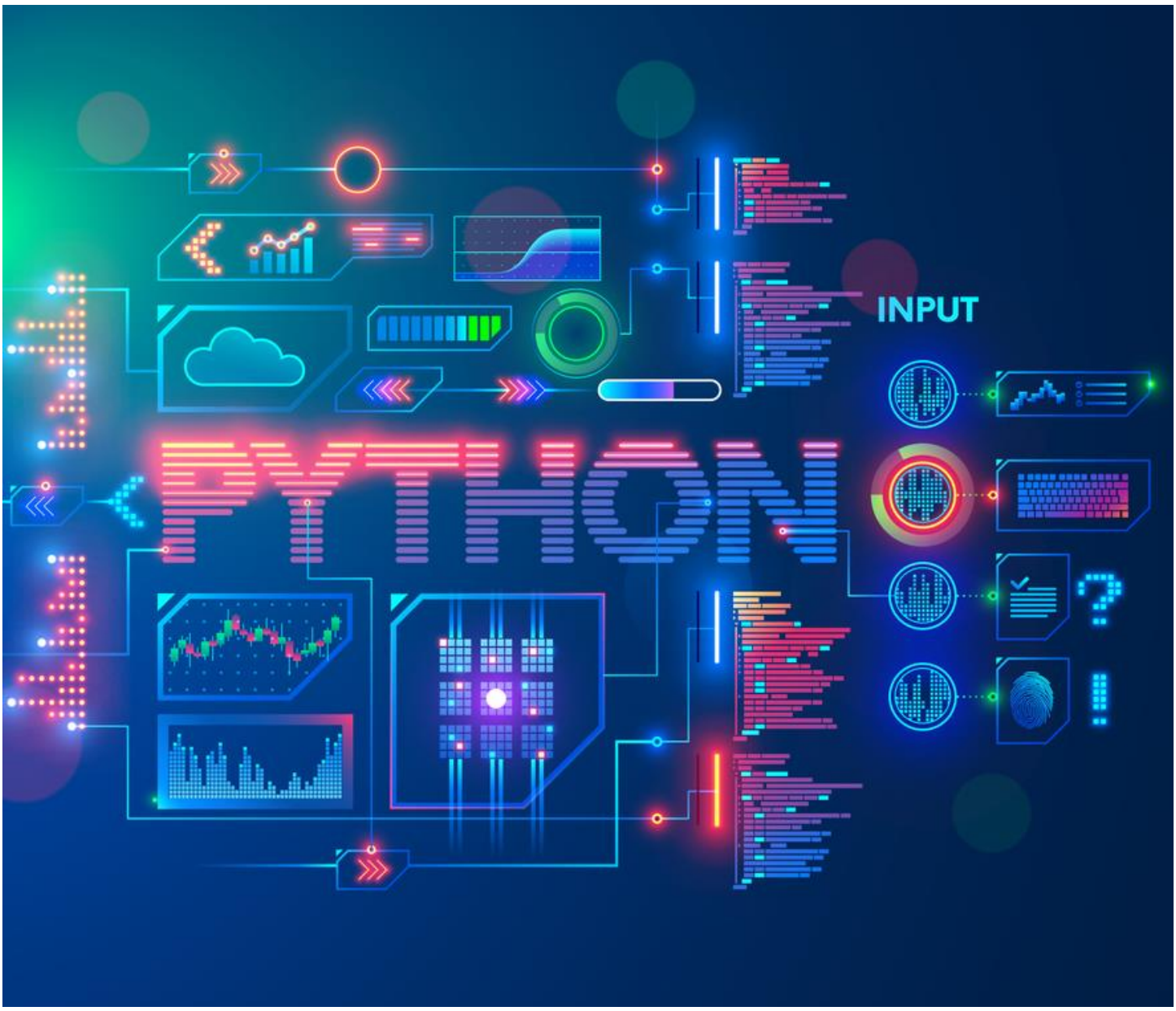


PYTHON II

Final Project Report

A project report

Of

“Create My Sign App”

Submitted as a part of the curriculum

for the degree of Postgraduate

In

Full stack Software Development

Submitted By

Meet Patel-C0912711

Submitted To

Mr. Darcy Gratton

Instructor of Python- II

LAMBTON COLLEGE-MISSISSAUGA

Table of Contents

1. Introduction
2. Project Overview
3. Software Requirements
4. Features
5. Installation Guide
6. Code Explanation
7. User Manual
8. Challenges and Solutions
9. Future Enhancements
10. Conclusion

Introduction:

In the digital age, personalization and digital identity have become paramount for individuals and businesses alike. Recognizing this trend, the "Create My Sign" application was developed to provide a user-friendly, efficient solution for creating and managing digital signs and signatures. The project was motivated by the need for a simple, accessible tool that allows users to design and produce custom digital signatures and text signs without requiring advanced graphic design skills or software.

The "Create My Sign" application aims to empower users by enhancing their ability to express themselves through unique digital signs and signatures, thereby enhancing their digital presence.

Project Overview:

The "Create My Sign" application is designed to allow users to effortlessly create digital signatures and text signs through a straightforward graphical interface. It features the ability to create and manage user accounts, select from various fonts and colors, and use freehand drawing tools for customization. Additionally, users can save their personalized signs within the application or download them for use in other digital formats. This tool aims to make the process of creating a digital identity simple and accessible for users regardless of their technical background.

Software Requirements

- **Python:** Version 3.8 or above
- **Libraries:** tkinter for GUI, PIL for image handling, pymongo for database interaction, and others.
- **Database:** MongoDB
- **Operating System:** Compatible with Windows, macOS, and Linux

Key Features:

- **User Authentication:** Allows users to create an account or login to access personalized features.
- **Customizable Text Input:** Users can input text to create a sign with options to select font styles and sizes.
- **Color Picker:** Users can choose different colors for their text or drawing.
- **Drawing Canvas:** Provides a canvas where users can draw their signatures manually.
- **Download Functionality:** Enables users to download their signs as PNG files.
- **Save to Database:** Signs can be saved to a MongoDB database for logged-in users.

Installation Guide

1. Python Installation: Download and install Python then ensure that Python is added to your system's PATH during installation.
2. Library Installation: Now open python and use pip to install the required libraries by running the following commands in your terminal or command prompt: **pip install tk pillow pymongo**
3. MongoDB Installation: Download and install MongoDB and make sure the MongoDB server is running.
4. Running the Application: Navigate to the directory containing the application files and run the Python script.

By following these steps, we'll be able to install and run the "Create My Sign" application on our system.

