MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

OBJECT ORIENTED PROGRAMMING (CSE -2143) MINI PROJECT REPORT ON

ImageFX

SUBMITTED TO
Department of Computer Science & Engineering

by

Rohan Khandelwal 220905143
Ishaan Vatus 220905043
Anshuman 220905087
Adithi Angeerasa 220905105

CSE-D

Name & Signature of Evaluator 1 Name & Signature of
Evaluator 2

Table of Contents

INTRODUCTION

General

Introducing our Image Processing Application, a Java program that puts the power of creative image manipulation into your hands. With a focus on user-friendly design and object-oriented principles, this application provides an array of features to enhance your visual creations. From classic filters like Grayscale and Sepia to advanced effects using Kernel Convolutions, you can effortlessly transform your images. Plus, efficient image compression and resizing tools make sharing and adapting images a breeze, all while preserving quality. Dive into detailed image analysis with our histogram tools.

The Image Processing Application

The motivation for creating an Image Processing Application with a set of features as outlined is driven by several factors, taking into account the evolving landscape of digital imagery and the diverse needs of users in today's world.

1. Artistic Expression:

The diverse selection of filters, including classic adjustments and advanced artistic effects, empowers users to infuse their images with their unique style and vision. This application becomes a canvas for their digital creativity.

2. Image Enhancement:

With features for resizing, compression, and histogram analysis, this application equips users to enhance image quality, ensuring that their visual content shines in various contexts.

3. User-Friendly Accessibility:

This application is motivated by the desire to make image manipulation accessible to a broader audience. Its user-friendly interface ensures that even those without extensive technical knowledge can confidently use the tool to transform, optimize, and analyze their images.

4. Efficiency:

Image compression is vital in today's data-centric world, as it reduces file sizes for efficient storage, sharing, and uploading. The application's efficient compression system contributes to a more efficient digital ecosystem.

5. Adaptation to Different Media Requirements:

The resizing feature is essential for adapting images to various platforms and media. Whether it's optimizing images for web display, creating thumbnails, or preparing images for specific media formats, the application simplifies these tasks, saving users time and effort while maintaining image quality.

In essence, the motivation for creating this Image Processing Application is grounded in the need to cater to the diverse and evolving requirements of users in an image-centric world.

BACKGROUND THEORY

The Java program is an application that uses JavaFX to create a simple image processing tool with various features. It utilizes several Java concepts and libraries to achieve its functionality. Here's a list of Java concepts and libraries used in the program:

1. JavaFX: The program is a JavaFX application, which is a Java library for creating graphical user interfaces (GUIs). It is used for creating the user interface, handling events, and displaying images and charts.

2. Inheritance: In JavaFX, theApplication class is extended in this program to create the main application class.

3. Event Handling: The program uses event handling to respond to user interactions. Event handlers are set up for buttons, combo boxes, and sliders to trigger specific actions when these GUI elements are interacted with.

4. Exception Handling: Exception handling is used to handle potential errors that might occur during image loading, image saving, or other file operations. The program includes try-catch blocks to handle exceptions and provide error messages in case of issues.

5. Multithreading: This program does not explicitly use multithreading, but it follows the JavaFX threading model. JavaFX applications run on a single thread, the JavaFX

Application Thread, where GUI updates and event handling occur. 6. Java I/O: The program uses Java I/O classes for reading and writing image files. It reads images from files using File and FileChooser and saves processed images as PNG files using Java's ImageIO and related classes.

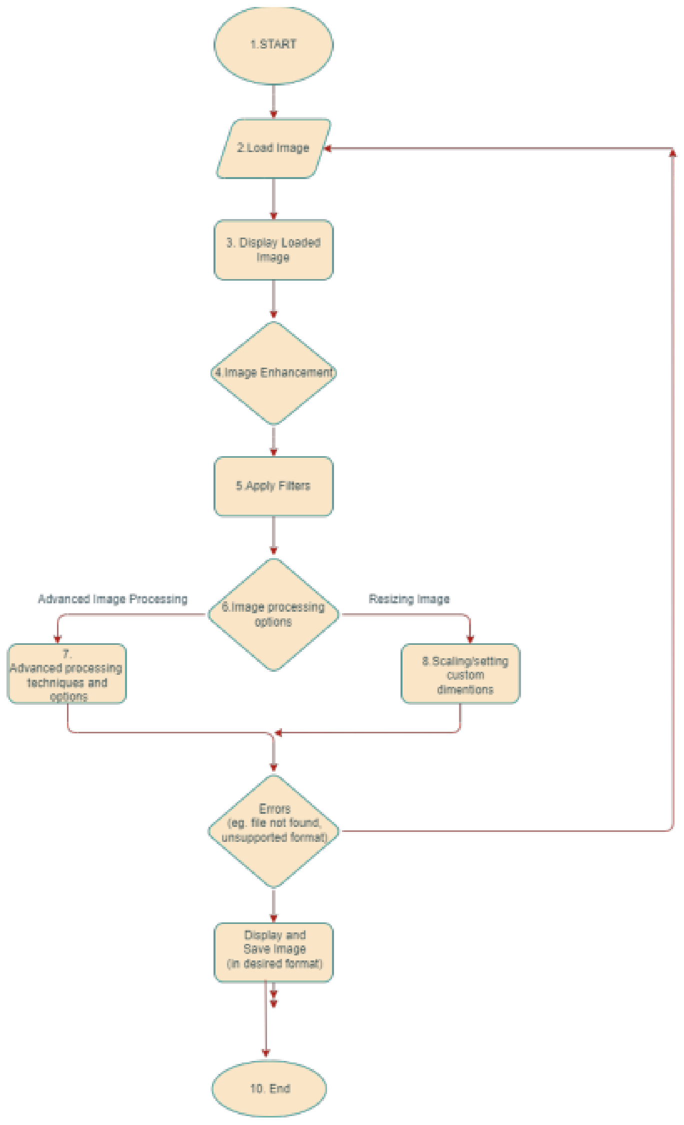
7. Graphics and Image Processing: The program performs basic image processing operations, such as applying filters like grayscale, sepia, and blur to images. It uses JavaFX's ImageView, Image, and pixel manipulation to display and process images.

