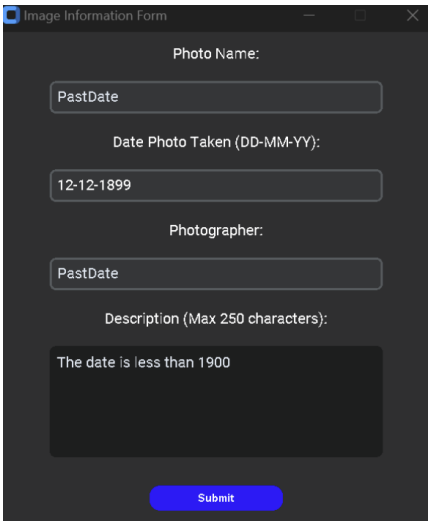

Figure 13: Invalid Date Input

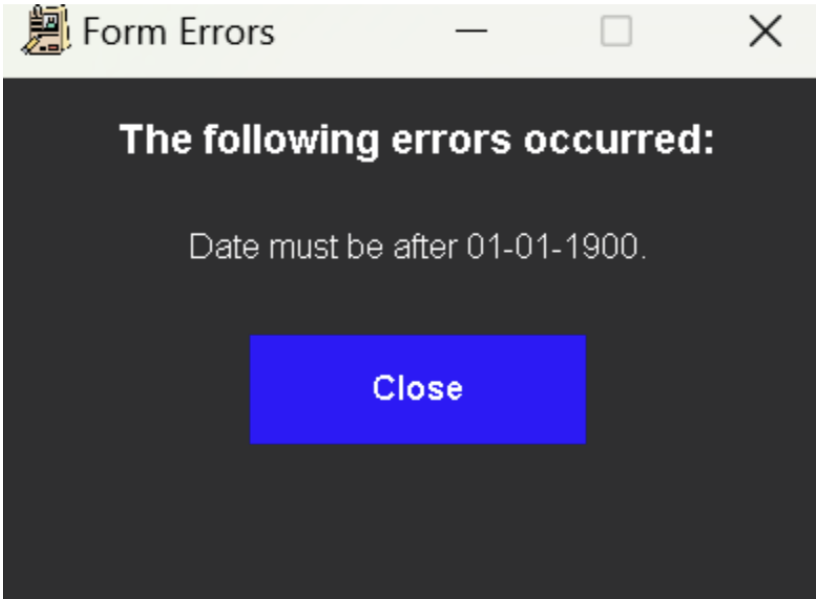

Figure 14: Invalid Date Output

4.3 Main GUI

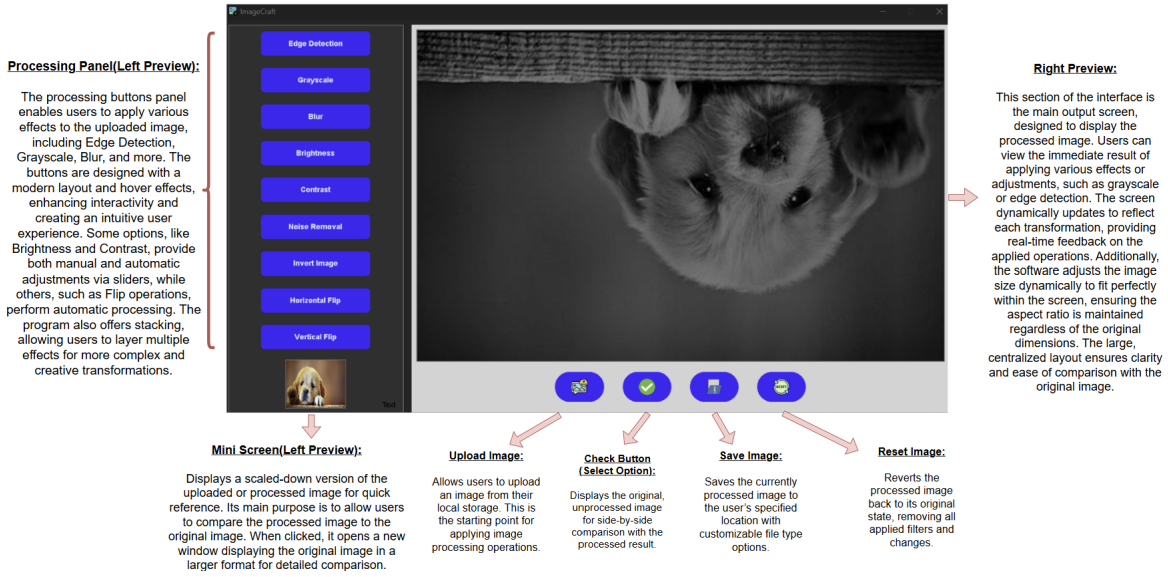

Figure 15: Form Interface

Once inside the main application, users can begin processing images and use additional functionalities like 'Select Option' and 'Reset Image'. However, before any processing the program will first validate if the image selected matches the set criteria once users click on the upload icon. This is done through a variety of error handling algorithms, as shown below:

```
def upload_image(self):
    """Upload an image and open the form popup.
    Signature checking is implemented for additional security to verify the actual content of the file.
    """
    file_path = filedialog.askopenfilename(
        title="Select an Image File",
        filetypes=[("Image Files", "*.png;*.jpg;*.jpeg;*.bmp;*.gif")]  # Basic validation: Restricts file selection by extension
    )

    if file_path:
        try:
            # Check if the file size is safe
            if not self.is_file_size_safe(file_path):
                messagebox.showerror("Error", "File size exceeds 5 MB. Please choose a smaller file.")
                return  # Exit the function if the file is too large

            # Advanced validation: Signature checking to prevent spoofed extensions
            with open(file_path, 'rb') as file:
                file_signature = file.read(8)  # Read first 8 bytes

            # Common image signatures
            valid_signatures = {
                b'\x89PNG\r\n\x1a\n': 'PNG',  # PNG
                b'BM': 'BMP',  # BMP
                b'GIF87a': 'GIF',  # GIF
                b'GIF89a': 'GIF'  # GIF
            }

            # Proper JPEG Handling: All JPEGs start with \xff\xd8
            if not (file_signature.startswith(b'\xff\xd8') or any(file_signature.startswith(sig) for sig in valid_signatures.keys())):
                raise ValueError("Invalid file type. Please select a valid image.")

            # Attempt to open the image (final corruption check)
            image = Image.open(file_path)
            self.app.uploaded_photo = image
            self.app.original_image = image.copy()
            self.open_form_popup()

        except Exception as e:
            messagebox.showerror("Error", str(e))
    else:
        messagebox.showwarning("No File Selected", "You must select a file.")
```