Project Code:

```
import tkinter
from tkinter import colorchooser, messagebox
from PIL import ImageGrab,ImageTk,Image
import tkinter.font as tkFont
from hashlib import sha256
import pymongo
import re
from bson.binary import Binary
from io import BytesIO


class CreateMySignApp:
    #initializing helper variables in constructor
    def __init__(self):
        #creating main Window
        self.window = self.createMainWindow()
        self.canvas = None
        self.color = "black"
        self.creating_sign = False
        self.pen_x = 0
        self.pen_y = 0
        self.email_entry = None
        self.password_entry = None
        self.login_window = None
        self.isLoggedIn = False
        self.useremail = None
        self.client = None
        self.scroll_frame = None
        self.login_create_account_btn = None
        self.input_box = None
        self.font_dropdown = None
        self.font_size_dropdown = None
        self.createGUI()

    #Function to create Main window
    def createMainWindow(self):
        window = tkinter.Tk()
        window.title("Create My Sign")
        window.config(bg="#e6e3e3")
        window.minsize(width=1000, height=1000)
        heading = tkinter.Label(window, text="Welcome to Make My Sign",
font=("Helvetica", 20, "bold"))
        heading.place(x=290, y= 30)
```

```
        window.protocol("WM_DELETE_WINDOW", self.destroyWindow)

        return window

    def destroyWindow(self):
        if not (self.client is None):
                self.client.close()
                self.client = None
        self.window.destroy()



    #Function which places button and canvas on the main screen by calling helper
functions
    def createGUI(self):
        self.createCanvas()
        self.createButtons()
        self.createInput()
        self.window.mainloop()

    #creating canvas
    def createCanvas(self):
        self.canvas = tkinter.Canvas(self.window, width=650, height=500,
borderwidth=2, relief="solid")
        self.canvas.place(x=150, y=100)
        self.canvas.bind("<Button-1>", self.onCanvasClick)
        self.canvas.bind("<B1-Motion>", self.onCanvasMotion)

    #creating fontsize,download,rest and font style,login,colorpicker buttons
    def createButtons(self):
        self.createDownloadButton()
        self.createResetButton()
        self.createFontDropdown()
        self.createFontSizeDropDown()
        self.createLoginButton()
        self.createColorPickerButton()

    #login button
    def createLoginButton(self):
        self.login_create_account_btn = tkinter.Button(self.window, text="Sign
In/ Sign Up", command=self.createLoginFrame, width=15, height=2)
        self.login_create_account_btn.place(x=850, y=50)

    #Enter sign text entry
    def createInput(self):
```

```
        input_label = tkinter.Label(self.window, text="Enter Your Sign : ")
        input_label.place(x=570, y=670)
        self.input_box = tkinter.Entry(self.window, width=15)
        self.input_box.place(x=570, y=700)
        self.input_box.bind("<KeyRelease>", self.getInputValue)

    #font style dropdown
    def createFontDropdown(self):
        font_dropdown_label = tkinter.Label(self.window, text="Font Styles : ")
        font_dropdown_label.place(x=690, y=670)
        font_names = tkFont.families()
        self.font_dropdown = tkinter.StringVar()
        self.font_dropdown.set(font_names[0])
        font_dropdown_menu = tkinter.OptionMenu(self.window, self.font_dropdown,
*font_names, command=self.changeFonts)
        font_dropdown_menu.place(x=690, y=700)

    #font size dropdown
    def createFontSizeDropDown(self):
        font_size_label = tkinter.Label(self.window, text="Font Size : ")
        font_size_label.place(x=820, y=670)
        font_sizes = [8, 10, 12, 14, 16, 18, 20, 22, 24]
        self.font_size_dropdown = tkinter.IntVar()
        self.font_size_dropdown.set(12)  # Default font size
        font_size_dropdown_menu = tkinter.OptionMenu(self.window,
self.font_size_dropdown, *font_sizes, command=self.changeFonts)
        font_size_dropdown_menu.place(x=820, y=700)

    #function to change font style and font size on changing font and size
    def changeFonts(self,*args):
        font_name = self.font_dropdown.get()
        font_size = self.font_size_dropdown.get()
        try:
            value = self.input_box.get()
            self.canvas.delete("all")
            x = self.canvas.winfo_width() // 2
            y = self.canvas.winfo_height() // 2
            self.creating_sign = True
            #recreating sign with changed style and size
            self.canvas.create_text(x, y, text=value, font=(font_name,
font_size), fill=self.color, tags=("text",))
        except Exception as e:
            messagebox.showerror("Error", f"Failed to get input value: {str(e)}")
    #Color picker
    def createColorPickerButton(self):
```

```
        colorLabel = tkinter.Label(self.window, text="Choose Sign Color : ")
        colorLabel.place(x=450,y=670)
        color_button = tkinter.Button(self.window, text="Choose Color",
command=self.chooseColor, width=15, height=2)
        color_button.place(x=430, y=700)

    #download button
    def createDownloadButton(self):
        downloadLabel = tkinter.Label(self.window, text="Click Here To Download :
")
        downloadLabel.place(x=290,y=670)
        download_button = tkinter.Button(self.window, text="Download",
command=self.onDownloadClick, width=15, height=2)
        download_button.place(x=290, y=700)
    #rest button
    def createResetButton(self):
        resetLabel = tkinter.Label(self.window, text="Click Here To Reset : ")
        resetLabel.place(x=150,y=670)
        reset_button = tkinter.Button(self.window, text="Reset",
command=self.onResetClick, width=15, height=2)
        reset_button.place(x=150, y=700)

    #function to create login window
    def createLoginFrame(self):
        self.login_window = tkinter.Toplevel(self.window)
        self.login_window.geometry("300x200")
        self.login_window.title("Login Window")
        email_label = tkinter.Label(self.login_window, text="Email")
        email_label.grid(row=1, column=0, padx=5, pady=5)
        self.email_entry = tkinter.Entry(self.login_window)
        self.email_entry.grid(row=1, column=1, padx=5, pady=5)
        password_label = tkinter.Label(self.login_window, text="Password")
        password_label.grid(row=2, column=0, padx=5, pady=5)
        self.password_entry = tkinter.Entry(self.login_window, show='*')
        self.password_entry.grid(row=2, column=1, padx=5, pady=5)
        login_button = tkinter.Button(self.login_window, text="Login",
command=self.login)
        login_button.grid(row=3, column=0, columnspan=2, padx=5, pady=5)
        create_account_button = tkinter.Button(self.login_window, text="Create
Account", command=self.createAccount)
        create_account_button.grid(row=4, column=0, columnspan=2, padx=5, pady=5)

    #function to DB connection object
    def getDBConnection(self):
        try:
```

```
            self.client = pymongo.MongoClient("mongodb://localhost:27017/")
            db = self.client["generatemysign"]
            return db
        except pymongo.errors.ConnectionError:
            messagebox.showerror("Error", "Failed to connect to the database.")
            return None

    #validating user credentials
    def login(self):
        self.useremail = self.email_entry.get()
        password = self.password_entry.get()
        client = self.getDBConnection()
        users_collection = client["users"]

        if users_collection is None:
            return False

        user = users_collection.find_one({"email": self.useremail})