8. Data Structures: The program uses arrays and 2D arrays to store and manipulate histogram data for image processing.

9. JavaFX Charts: The program uses JavaFX charts to display histograms. It utilizes BarChart, CategoryAxis, NumberAxis, and XYChart.Series to create and display histograms of image color channels (red, green, blue).

In summary, the program primarily revolves around JavaFX for creating the GUI and image processing components, while also making use of several core Java concepts such as exception handling, I/O, and event handling to provide its functionality.

METHODOLOGY


1.START
2.Load Image
3. Display Loaded Image
4.Image Enhancement
5.Apply Filters
6.Image processing options
Advanced Image Processing
Resizing Image
7. Advanced processing techniques and options
8.Scaling/setting custom dimentions
Errors (eg. file not found, unsupported format)
Display and Save Image (in desired format)
10. End

These are the functions we used to create an image processing application using JavaFX:

1. start: Acts as the main entry point for the JavaFX application. It sets up the initial user interface with buttons, sliders, and an image view for loading, displaying, manipulating, and saving images.

2. loadAndDisplayImage: Opens a file chooser dialog for selecting an image file (supporting various formats) and displays the chosen image in the application's user interface.

3. applyFilter: Applies different image filters such as Grayscale, Sepia, Blur, or removes all filters, altering the visual appearance of the displayed image.

4. histogram: Generates a histogram chart from the selected image, illustrating the distribution of pixel intensities for the red, green, and blue channels in a separate window.

5. setGrayscaleFilter: Converts the displayed image to grayscale, representing it in black and white.

6. setSepiaFilter: Applies a sepia tone filter to the displayed image, giving it a vintage, sepia-like appearance.

7. setBlurFilter: Adds a Gaussian blur effect to the displayed image, creating a blurred visual effect.

8. clearFilters: Removes any applied image filters from the displayed image, restoring it to its original state.

9. saveImage: Allows the user to save the currently displayed image after potential modifications. It provides options to specify the save location and adjust image compression quality using a slider.

These functions collectively enable users to load, view, edit, and save images with various filters and visual representations.

```java
import javafx.application.Application;

import javafx.scene.Scene;

import javafx.stage.FileChooser;

import javafx.stage.Stage;

import javafx.stage.Window;

import javafx.stage.FileChooser.ExtensionFilter;

import javafx.scene.layout.VBox;

import javafx.scene.paint.Color;

import javafx.scene.control.Button;

import javafx.scene.control.ComboBox;

import javafx.scene.effect.ColorAdjust;

import javafx.scene.effect.GaussianBlur;

import javafx.scene.effect.SepiaTone;

import javafx.scene.image.Image;

import javafx.scene.image.ImageView;

import javafx.scene.image.PixelReader;


//Importing modules for Charts

import javafx.scene.chart.BarChart;

import javafx.scene.chart.CategoryAxis;

import javafx.scene.chart.NumberAxis;

import javafx.scene.chart.XYChart;


import java.io.File;


import javafx.scene.image.WritableImage;

import javafx.embed.swing.SwingFXUtils;


import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;

import java.awt.image.BufferedImage;

import java.util.Iterator;


import javax.imageio.ImageWriteParam;

import javax.imageio.ImageWriter;

import javax.imageio.IIOImage;

import javax.imageio.stream.FileImageOutputStream;


import javafx.scene.control.Slider;


public class ImageFX extends Application

{

    private ImageView imageView = new ImageView();

    private ImageView imageView1 = new ImageView();


    private Image selectedImage = null;

    private Image resized;


    public static void main(String[] args) {

        launch(args);

    }


    public void start(Stage primaryStage) {

        primaryStage.setTitle("Image Processing App");


        VBox root = new VBox(10);
        Button loadImageButton = new Button("Load Image");
        loadImageButton.setStyle("-fx-background-color: gray; -fx-
background-radius: 20; -fx-font-size: 16; -fx-text-fill: white; -fx-border-
color: gray; -fx-effect: dropshadow(gaussian, rgba(0,0,0,0.2), 0, 0, 0,
1);");
        loadImageButton.setOnAction(e -> loadAndDisplayImage(primaryStage));

        imageView1.setFitWidth(300); // Width in pixels
        imageView1.setFitHeight(200); // Height in pixels
        imageView1.setPreserveRatio(true);

        Slider qualitySlider = new Slider(0, 1, 0.7);
        qualitySlider.setBlockIncrement(0.1);
        qualitySlider.setShowTickLabels(true);
        qualitySlider.setShowTickMarks(true);
        qualitySlider.setMajorTickUnit(0.1);
        qualitySlider.setMinorTickCount(0);

        ComboBox<String> filterComboBox = new ComboBox<>();
        filterComboBox.getItems().addAll("Original", "Grayscale", "Sepia",
"Blur");
        filterComboBox.setValue("Original");
        filterComboBox.setOnAction(e ->
applyFilter(filterComboBox.getValue()));

        Button histoButton = new Button("Histogram");
        histoButton.setOnAction(e -> histogram(selectedImage));


        Button saveImageButton = new Button("Save Image");
        saveImageButton.setOnAction(e -> saveImage(primaryStage,
qualitySlider));

        root.getChildren().addAll(loadImageButton, imageView1,
filterComboBox, histoButton, qualitySlider, saveImageButton);

        Scene scene = new Scene(root, 800, 600);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private void loadAndDisplayImage(Window ownerWindow) {
        FileChooser fileChooser = new FileChooser();
        fileChooser.getExtensionFilters().add(new ExtensionFilter("Image
Files", "*.png", "*.jpg", "*.jpeg", "*.gif", "*.bmp"));

        File selectedFile = fileChooser.showOpenDialog(ownerWindow);

        if (selectedFile != null) {
            String imagePath = selectedFile.toURI().toString();
            selectedImage = new Image(imagePath);
            resized = new Image(imagePath);
            imageView.setImage(selectedImage);
            imageView1.setImage(resized);
        }
    }

    private void applyFilter(String filter) {
        switch (filter) {
            case "Grayscale":
```

```java
            setGrayscaleFilter();
            break;
        case "Sepia":
            setSepiaFilter();
            break;
        case "Blur":
            setBlurFilter();
            break;
        case "Original":
        default:
            clearFilters();
            break;
    }
}

private void histogram(Image img)
{
    int[][] histogram_data = new int[3][256];

    for(int i[]: histogram_data)
    {
        for(int j: i)
        {
            j = 0;
        }
    }

    PixelReader pixelReader = img.getPixelReader();
    int width = (int) img.getWidth();
```