Figure 16: Upload Image Validation Code

Error Testing 1: when the file is too large

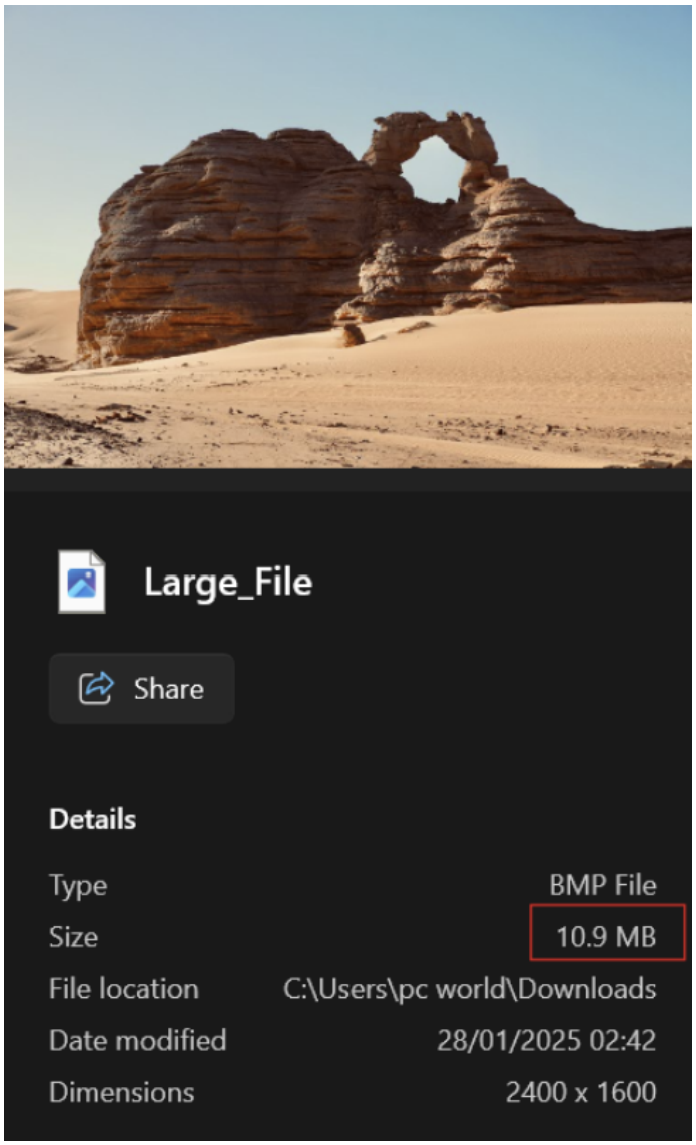

Figure 17: Large File Example

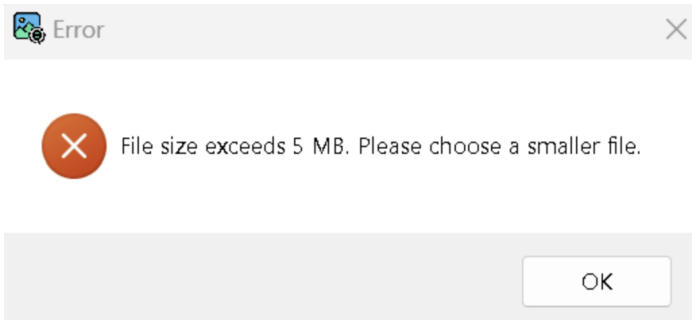

Figure 18: Output

Error Testing 2: when the file is invalid

Our program's basic validation ensures that only allowed file types are visible in the file selection dialog, preventing users from selecting invalid file formats. In this instance, we will disable this basic validation to demonstrate how signature checking independently prevents unauthorised file uploads incase someone bypassed the first check.

Change:

```
#file_path = filedialog.askopenfilename(
    #title="Select an Image File",
    #filetypes=[("Image Files", "*.png;*.jpg;*.jpeg;*.bmp;*.gif")]
#)

file_path = filedialog.askopenfilename(
    title="Select an Image File",
    filetypes=[('All Files', '*.*')]  # Changed to [('All Files', '*.*')] for testing purposes.
)
```

Figure 19: Basic Check Disabled

Testing Item:

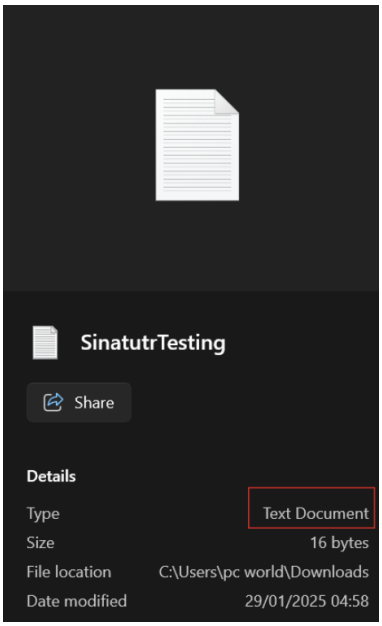

Figure 20: Testing a txt filetype

Output:

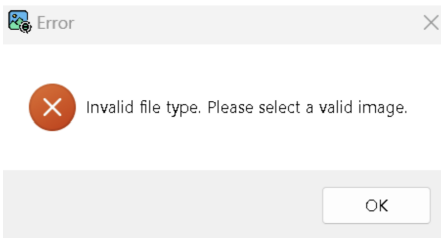

Figure 21: Output for signature testing

If upload image checks are all successeful, the user will then be able to start processing. This involves selecting their button of choice from the processing panel. If an automatic processing button (the bottom three buttons) is chosen, the process will execute instantly without further user input. However, if the other buttons are selected the program will prompt a window offering both automatic and manual processing options. If automatic is selected the program will automatically perform the process with a preset intensity (slider value: 50), else the user will be able to select the process intensity. Below is an example showing how it would look like if Greyscale got selected:

Relevant Code:

```
# Button-linked functions
def open_grayscale_window(self):
    """Open slider for grayscale intensity."""
    self.open_slider_window("Grayscale", self.apply_grayscale_with_intensity)
```

Figure 22: Greyscale code snippet 1

```
# Apply button
apply_button = ctk.CTkButton(
    self.slider_window,
    text="Apply",
    command=lambda: [self.apply_processing_with_intensity(process_function, intensity.get()), self.slider_window.destroy()],
    fg_color="blue",
    hover_color="lightblue",
    font=("Arial", 12)
)
apply_button.pack(pady=10)

self.slider_window.protocol("WM_DELETE_WINDOW", close_window)
```

Figure 23: Greyscale code snippet 2

```
def apply_processing_with_intensity(self, process_function, intensity):
    """Apply processing with intensity and update the processed image."""
    if self.app.uploaded_photo:
        processed_image = process_function(self.app.uploaded_photo, intensity)
        self.app.uploaded_photo = processed_image
        self.app.image_handler.display_right_preview()
        messagebox.showinfo("Success", f"{process_function.__name__.replace('_', ' ').capitalize()} applied with intensity {intensity}!")
    else:
        messagebox.showwarning("No Image", "Please upload an image first.")

def apply_grayscale_with_intensity(self, image, intensity):
    """Custom grayscale processing based on intensity."""
    grayscale_image = image.convert("L")  # Convert to grayscale
    return grayscale_image.point(lambda p: p * (intensity / 100))  # Adjust pixel intensity
```

Figure 24: Greyscale code snippet 3

Greyscale window output:

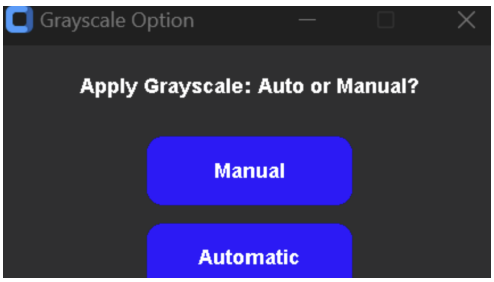

Figure 25: Greyscale window

Example where automatic option was pressed:

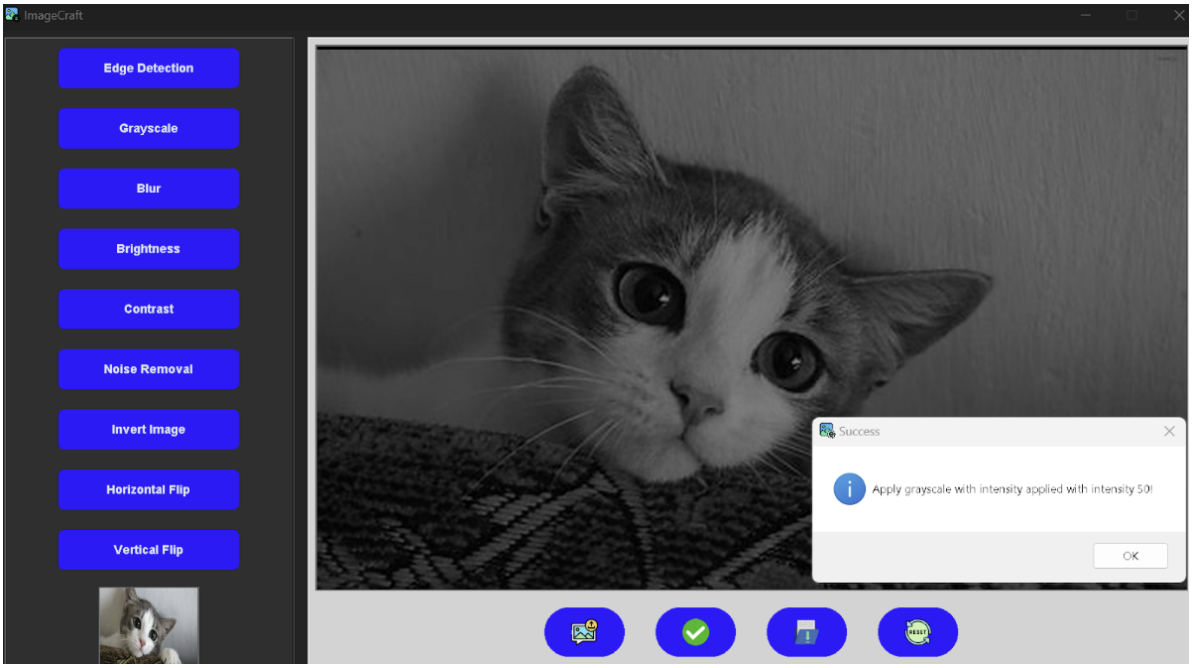

Figure 26: Automatic Processing

Example where automatic option was pressed:

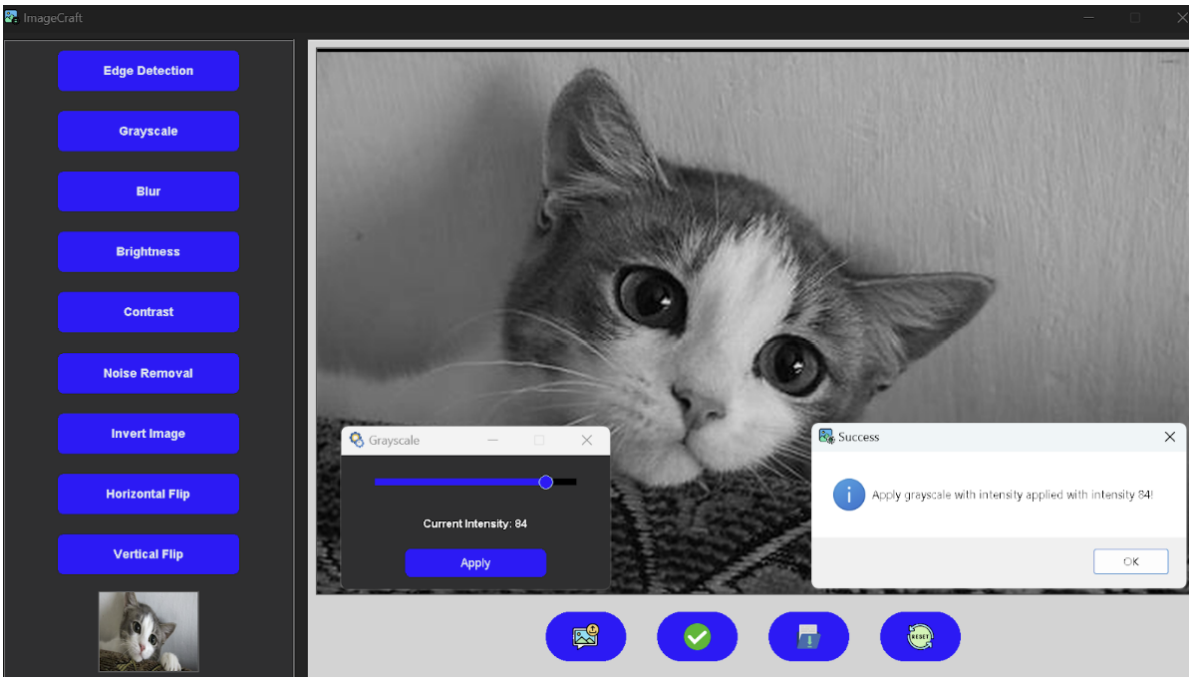

Figure 27: Manual Processing

5 Development Challenges and Solutions

This section will cover the development of ImageCraft, briefly highlighting the key challenges that were encountered along with the solutions implemented to address the problems.

5.1 Repeating Code

While implementing the Contrast feature in the Image Processing Tool, I noticed significant redundancy across multiple image adjustment functionalities, such as Brightness, Grayscale, and Blur. Each function required similar code blocks for setting up sliders, applying intensity adjustments, and managing UI elements, leading to code duplication. This not only made the codebase harder to maintain but also violated the DRY (Don't Repeat Yourself) principle.

```
def apply_inversion(self):
    """Apply image inversion and update the GUI inefficiently."""
    if self.app.original_image:
        self.app.uploaded_photo = self.app.original_image.copy()



        inverted_image = self.app.uploaded_photo
        inverted_image = inverted_image.convert("RGB")

        self.app.uploaded_photo = inverted_image



        self.app.image_handler.display_right_preview()
        self.app.image_handler.display_right_preview()

        messagebox.showinfo("Success", "The image has been inverted!")
    else:
        messagebox.showwarning("No Image", "Please upload an image first.")
```

Figure 28: Inefficient Code

To address this, I refactored the code by abstracting common functionalities into reusable methods, significantly improving code efficiency and maintainability.

```
def apply_inversion(self):
    """Apply image inversion and update the GUI."""
    if self.app.uploaded_photo:
        inverted_image = self.apply_image_inversion(self.app.uploaded_photo)  # Apply inversio
        self.app.uploaded_photo = inverted_image  # Update with the new processed image
        self.app.image_handler.display_right_preview()  # Display the processed image
        messagebox.showinfo("Success", "The image has been inverted!")
    else:
        messagebox.showwarning("No Image", "Please upload an image first.")
```

Figure 29: Solution Code

This refactoring reduced repetition, streamlined future feature additions, and minimized potential errors, ultimately enhancing the software's scalability and robustness.

Why it Works:

apply_processing_with_intensity:

1. Changed `self.app.original_image` to `self.app.uploaded_photo` to ensure effects are applied on the last processed image.
2. Updates `self.app.uploaded_photo` with the new processed image, enabling stacking of effects.

apply_inversion:

1. Same change as above to use `self.app.uploaded_photo` instead of `self.app.original_image`.
2. Ensures that inversion is applied on the latest version of the image rather than resetting to the original.