        if user is None:
            messagebox.showerror("Error", "Invalid Credentials")
            return False

        #if correct credentials then displaying username and user previous
download signs image
        if sha256(password.encode('utf-8')).hexdigest() == user["password"]:
            messagebox.showinfo("Success", "Login Successfully.")
            self.login_window.destroy()

self.login_create_account_btn.config(text="Logout",command=self.logout)
            self.greeting_label = tkinter.Label(self.window, text=f"Hello,
{self.useremail}!")
            self.greeting_label.place(x=750, y=60)
            self.retrieveSigns(self.useremail)
            self.isLoggedIn = True
            return True
        else:
            messagebox.showerror("Error", "Invalid Credentials")
            return False

    #logut function.
    def logout(self):
        if messagebox.askokcancel("Logout", "Are you sure you want to logout?"):
            self.login_create_account_btn.config(text="Sign In/ Sign Up",
command=self.createLoginFrame)
```

```
            self.greeting_label.destroy()
            self.canvas.delete("all")
            self.input_box.delete(0, "end")
            self.scroll_frame.destroy()
            if not (self.client is None):
                self.client.close()
                self.client = None


    #Create account function
    def createAccount(self):
        client = self.getDBConnection()
        users_collection = client["users"]

        if users_collection is None:
            return False

        userpassword = self.password_entry.get()
        useremail = self.email_entry.get()

        #if user alredy exists throwing error.
        user = users_collection.find_one({"email": useremail})
        if user is not None:
            messagebox.showerror("Error", "User already exists!")
            return

        #Checking password criteria. One special character , one cap letter and
minimum 8 characrers.
        if len(userpassword) < 8:
            messagebox.showerror("Error", "Password should be at least 8
characters long!")
            return
        elif not re.search(r'[A-Z]', userpassword):
            messagebox.showerror("Error", "Password should have at least one
uppercase letter!")
            return
        elif not re.search(r'[!@#$%^&*(),.?":{}|<>]', userpassword):
            messagebox.showerror("Error", "Password should have at least one
special character!")
            return

        #encoding password with sha256 module
        userPasswordHash = sha256(userpassword.encode('utf-8')).hexdigest()
        user = {"password": userPasswordHash, "email": useremail}
        users_collection.insert_one(user)
```

```
        messagebox.showinfo("Success", "Your account has been created.")
        self.login_window.destroy()

    #changig colors
    def chooseColor(self):
        self.color = colorchooser.askcolor(title="Choose color")[1]
        if self.creating_sign:
            self.changeFonts()

    #initializing pens point
    def onCanvasClick(self, event):
        if not self.creating_sign:
            self.pen_x = event.x
            self.pen_y = event.y

    #capturing the user sign drawing.
    def onCanvasMotion(self, event):
        if not self.creating_sign:
            self.canvas.create_line(self.pen_x, self.pen_y, event.x, event.y,
fill=self.color, width=5, capstyle=tkinter.ROUND)
            self.pen_x = event.x
            self.pen_y = event.y

    #Download sign function
    def onDownloadClick(self):
        try:
            x = self.canvas.winfo_rootx() + 100
            y = self.canvas.winfo_rooty() + 35
            x1 = x + 600
            y1 = y + 600

            #grabbing sign from canvas and storing it.
            image = ImageGrab.grab(bbox=(x, y, x1, y1))
            image.save("sign.png")
            messagebox.showinfo("Success","Sign downloaded successfully.")

            #if logged in then displaying all the previous and current sing in
separate window
            if self.isLoggedIn:
                self.storeSignImage(self.useremail)
                self.scroll_frame.destroy()
                self.retrieveSigns(self.useremail)
        except Exception as e:
            messagebox.showerror("Error", f"Failed to download the image:
{str(e)}")
```

```
    #reseting the canvas
    def onResetClick(self):
        try:
            self.canvas.delete("all")
            self.creating_sign = False
            self.input_box.delete(0, "end")
        except Exception as e:
            messagebox.showerror("Error", f"Failed to reset: {str(e)}")

    #creating sign using text written by user
    def getInputValue(self, event):
        try:
            value = self.input_box.get()
            font_name = self.font_dropdown.get()
            font_size = self.font_size_dropdown.get()
            font = tkFont.Font(family=font_name)
            self.canvas.delete("all")
            x = self.canvas.winfo_width() // 2
            y = self.canvas.winfo_height() // 2
            self.creating_sign = True
            self.canvas.create_text(x, y, text=value, font=(font_name,
font_size), fill=self.color, tags=("text",))
        except Exception as e:
            messagebox.showerror("Error", f"Failed to get input value: {str(e)}")

    #storing sign image in DB
    def storeSignImage(self,useremail):
        try:
            client = self.getDBConnection()
            sign_collections = client["signCollection"]
            with open('sign.png', "rb") as f:
                image_binary = Binary(f.read())
            sign_collections.insert_one({"userEmail": useremail, "signImage":
image_binary})
        except Exception as e:
            messagebox.showerror("Error", f"Failed to store image in database:
{str(e)}")

    #retrieving image from db.
    def retrieveSigns(self,useremail):
        try:
            images = []
            client = self.getDBConnection()
            sign_collections = client["signCollection"]
```

```
            for image_doc in sign_collections.find({"userEmail": useremail}):
                signImage = image_doc["signImage"]
                image = Image.open(BytesIO(signImage))
                photo = ImageTk.PhotoImage(image)
                images.append(photo)
            #displaying image on separate window
            self.displaySigns(images)
        except Exception as e:
            messagebox.showerror("Error", f"Failed to retrieve images: {str(e)}")

    #function to display images
    def displaySigns(self, images):
        self.scroll_frame = tkinter.Toplevel(self.window)
        self.scroll_frame.geometry("700x700")
        self.scroll_frame.title("Your Signs")

        canvas = tkinter.Canvas(self.scroll_frame, bg="#e6e3e3")
        canvas.pack(side="left", fill="both", expand=True)

        scrollbar = tkinter.Scrollbar(self.scroll_frame, orient="vertical",
command=canvas.yview)
        scrollbar.pack(side="right", fill="y")

        canvas.configure(yscrollcommand=scrollbar.set)
        canvas.bind("<Configure>", lambda e:
canvas.configure(scrollregion=canvas.bbox("all")))

        inner_frame = tkinter.Frame(canvas, bg="#e6e3e3")
        canvas.create_window((0, 0), window=inner_frame, anchor="nw")

        for idx, image in enumerate(images):
            signImage = tkinter.Label(inner_frame,image=image)
            signImage.image = image
            signImage.grid(row=idx + 1, column=0, padx=10, pady=10)

    #closing DB connection
    def __del__(self):
        if not (self.client is None) :
            if self.client:
                self.client.close()
                self.client = None

#creating a object
app = CreateMySignApp()
```