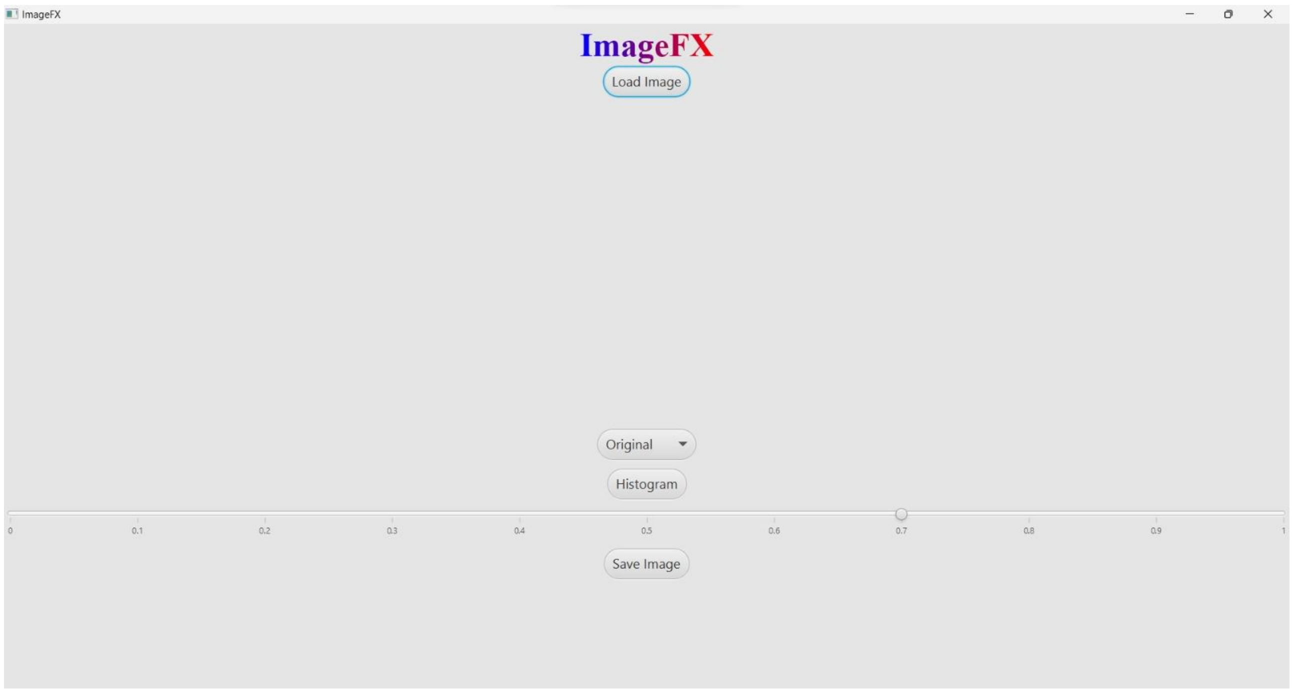
```
        int height = (int) img.getHeight();

        for (int x = 0; x < width; x++) {
            for (int y = 0; y < height; y++)
            {

                int pixel = pixelReader.getArgb(x, y);
                int red = (pixel >> 16) & 0xFF;
                int green = (pixel >> 8) & 0xFF;
                int blue = pixel & 0xFF;

                histogram_data[0][red]++;
                histogram_data[1][green]++;
                histogram_data[2][blue]++;
            }
        }

        CategoryAxis categoryAxis = new CategoryAxis(); // X-axis for
categories
        NumberAxis valueAxis = new NumberAxis(); // Y-axis for values
        BarChart<String, Number> histogramChart = new
BarChart<>(categoryAxis, valueAxis);
        histogramChart.setTitle("Histogram");
        categoryAxis.setLabel("Intensity Level");
        valueAxis.setLabel("Frequency");

        XYChart.Series<String, Number> redSeries = new XYChart.Series<>();
        XYChart.Series<String, Number> greenSeries = new XYChart.Series<>();
        XYChart.Series<String, Number> blueSeries = new XYChart.Series<>();
        for (int i = 0; i < 256; i++)
        {
            redSeries.getData().add(new XYChart.Data<>(Integer.toString(i),
histogram_data[0][i]));
            greenSeries.getData().add(new
XYChart.Data<>(Integer.toString(i), histogram_data[1][i]));
            blueSeries.getData().add(new XYChart.Data<>(Integer.toString(i),
histogram_data[2][i]));
        }

        redSeries.setName("Red Channel");
        greenSeries.setName("Green Channel");
        blueSeries.setName("Blue Channel");

        histogramChart.getData().addAll(redSeries, greenSeries, blueSeries);



        Stage histogramStage = new Stage();
        histogramStage.setTitle("Histogram");
        histogramStage.setScene(new Scene(histogramChart, 600, 400));
        histogramStage.show();
    }



    private void setGrayscaleFilter() {
        ColorAdjust colorAdjust = new ColorAdjust();
        colorAdjust.setSaturation(-1);
        imageView.setEffect(colorAdjust);
        imageView1.setEffect(colorAdjust);
    }
    private void setSepiaFilter() {
        SepiaTone sepiaTone = new SepiaTone();
        sepiaTone.setLevel(0.8); // Adjust sepia intensity here
        imageView.setEffect(sepiaTone);
        imageView1.setEffect(sepiaTone);

    }

    private void setBlurFilter() {
        GaussianBlur blur = new GaussianBlur(10);
        imageView.setEffect(blur);
        imageView1.setEffect(blur);

    }

    private void clearFilters() {
        imageView.setEffect(null);
        imageView1.setEffect(null);
    }



    private void saveImage(Window ownerWindow, Slider qualitySlider) {
        if (selectedImage != null) {
            FileChooser fileChooser = new FileChooser();
            fileChooser.getExtensionFilters().add(new ExtensionFilter("PNG
Files", "*.png"));

            File selectedFile = fileChooser.showSaveDialog((Stage)
ownerWindow);
```

```
            if (selectedFile != null) {
                try {
                    int width = (int) selectedImage.getWidth();
                    int height = (int) selectedImage.getHeight();

                    WritableImage writableImage = new WritableImage(width,
height);
                    imageView.snapshot(null, writableImage);

                    BufferedImage bufferedImage =
SwingFXUtils.fromFXImage(writableImage, null);

                    Iterator<ImageWriter> writers =
ImageIO.getImageWritersByFormatName("png");
                    if (writers.hasNext()) {
                        ImageWriter writer = writers.next();
                        ImageWriteParam writeParam =
writer.getDefaultWriteParam();

writeParam.setCompressionMode(ImageWriteParam.MODE_EXPLICIT);

                        double compressionQuality =
qualitySlider.getValue();
                        writeParam.setCompressionQuality((float)
compressionQuality);

                        FileImageOutputStream output = new
FileImageOutputStream(selectedFile);
                        writer.setOutput(output);
                        writer.write(null, new IIOImage(bufferedImage, null,
null), writeParam);
```