5.2 Inconsistent Date Validation and Year Handling

During the development of the form validation feature, an issue was encountered where the application incorrectly accepted invalid date formats and unrealistic years (e.g., 12-12-1234).

```
if "-" not in date_taken or len(date_taken) < 6:
    errors.append("Invalid date format. Please enter in DD-MM-YY format.")
else:
    try:
        entered_date = datetime.strptime(date_taken, "%d-%m-%y")

        if entered_date.year < 1900:
            errors.append("Invalid year. Enter a valid date.")

    except ValueError:
        errors.append("Invalid date entry.")
```

Figure 30: Wrong Validation code

This issue occurred due to a lack of strict validation and incorrect handling of year conversions, leading to potential data inconsistencies.

```
# 3. Validate Date (Format DD-MM-YY, valid calendar date, not in the future, after 1900)
date_pattern = r"^\d{2}-\d{2}-\d{2}$"  # Expecting DD-MM-YY format
if not re.match(date_pattern, date_taken):
    errors.append("Date must be in the format DD-MM-YY and contain only numbers.")
else:
    try:
        entered_date = datetime.strptime(date_taken, "%d-%m-%y")

        # Ensure proper interpretation of past dates (e.g., 45 = 1945)
        current_year = datetime.today().year % 100  # Get last two digits of the current year
        century = 2000 if int(date_taken[-2:]) <= current_year else 1900
        entered_date = entered_date.replace(year=century + int(date_taken[-2:]))

        # Ensure the date is not in the future
        if entered_date > datetime.today():
            errors.append("Date cannot be in the future.")

        # Ensure the date is not before 1900
        min_year = 1900
        if entered_date.year < min_year:
            errors.append(f"Date must be after 01-01-{min_year}.")

    except ValueError:
        errors.append("Invalid date. Ensure the day and month are correct.")
```

Figure 31: Right Validation code

Why it Works:

Strict Format Enforcement:

1. Previously, date validation only checked for hyphens and length, allowing incorrect formats.
2. Now, it uses regex to strictly enforce the DD-MM-YY format.

Correct Year Interpretation:

1. Previously, the code relied on Python's default handling of two-digit years, leading to incorrect century assignments.
2. Now, the century is dynamically determined based on the current year, ensuring correct classification between the 1900s and 2000s.

Valid Date Constraints:

1. Previously, future dates were not restricted, allowing unrealistic entries.
2. Now, a check ensures that no future dates can be entered.

6 Test Plan and Execution

Following the completion of the design and development phases, a comprehensive test plan was created to ensure that all functionalities of the application were working as intended. The iterative testing approach was adopted throughout the development process, allowing for continuous identification and resolution of issues. Upon finalizing development, all test cases were executed again to verify the program's stability and robustness.

6.1 PIN Entry System

Test Case	Test Type	Input	Justification	Expected Outcome	Actual Outcome	Pass/Fail
1. Valid PIN	Normal	1234	Represents a valid 4-digit PIN, which follows the required format for normal operation.	Opens Image Craft GUI	Figure 32	Pass
2. Invalid PIN	Invalid	5678	Tests incorrect PIN entry to check if the system rejects unauthorised access.	Displays error message “Invalid PIN”	Figure 33	Pass
3. Empty Input	Erroneous	""	Tests blank input to ensure the system replies with correct error handling message.	Displays error message ”Invalid PIN”	Figure 33	Pass
4. Too Short PIN	Invalid Extreme	12	Tests a boundary case where the PIN is shorter than the required length.	Displays error message “Invalid PIN”	expected outcome	Pass
5. Too Long PIN	Invalid Extreme	12345	Tests a boundary case where the PIN is longer than the required length.	Does not allow the user to enter the 5th number	The 5th digit is blocked from being entered	Pass
6. Clear Button Functionality	Normal	Input 123, then click Clear	Tests if the clear button works as intended.	The PIN entry screen clears	Figure 34	Pass

Table 2: PIN Entry System

6.2 Screenshots for Pin Entry Test Table

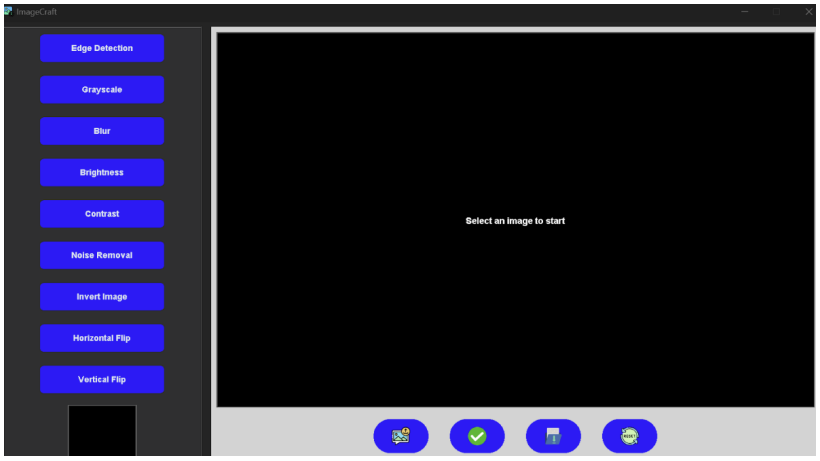

Figure 32: Outcome to Test Case 1

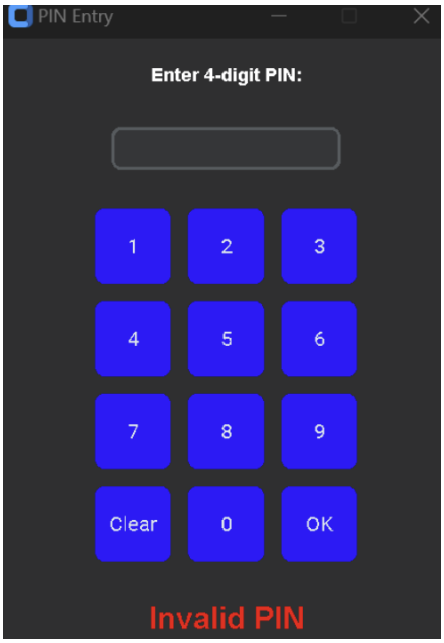

Figure 33: Outcome to Test Case 2

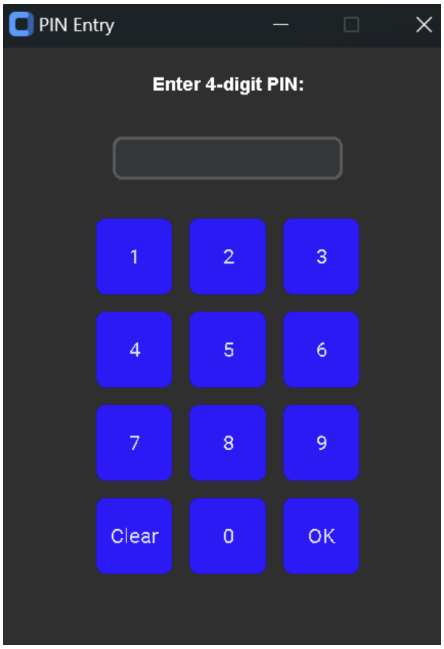

Figure 34: Outcome to Test Case 6

6.3 Upload Image

Test Case	Test Type	Input (File Type)	Justification	Expected Outcome	Actual Outcome	Pass/Fail
1. Valid File (PNG, JPG, JPEG, BMP, GIF)	Normal	PNG, JPG, JPEG, BMP, GIF	Ensures the program successfully uploads all valid image formats specified in the brief.	File is uploaded and proceeds to the form popup.	expected outcome	Pass
2. File larger than 5 MB	Extreme Boundary	PNG, JPG, BMP, GIF	Tests if the program correctly prevents the upload of files exceeding the 5 MB size limit.	Displays error: "File exceeds 5 MB. Please choose a smaller file."	Figure 18.	Pass
3. Unsupported file type	Invalid	TXT	Ensures that unsupported file types (e.g., .txt files) are not even shown in the file dialog, as per implementation.	File does not appear in the file dialog.	Expected Outcome	Pass
4. No file selected	Erroneous	N/A	Checks behavior when the user selects no file.	Displays error: "You must select a file."	Figure 35	Pass
5. File path with special characters	Boundary	PNG	Ensures file paths with special characters do not cause issues.	File is seen in the directory.	Figure 36	Pass