Code Explanation:

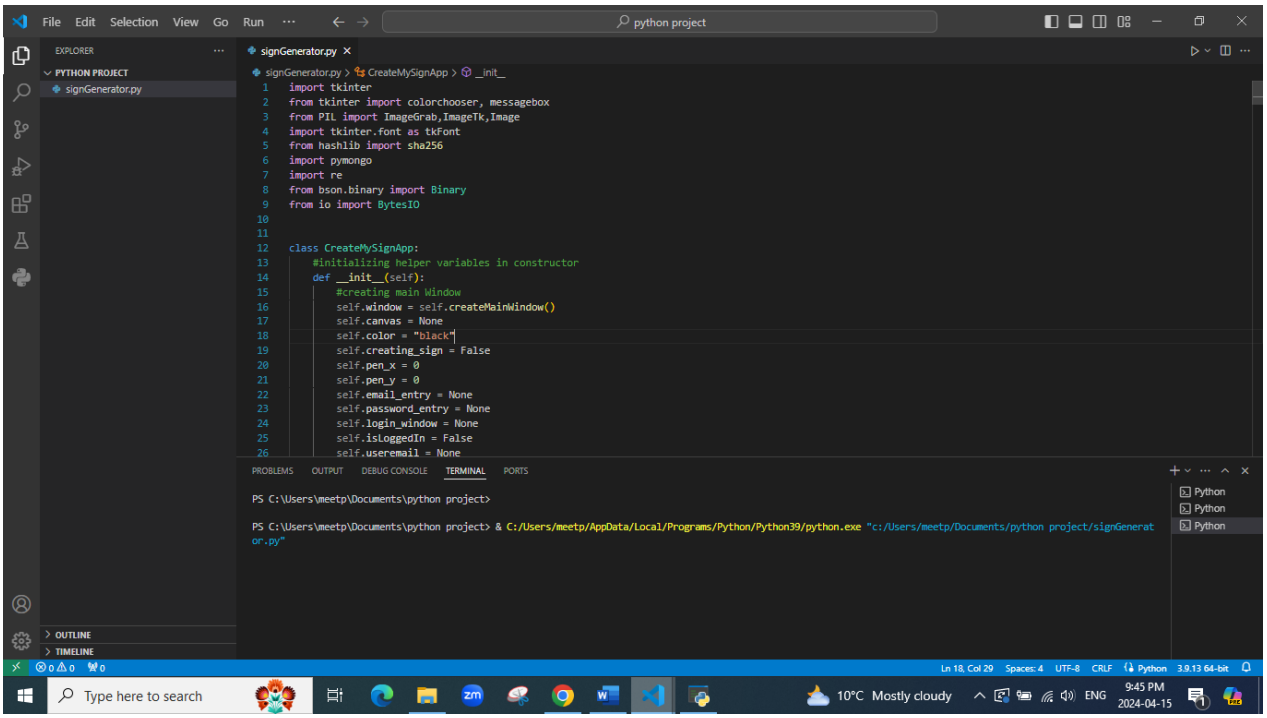
Class CreateMySignApp: This class encapsulates the entire application, managing its graphical user interface (GUI) and interaction with users.

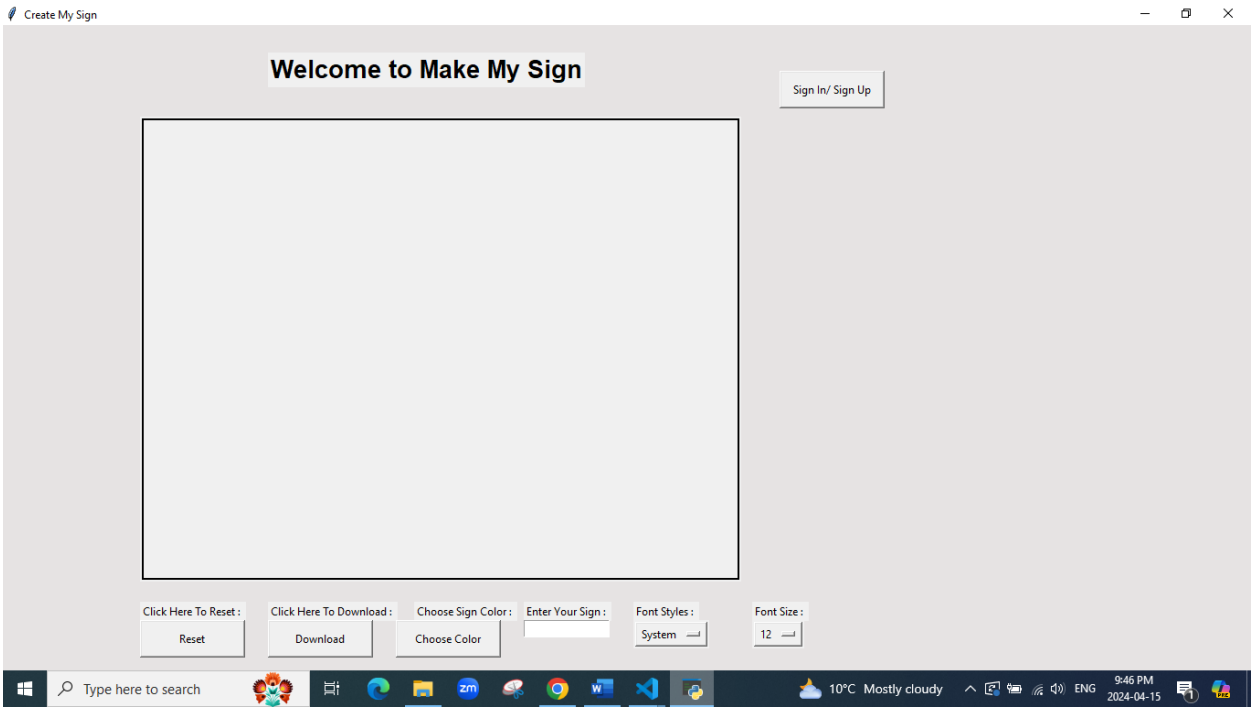
- **Constructor (__init__):** Initializes essential attributes such as the main window, canvas, color, and login status.
- **createMainWindow():** Sets up the main window of the application with appropriate properties like title, background color, and size.
- **destroyWindow():** Handles the proper closure of the application window and associated resources like the MongoDB client upon window destruction.
- **createGUI():** Orchestrates the creation of the entire graphical interface by invoking helper methods for canvas creation, button placement, and input fields.
- **createCanvas():** Creates a canvas widget where users can draw their signatures or input text.
- **createButtons():** Sets up buttons for functionalities like downloading, resetting, logging in, choosing colors, etc., binding them to their respective actions.
- **createLoginFrame():** Generates a separate window for user authentication or account creation, providing necessary entry fields and buttons.
- **getDBConnection():** Establishes a connection to the MongoDB database and returns the database object for further operations.
- **login():** Validates user credentials against the database and controls the login/logout functionality based on the authentication result.
- **createAccount():** Handles the creation of user accounts, ensuring password criteria compliance and securely storing hashed passwords in the database.
- **chooseColor():** Opens a color chooser dialog, allowing users to select the color for their signatures and updating the canvas accordingly.
- onCanvasClick() and onCanvasMotion(): Manage user interactions with the canvas, enabling them to draw signatures freely using the mouse.
- onDownloadClick(): Captures the signature drawn on the canvas, saves it as an image file, and optionally stores it in the database if the user is logged in.
- onResetClick(): Clears the canvas, enabling users to start over by removing existing drawings or text.
- getInputValue(): Retrieves text input from the entry field and displays it on the canvas using the selected font style and size.