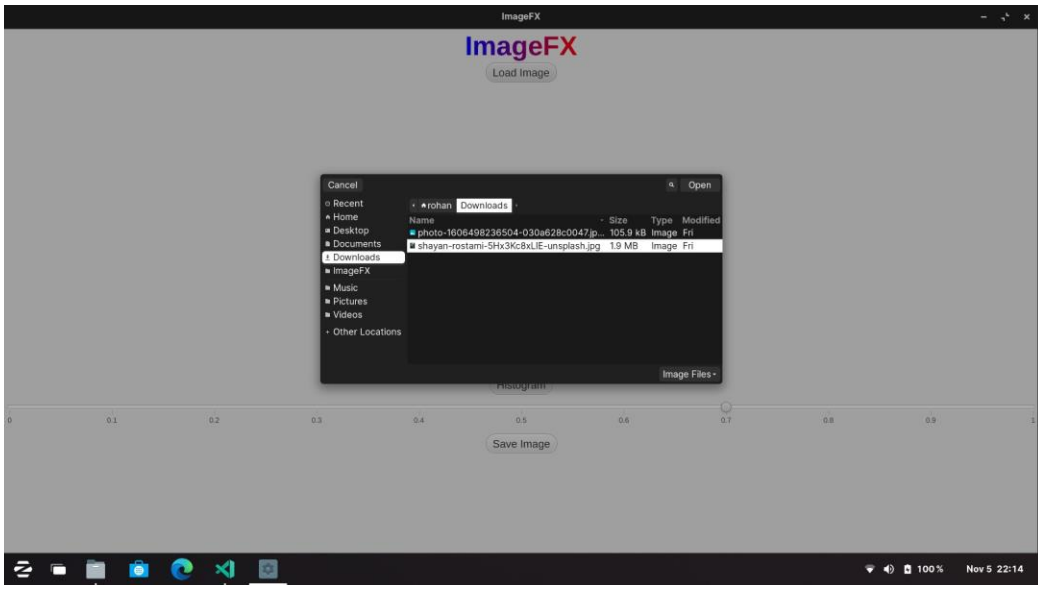
```
                        writer.dispose();

                        System.out.println("Image saved to: " +
selectedFile.getAbsolutePath());
                    } else {
                        System.out.println("No suitable image writer
found.");
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            } else {
                System.out.println("No image to save.");
            }
        }
    }
}
```