Table 3: Upload Image Test Cases

6.4 Screenshots for Upload Image Test Table

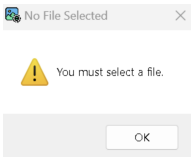

Figure 35: Outcome to Test Case 4

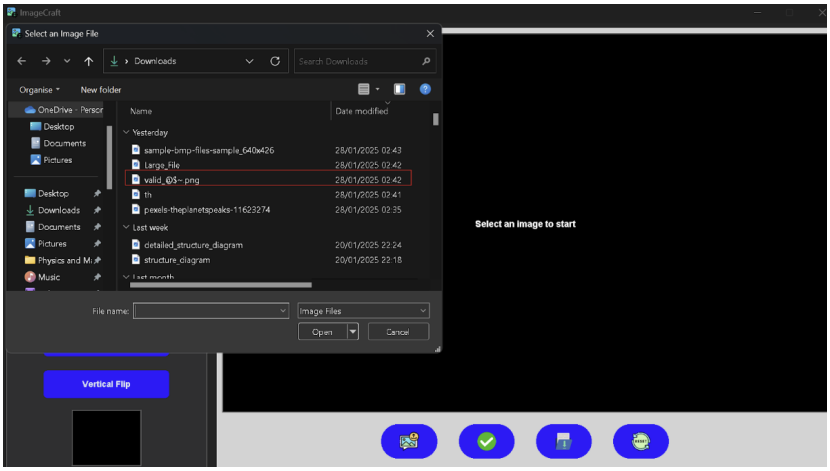

Figure 36: Outcome to Test Case 5

6.5 Form Handling

Test Case	Test Type	Input	Expected Outcome	Actual Outcome	Pass/Fail
1. Valid Form Submission	Normal	Photo Name: Mine Date: 12-01-22 Photographer: John Description: Sample	Form is saved, success message displayed, popup closes.	Figure 37	Pass
2. Empty Photo Name	Invalid	""	Error: "Photo Name cannot be empty."	Figure 38.	Pass
3. Invalid Characters in Name	Invalid	My@Photo	Error: "Photo Name can only contain letters, numbers, spaces, and dashes."	Figure 39	Pass
4. Name Exceeds Character Limit	Boundary	A (repeated 51 times)	Error: "Photo Name must not exceed 50 characters."	Figure 40	Pass
5. Duplicate Photo Name	Invalid	Mine	Error: "A photo with this name has already been submitted."	Figure 41.	Pass
6. Invalid Date Format	Invalid	01-2023	Error: "Date must be in the format DD-MM-YY and contain only numbers."	Figure 42.	Pass
7. Date in the Future	Invalid	12-12-25	Error: "Date cannot be in the future."	Figure 43.	Pass
8. Date Before 1900	Invalid	12-12-1889	Error: "Date must be after 01-01-1900."	Figure 44.	Pass
9. Description ¿250	Invalid	A*251	Description must not exceed 250 characters"	Figure 45.	Pass