- **storeSignImage():** Stores the signature image in the MongoDB database, associating it with the user's email for future retrieval.
- **retrieveSigns() and displaySigns():** Retrieve and display previously saved signature images from the database in a separate window for user reference.
- **__del__():** Ensures proper closure of the MongoDB client upon deletion of the application object, preventing resource leaks.

User Manual:

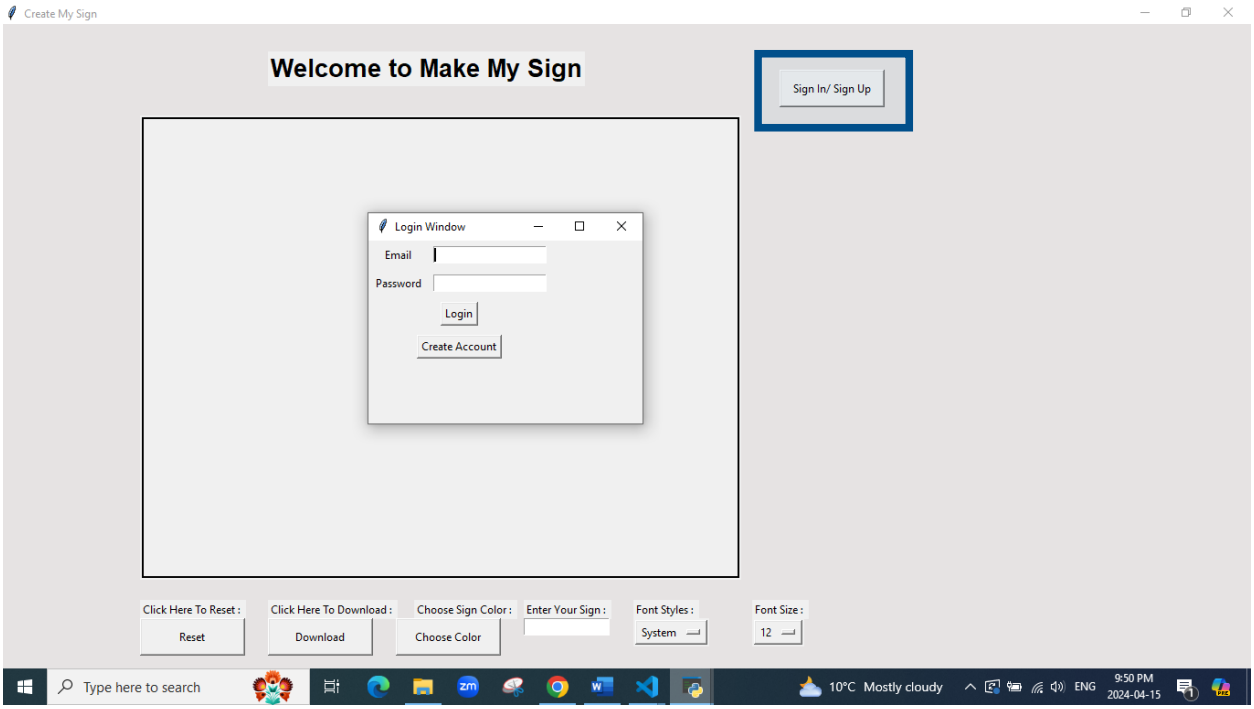
- **Step 1:** Installation and Setup Python and MongoDB as per installation guide and use pip, the package installer for Python, to install the necessary libraries.
- **Step 2:** Open Python and run the project python file after executing the command, the main window of the application titled "Create My Sign" will appear on your screen.



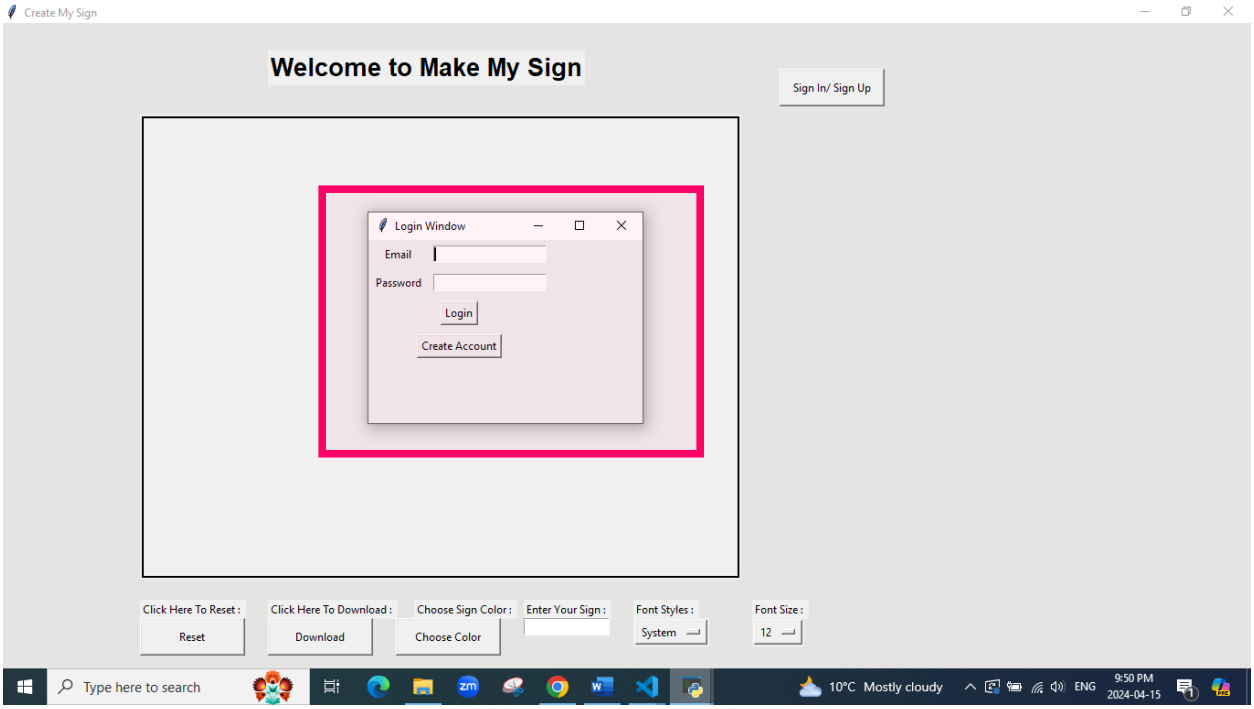


User Authentication

- **Sign In or Sign Up:** we create our account by clicking on the sign In/sign Up button.