RESULTS AND DISCUSSION

Our application has a professional looking interface that is easy to use.



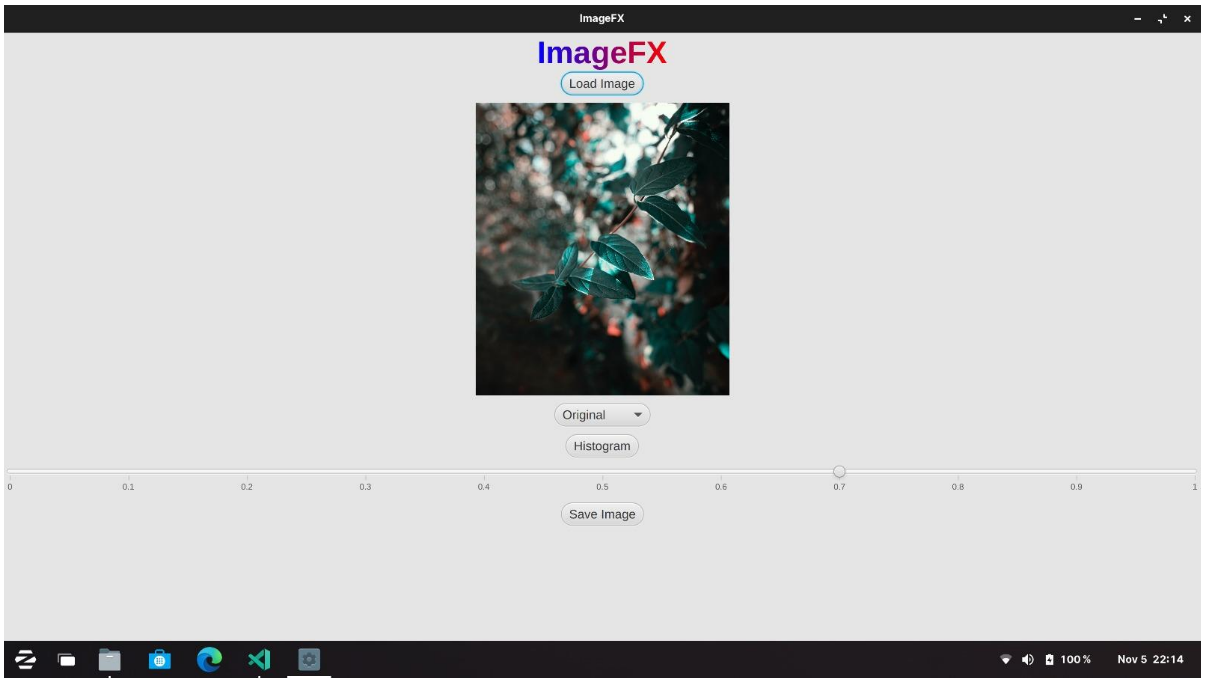
The user can load the image to be filtered by simply dragging and dropping it onto the new window that opens.