Table 4: Form Handling Test Cases

6.6 Screenshots Form Handling Test Table

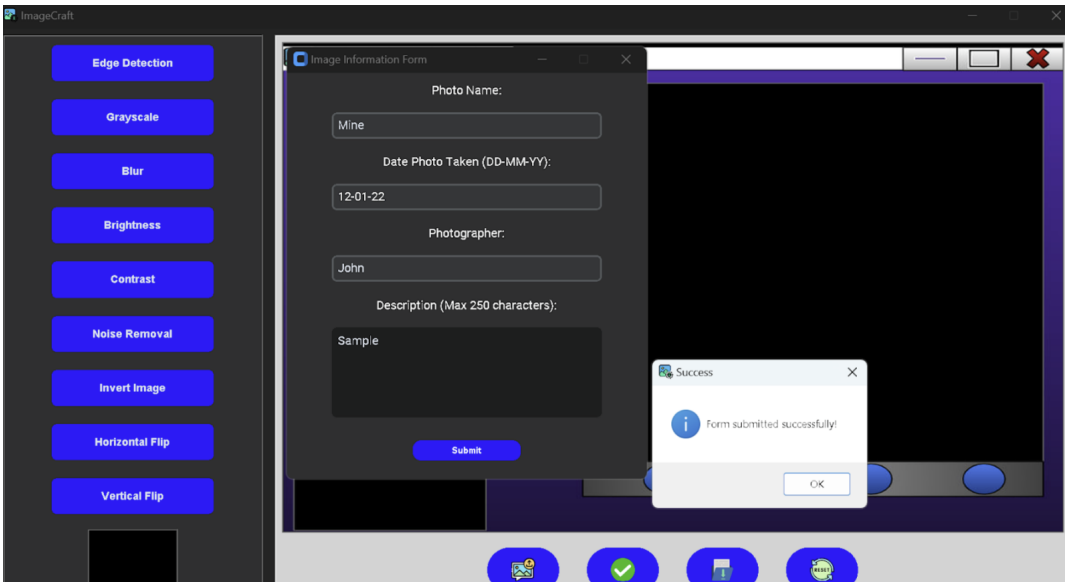

Figure 37: Outcome to Test Case 1

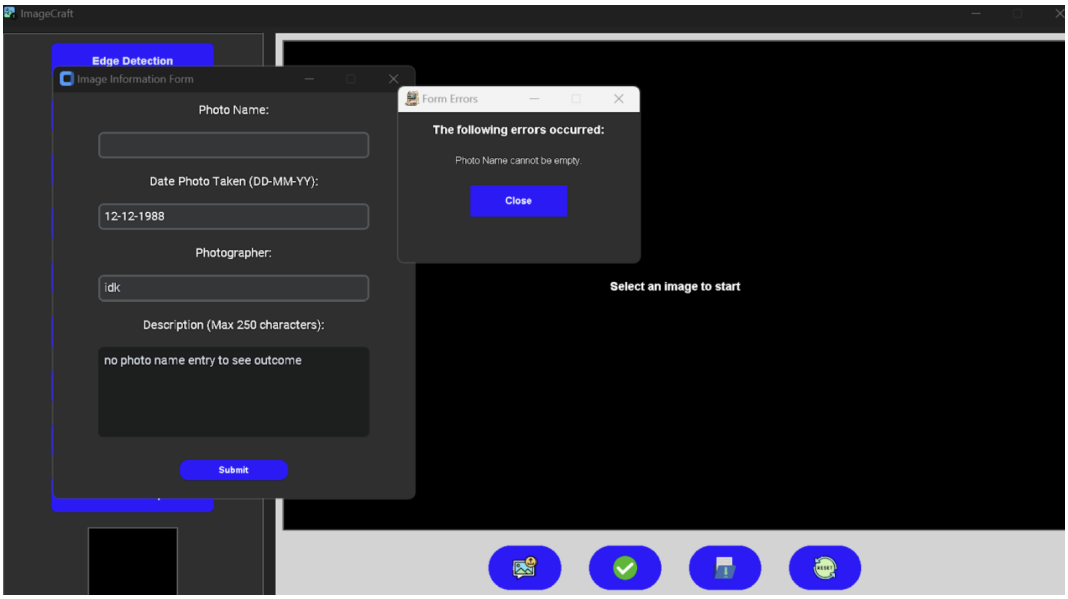

Figure 38: Outcome to Test Case 2

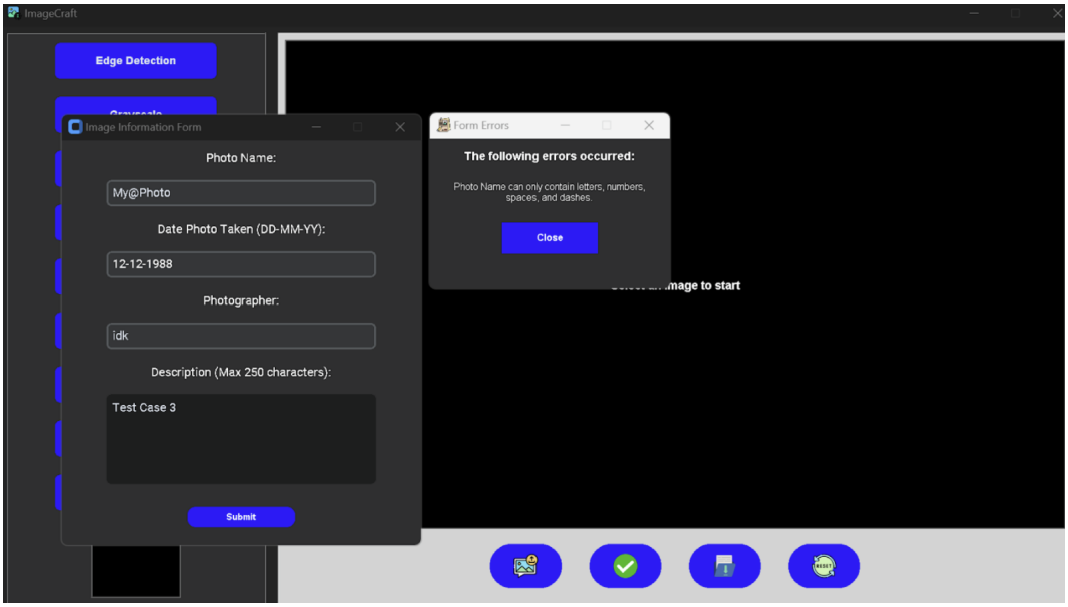

Figure 39: Outcome to Test Case 3

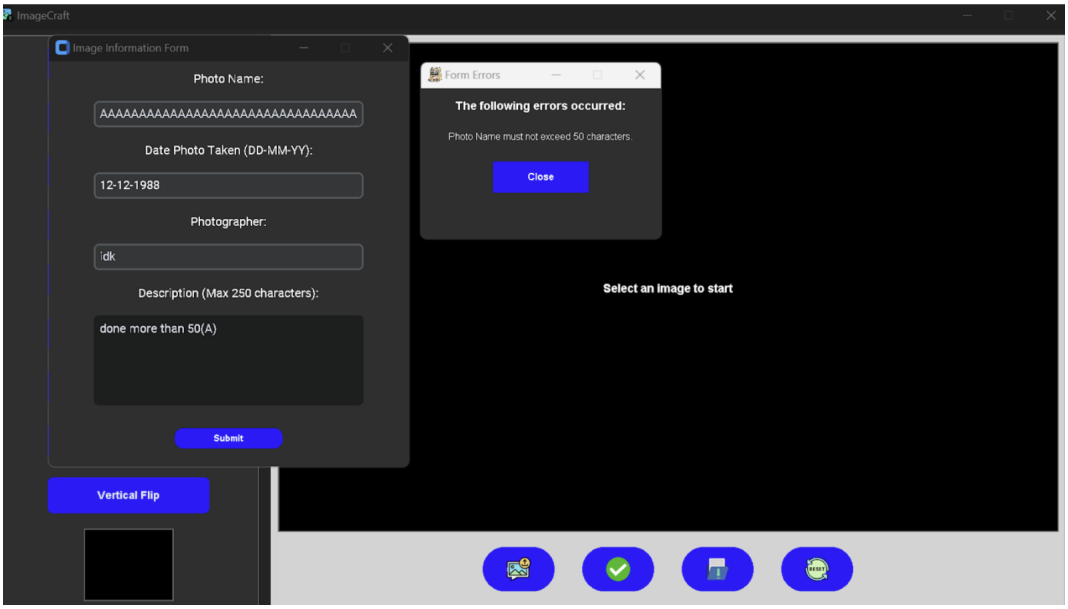

Figure 40: Outcome to Test Case 4

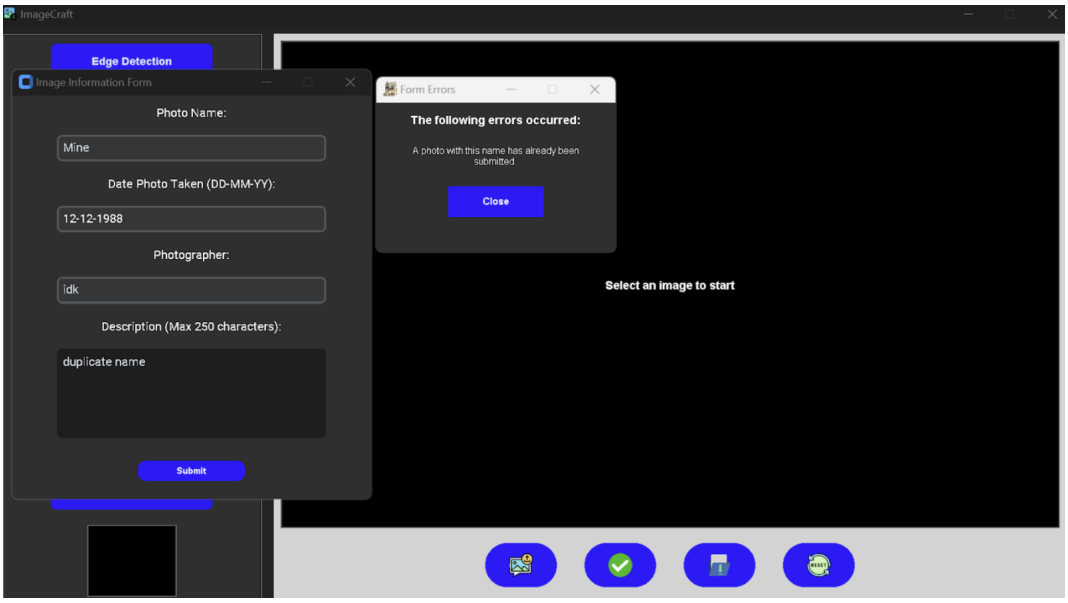

Figure 41: Outcome to Test Case 5

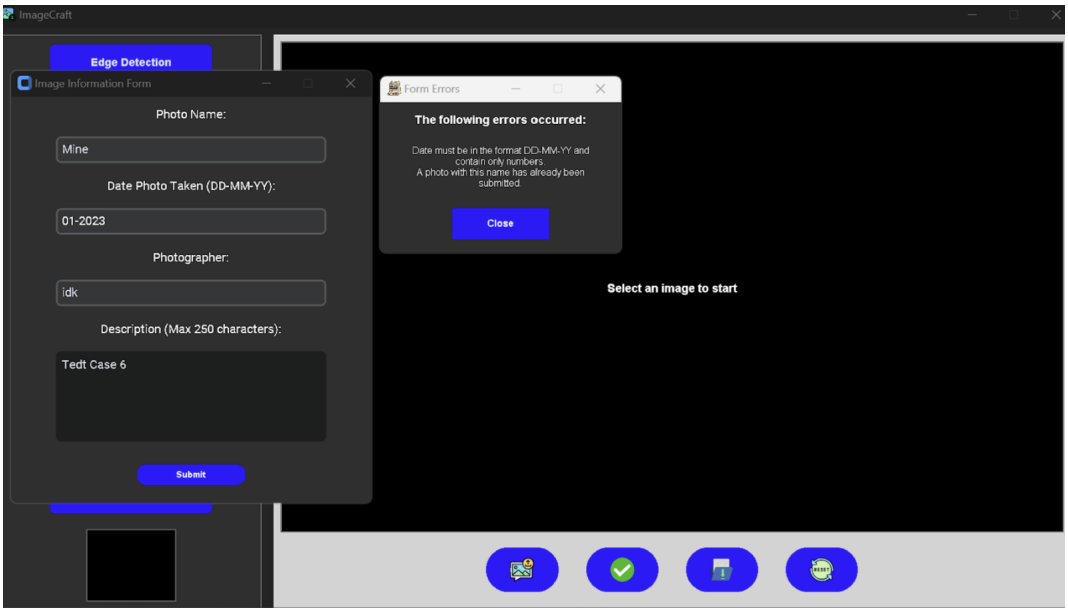

Figure 42: Outcome to Test Case 6

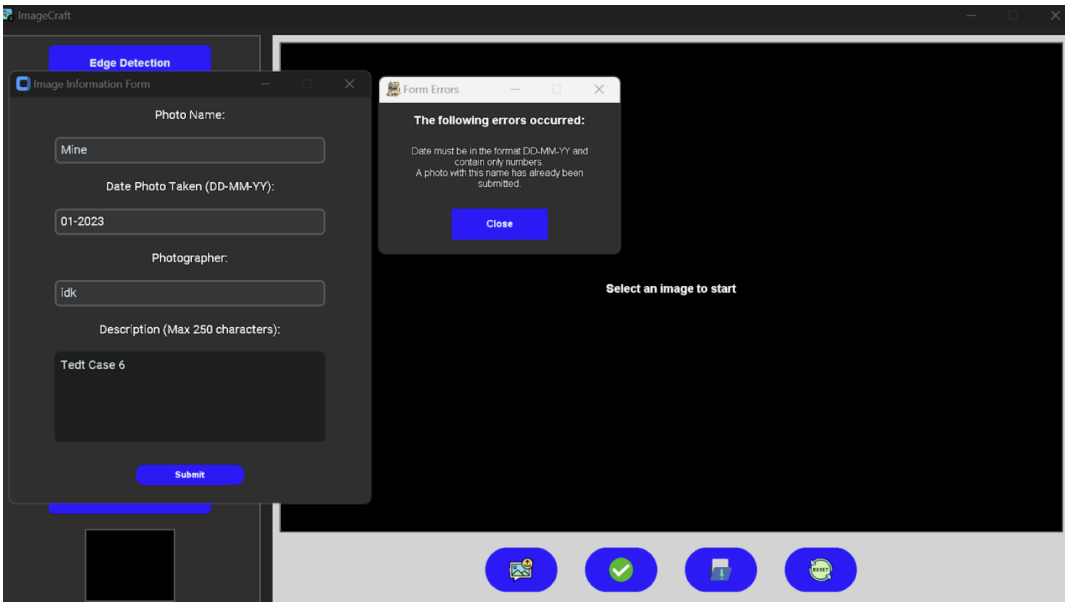

Figure 43: Outcome to Test Case 7

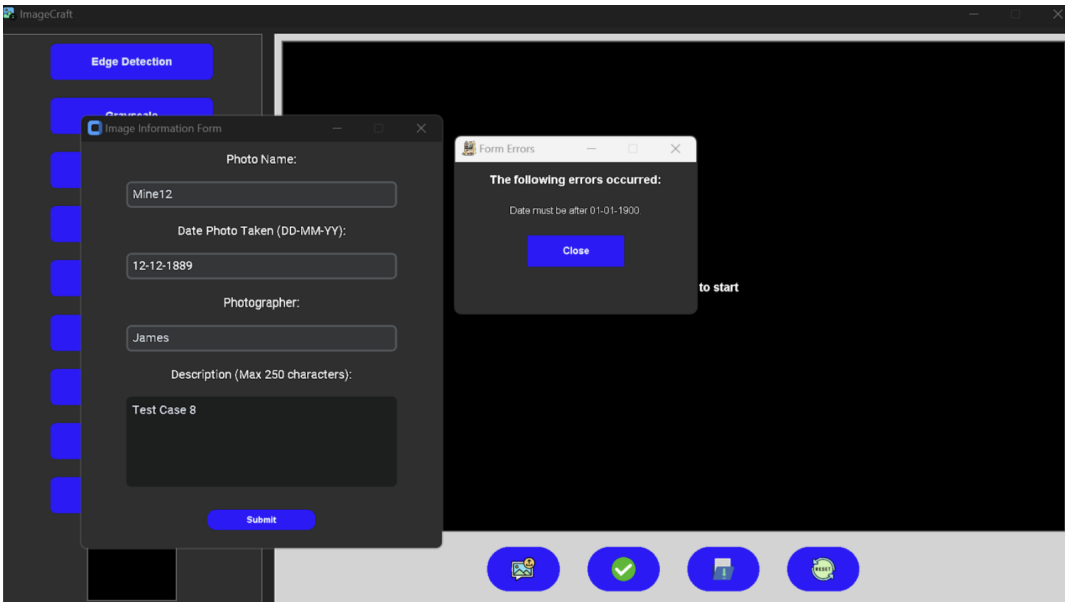

Figure 44: Outcome to Test Case 8

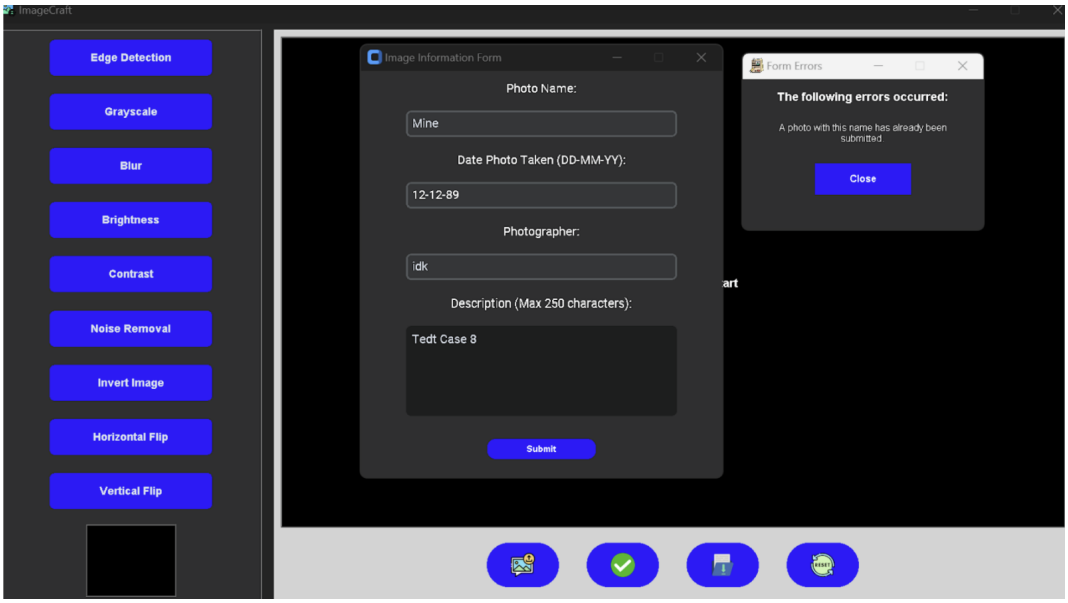

Figure 45: Outcome to Test Case 9

6.7 Image Processing

Test Case	Test Type	Input	Justification	Expected Outcome	Actual Outcome	Pass/Fail
1. Apply Grayscale (Automatic)	Normal	Apply grayscale using automatic mode	Ensures that automatic grayscale processing applies the effect correctly without user input.	Image is converted to grayscale.	Figure 26	Pass
2. Apply Grayscale (Manual)	Normal	Apply grayscale using manual mode	Tests the manual slider functionality for grayscale intensity.	Image is converted to grayscale with adjustable intensity.	Figure 27	Pass
3. Apply Grayscale + Invert Image	Normal	Apply grayscale manually, then invert colors	Verifies that multiple processing effects can be stacked correctly.	Image is first converted to grayscale, then colors are inverted.	Figure 15	Pass