When we click on the button, we see another login window

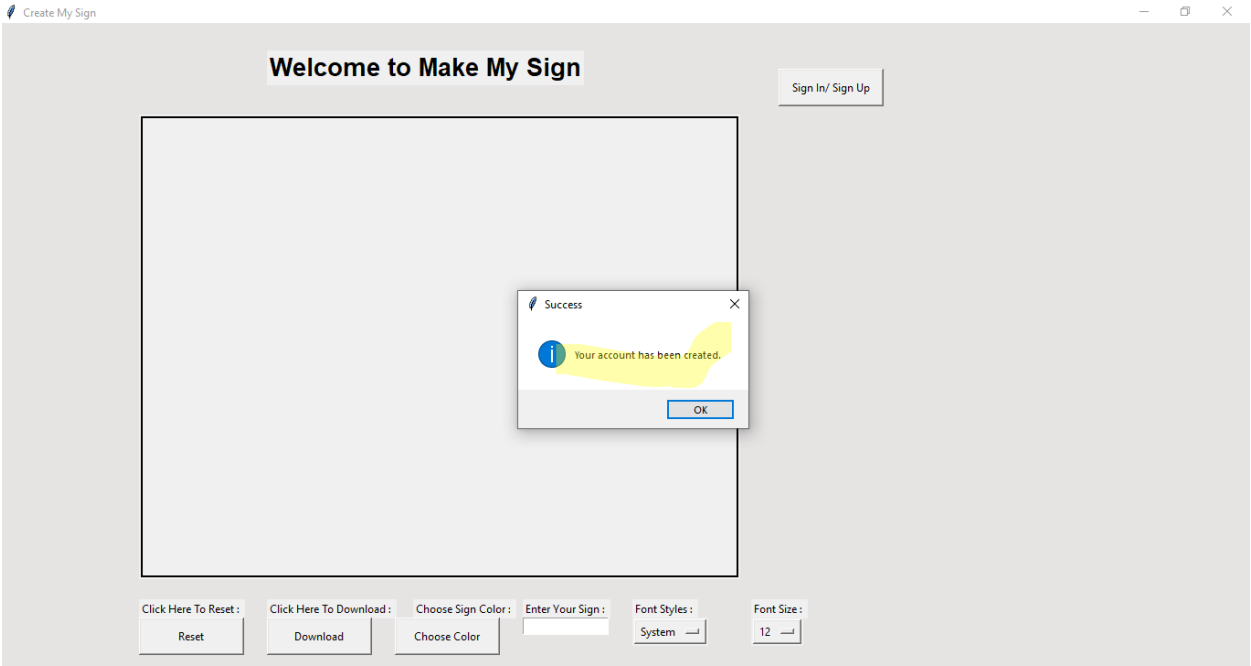


- **Sign In or Sign Up:** If you already have an account, click on the "Login" button. Otherwise, click the “create account” button to create a new account.
- We added password criteria. One special character, one cap letter and a minimum of 8 characters.

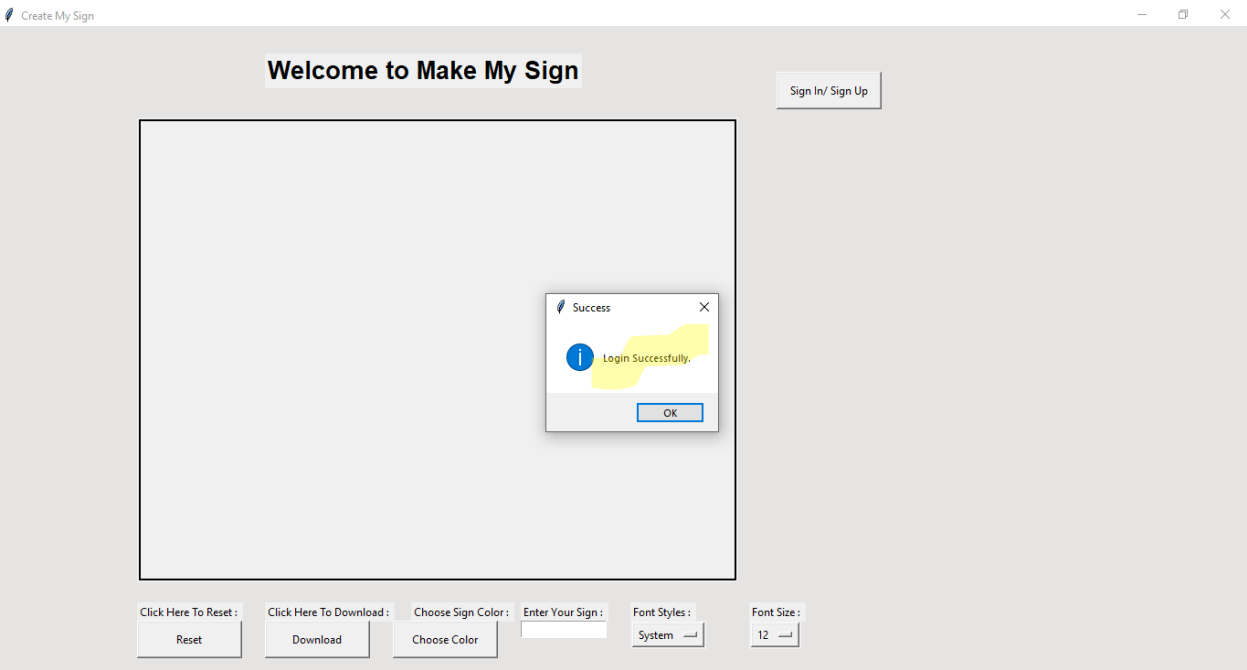

Create My Sign
Login Window
Email
meet@123.com
Password
Login
Create Account
ome to Make My Sign
Sign In/ Sign Up
Error
Password should be at least 8 characters long!
OK
Click Here To Reset :
Reset
Click Here To Download :
Download
Choose Sign Color :
Choose Color
Enter Your Sign :
Font Styles :
System
Font Size :
12


Create My Sign
Login Window
Email
meet@123.com
Password
Login
Create Account
ome to Make My Sign
Sign In/ Sign Up
Error
Password should have at least one uppercase letter!
OK
Click Here To Reset :
Reset
Click Here To Download :
Download
Choose Sign Color :
Choose Color
Enter Your Sign :
Font Styles :
System
Font Size :
12