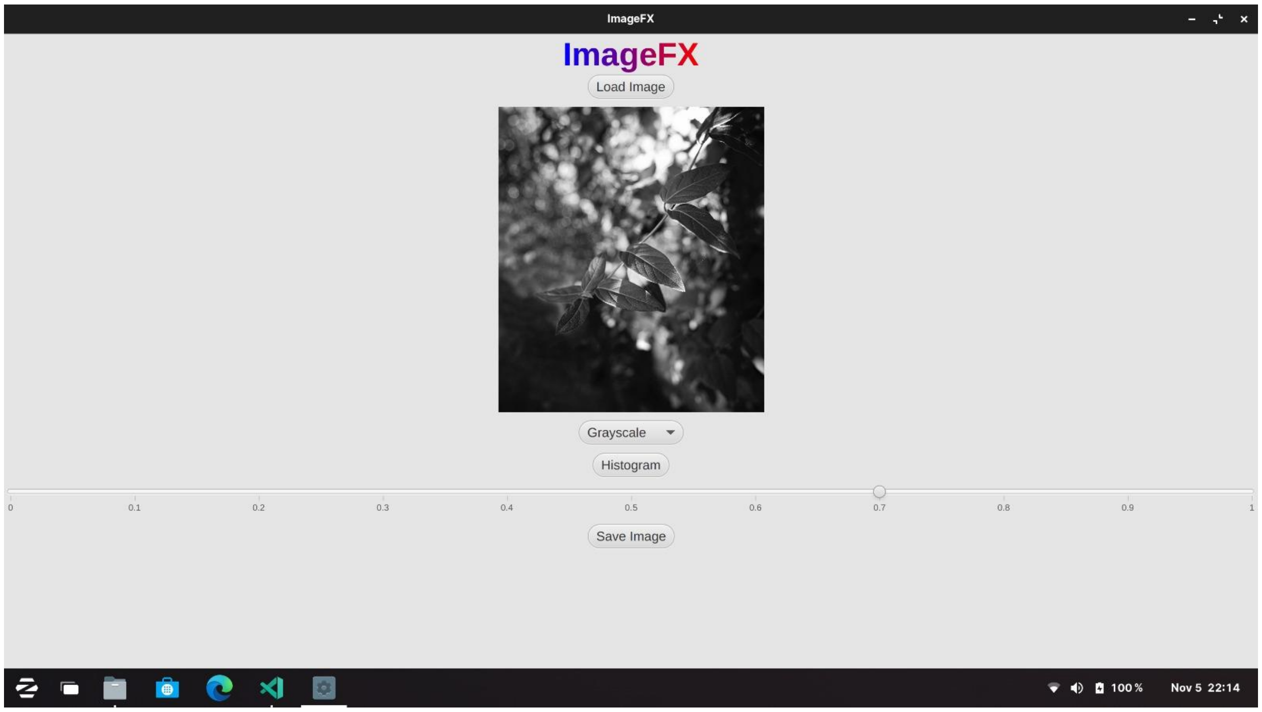


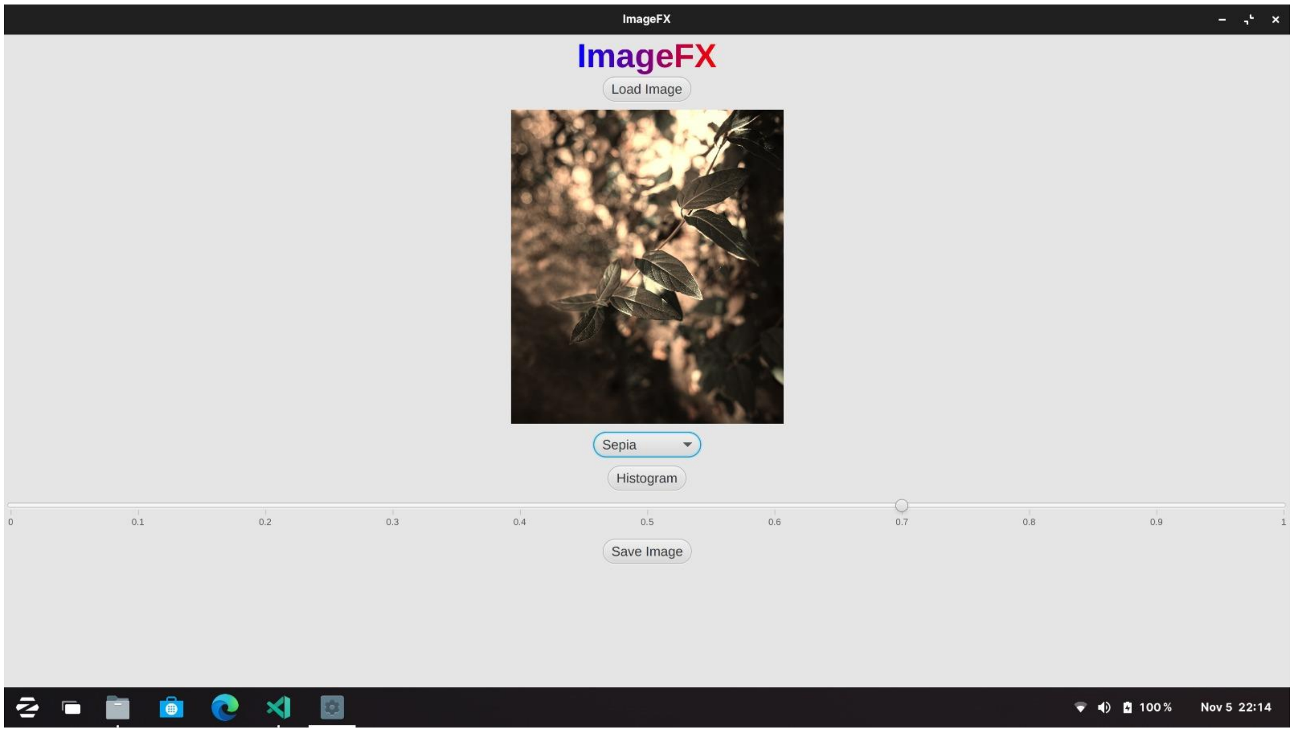
Name	Size	Type	Modified
photo-1606498236504-030a628c0047.jp...	105.9 kB	Image	Fri
shayan-rostami-5Hx3Kc8xLIE-unsplash.jpg	1.9 MB	Image	Fri