4. Try to Process Image Without Uploading	Invalid	Attempt to apply any processing function without an uploaded image	Ensures the system prevents processing if no image is uploaded.	Displays error: "Please upload an image first."	figure 46	Pass

Table 5: Image Processing Test Cases

6.8 Screenshots Image Processing Test Cases

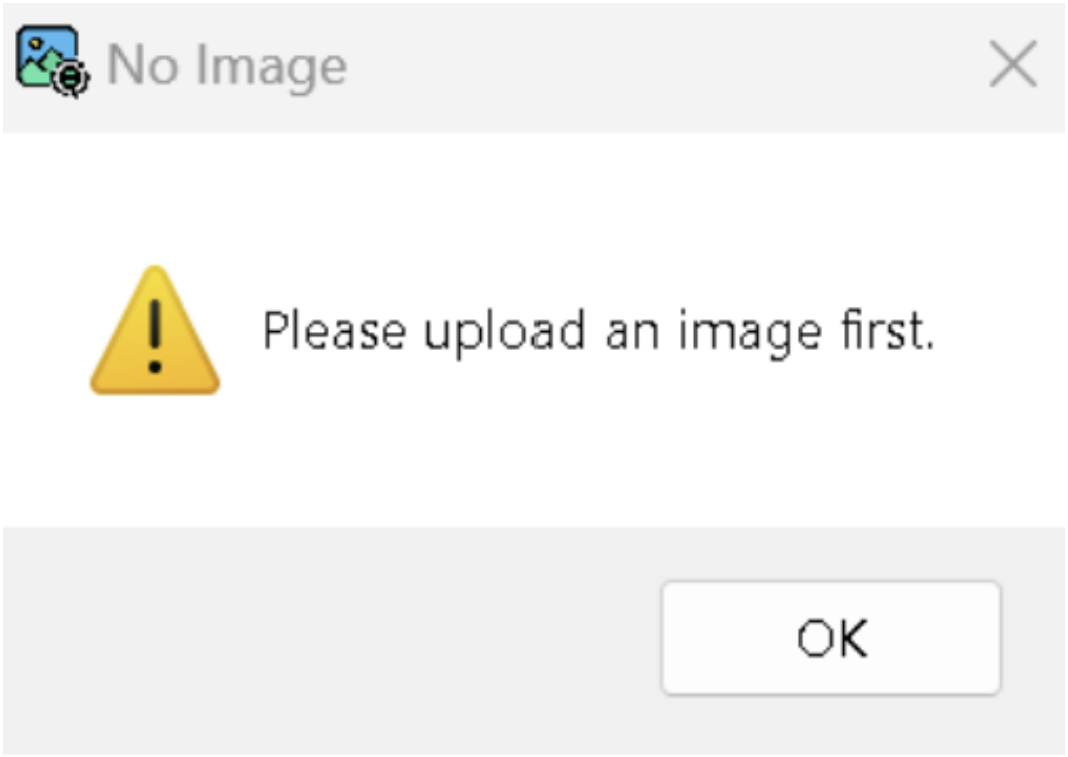

Figure 46: Outcome to Test Case 4

7 Evaluation

Our final product successfully meets all the outlined criteria, establishing itself as a capable and functional image processing software. It has powerful image processing functionalities, enabling it to perform complex image transformations from one state to another. However, like every other software in the world, ImageCraft contains certain limitations that, if addressed, could significantly enhance its functionality and user experience. I am quickly going to mention some of these, and why they would increase the softwares overall usability and reach.

7.1 Limitations

1. Absence of Batch Processing: The software can only process one image at a time, limiting efficiency for users working with multiple files. This is because they would have to process one image at a time which can be very time consuming.

2. No Undo Option: Once an effect is applied, users cannot undo specific steps without resetting the image, reducing flexibility. This means that mistakes are not allowed when using ImageCraft, since it means the user will have to start right again from the beginning.

3. Limited Import Options: Users can only import files from device storage, with no support for cloud-based file imports. This means that the software is tied down to the local disk space and is not able to benefit from remote cloud-based storage systems such as Google Drive.