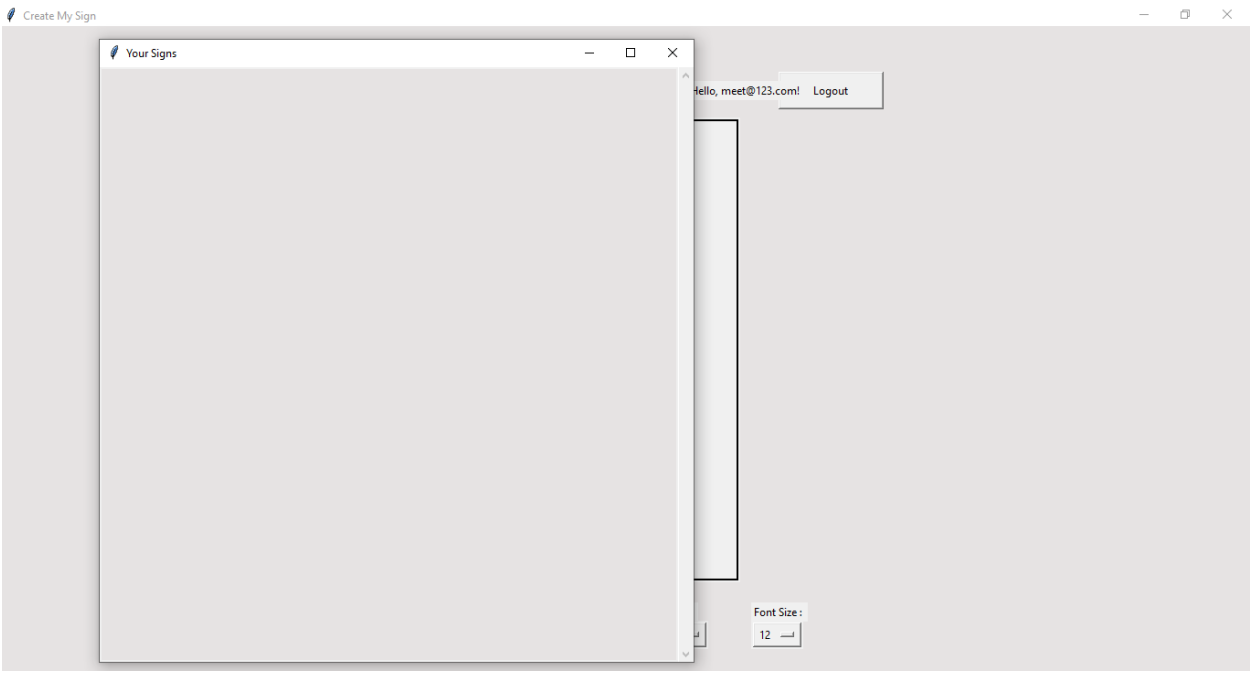
When you Insert proper password and click to create account button then you see success message.



Now you need to Log in by clicking the same sign in/sign up button and inserting same details as used when created account.

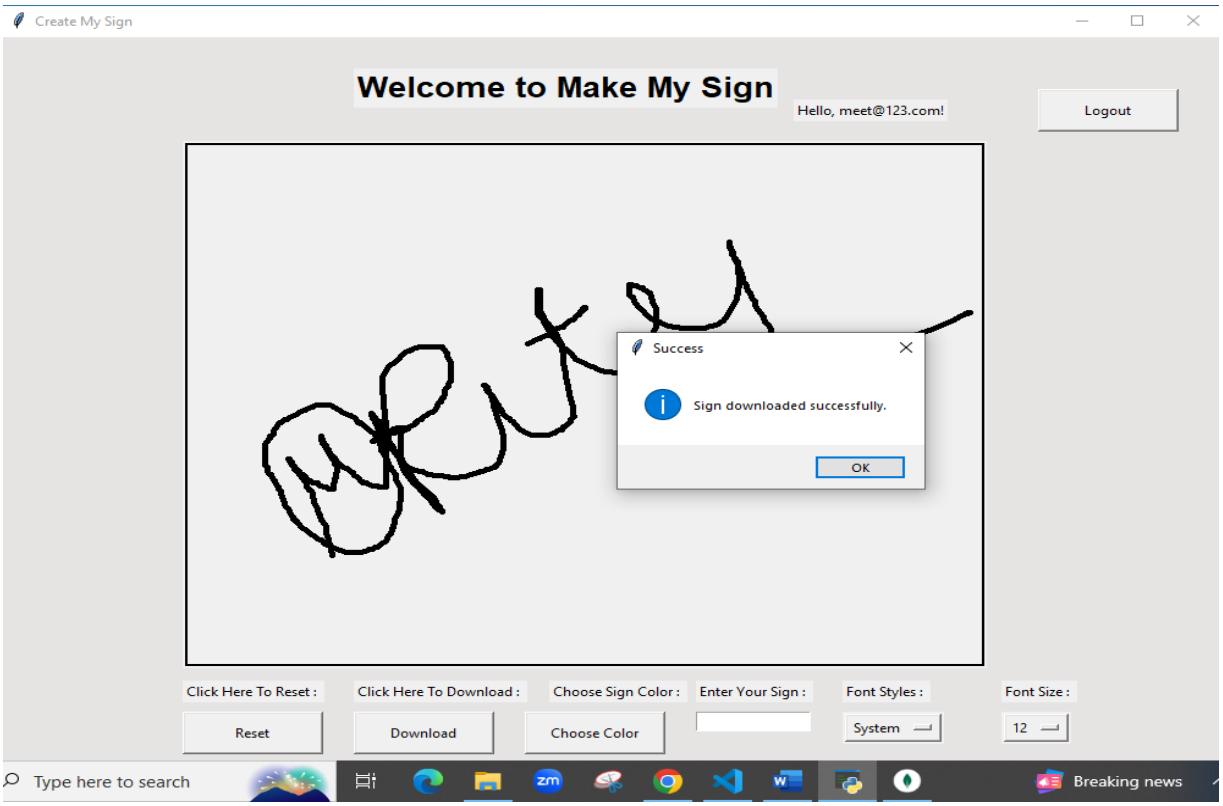


You can see login successfully message then open your signs window which show your all-pervious signs.



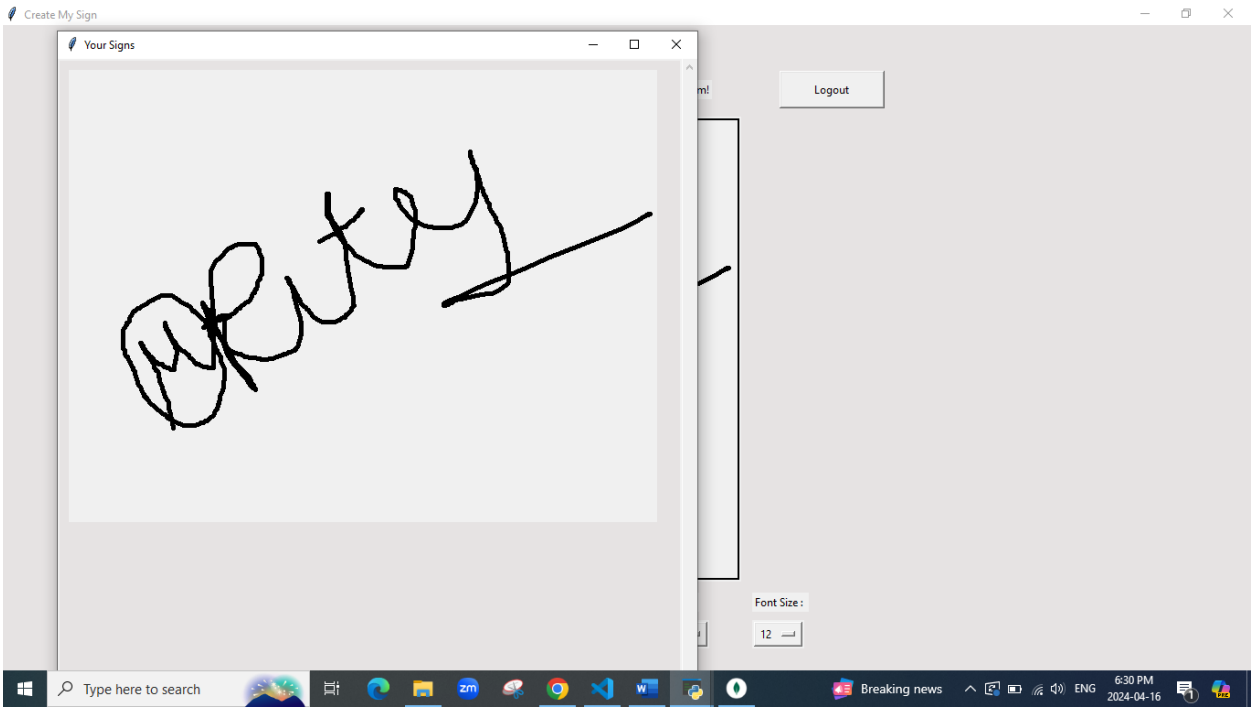
Here you can see a welcome message when you successfully log in.