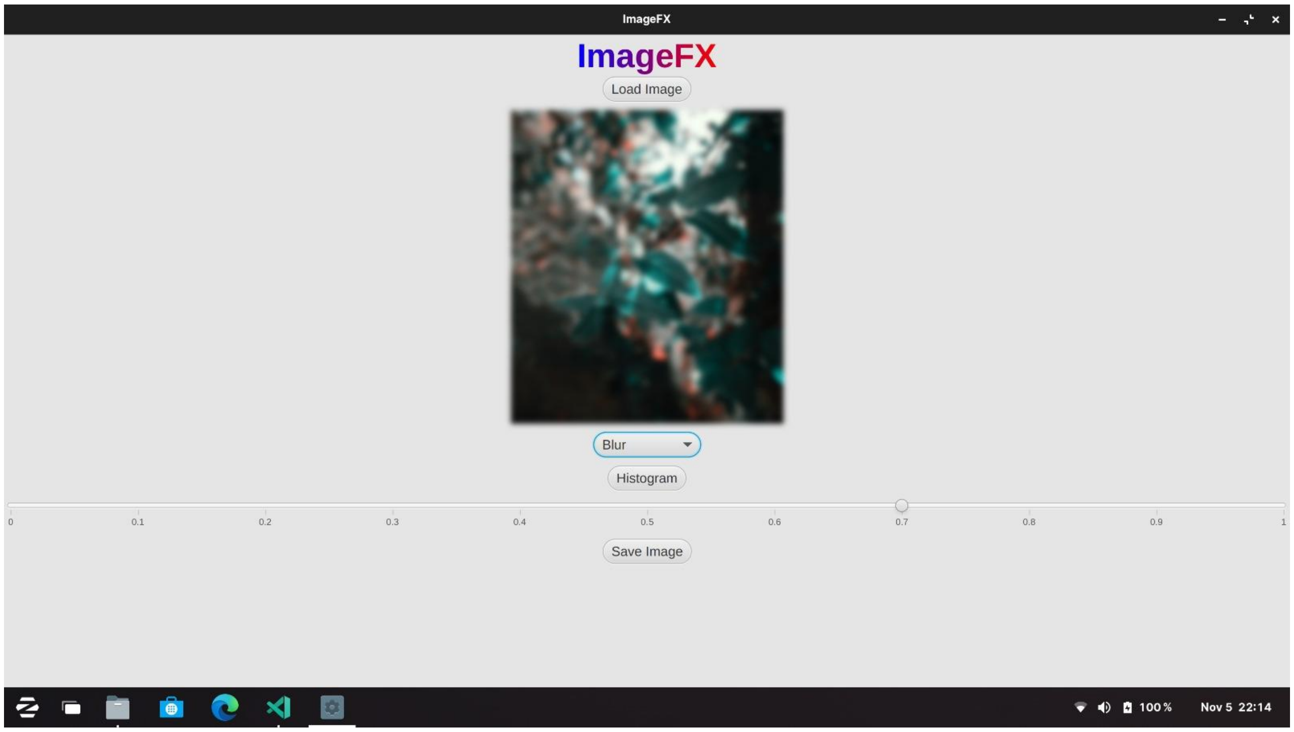
The original image is displayed. Desired filters can be chosen from the drop-down box.