4. Lack of Advanced Processing Features: Advanced options such as image filters are absent, limiting creative processing capabilities. Additional operations such as image cropping are also absent, seriously limiting the processes one can do with the software.

8 References

References

[1] Clark, A. et al. (2024) *Pillow (PIL Fork) Documentation*. Python Software Foundation. Available at: `https://pillow.readthedocs.io/en/stable/` (Accessed: 18 December 2024).

[2] Lundh, F. (2024) *An Introduction to Tkinter*. Python Software Foundation. Available at: `https://docs.python.org/3/library/tkinter.html` (Accessed: 22 December 2024).

[3] Haertel, T. (2024) *CustomTkinter Documentation*. Available at: `https://github.com/TomSchimansky/CustomTkinter` (Accessed: 12 December 2024).

[4] Summerfield, M. (2024) *Advanced Image Processing with PIL*. Available at: `https://realpython.com/image-processing-with-the-python-pillow-library/` (Accessed: 25 December 2024).

[5] Bro Code (2024) *Python Tkinter Full Course for free*. YouTube. Available at: `https://youtu.be/TuLxsvK4svQ` (Accessed: 20 December 2024).

[6] PyMoondra (2024) *Image Processing Tutorial for Beginners with Python PIL in 30 mins*. YouTube. Available at: `https://youtu.be/dkp4wUhCwR4` (Accessed: 16 December 2024).

[7] Spadeyk (2024) *Snake Maths Game - Login System*. GitHub Repository. Available at: `https://github.com/spadeyk/Snake-Maths-Game-/blob/main/LogInSystem.py` (Accessed: 28 December 2024).