Here are a few of the available filters

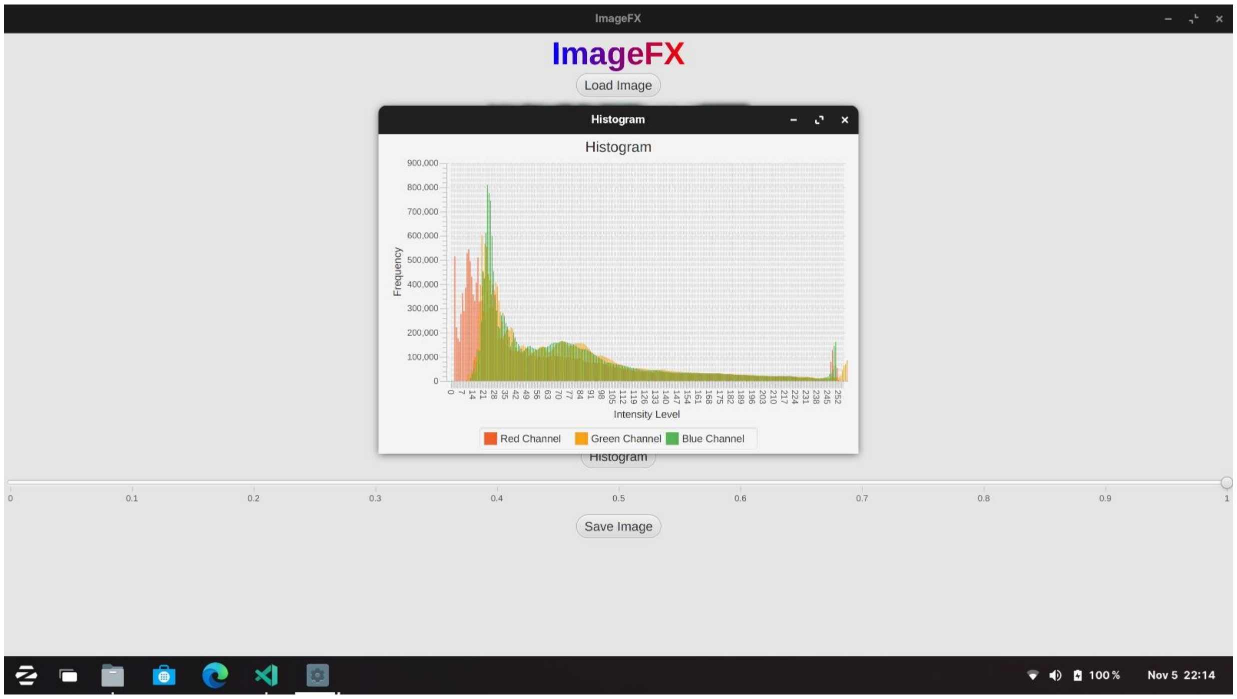


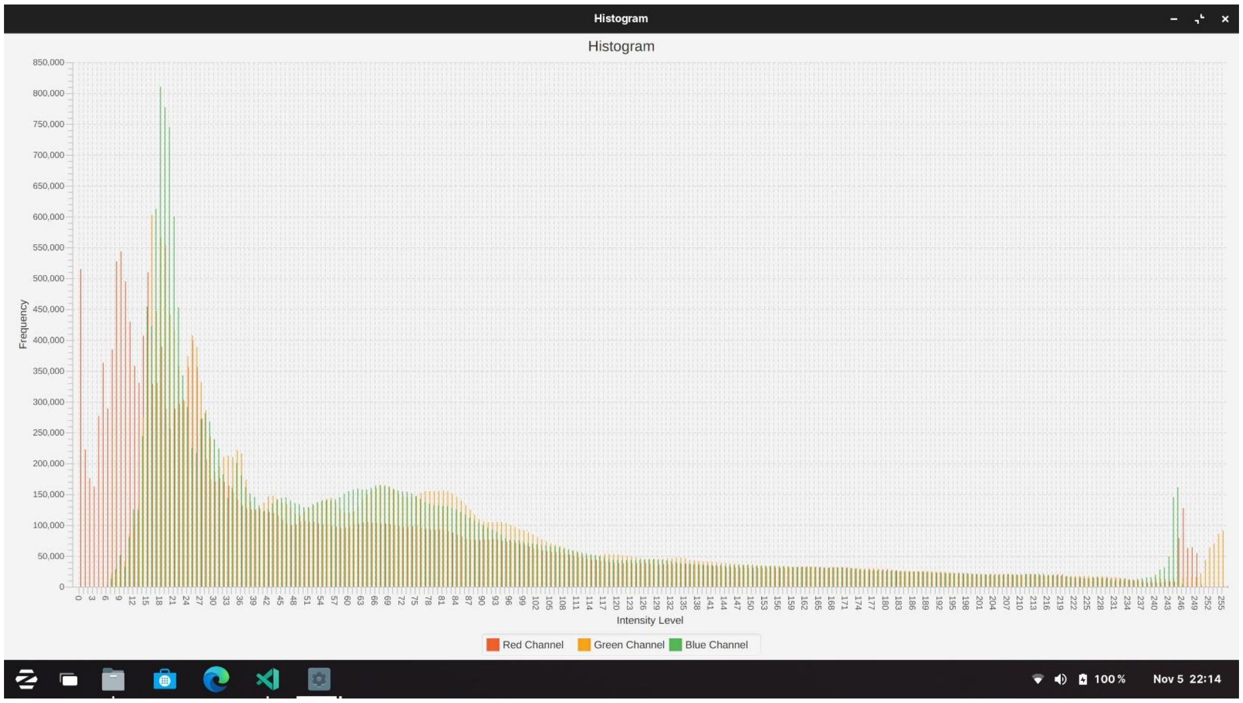



ImageFX
ImageFX
Load Image
Sepia
Histogram
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Save Image
100 %
Nov 5 22:14


ImageFX
ImageFX
Load Image
Blur
Histogram
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Save Image
100 %
Nov 5 22:14

There is also an option to display the histogram:





CONCLUSION

In conclusion, the development of an Image Processing Application using JavaFX by implementing a rich array of image filters empowers users to transform their images.

Moreover, the application's robust image compression algorithms and resizing capabilities address the practical demands of our digital era. Sharing and adapting images for different platforms and purposes become effortless, while the preservation of image quality ensures that visual integrity remains intact.

The inclusion of histogram analysis tools opens the door to precise adjustments in brightness, contrast, and tonal balance, facilitating optimal exposure and color balance. This application provides users with the tools they need to achieve the highest standards of image quality.

JavaFX, with its scene graph, event handling, and CSS styling, forms the bedrock for an intuitive and user-friendly interface. The efficient integration of Java and JavaFX enables a seamless user experience.

It should be noted thatin the future, this application can implement multithreading to improve performance. Long-running tasks, such as loading and processing images, should be executed off the Application Thread to avoid freezing the GUI. However, in this program, image loading and processing are executed on the JavaFX Application Thread, which might cause unresponsiveness if dealing with large images.

In essence, this Image Processing Application is not merely a software tool but a canvas for digital artists. With this application, we aim to empower users to explore the boundless possibilities of image manipulation while simplifying their interaction with images.

REFERENCES

[1]. *Java Programming: A Comprehensive Introduction, Herb Schildt and Dale Skrien*

[2]. OOP slides uploaded on lighthouse

[3]. *Using JavaFX Charts: About this tutorial* | *JavaFX 2 tutorials and documentation*. (n.d.). https://docs.oracle.com/javafx/2/charts/jfxpub-charts.htm

[4]. Getting Started with JavaFX. (n.d.). https://openjfx.io/openjfx-docs/