Drawing a Signature: Use the mouse to draw your signature on the canvas area. Click and drag to draw lines freely.

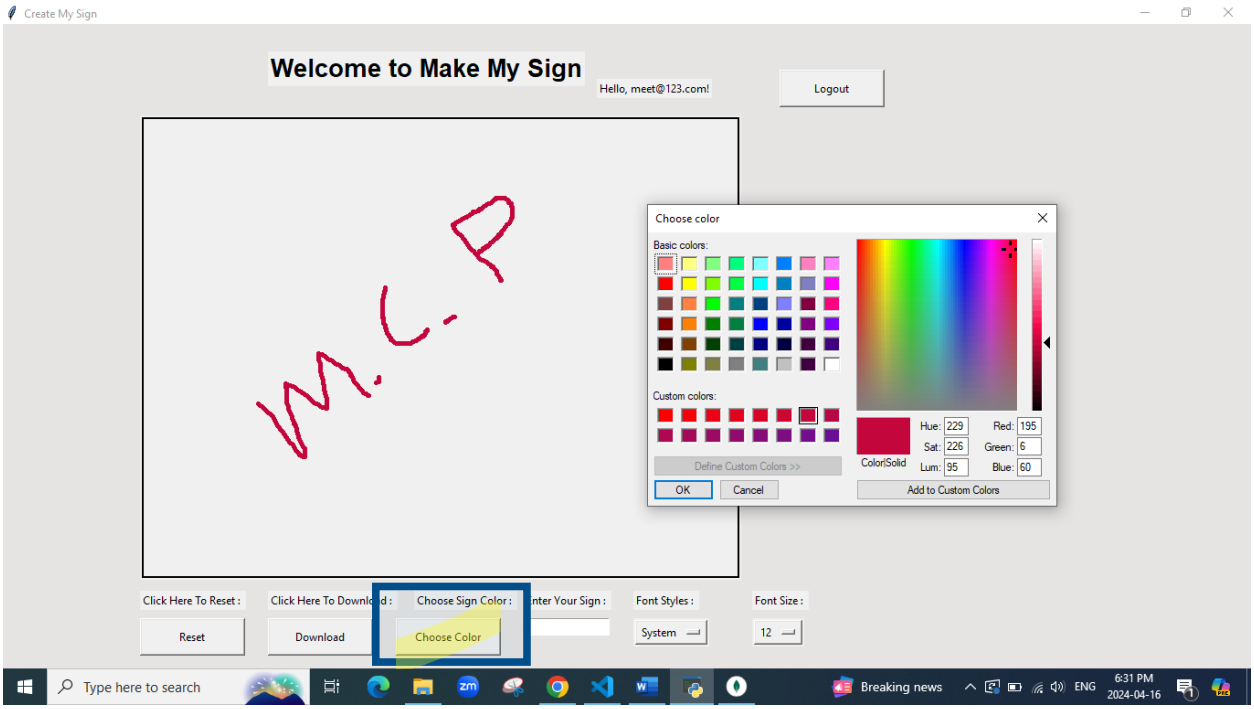


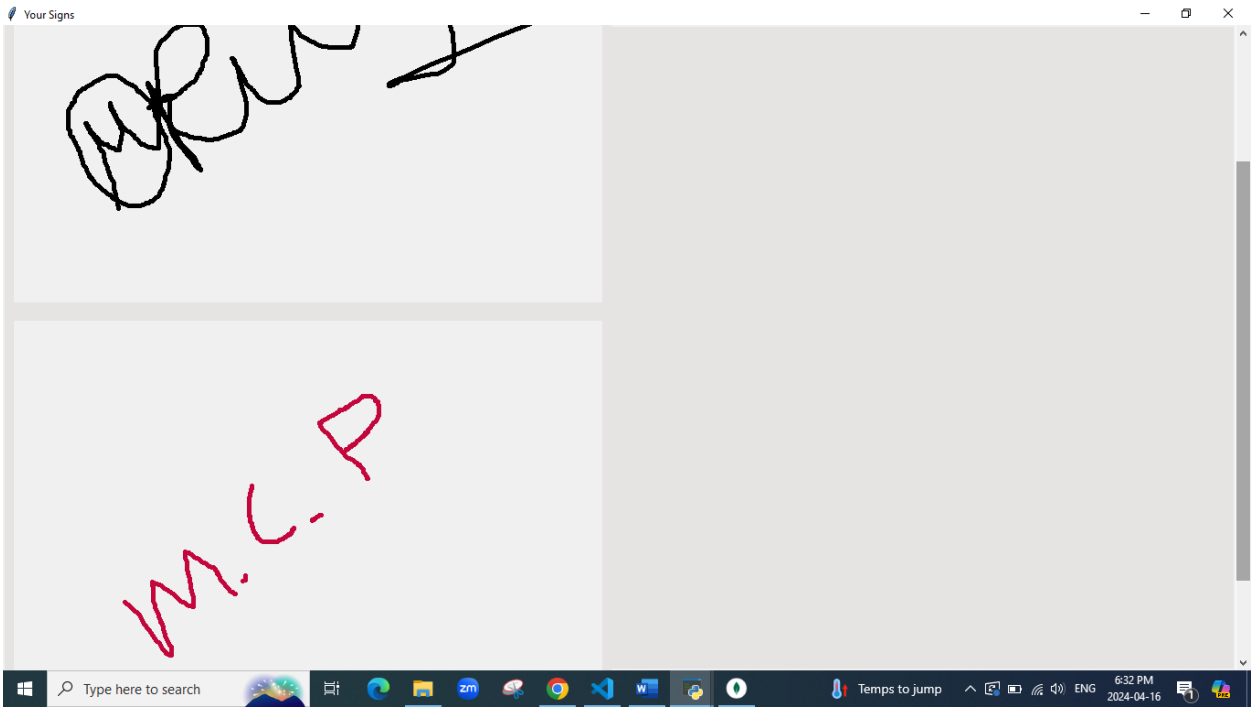
Download Button: Once you are satisfied with your signature, click on the "Download" button. This will save your signature as an image file (sign.png) on your system.

Then open Your sign window in that window we see our all-downloaded sign.



Using Chose color button we can be able to change the color of pen to sign in different colors.

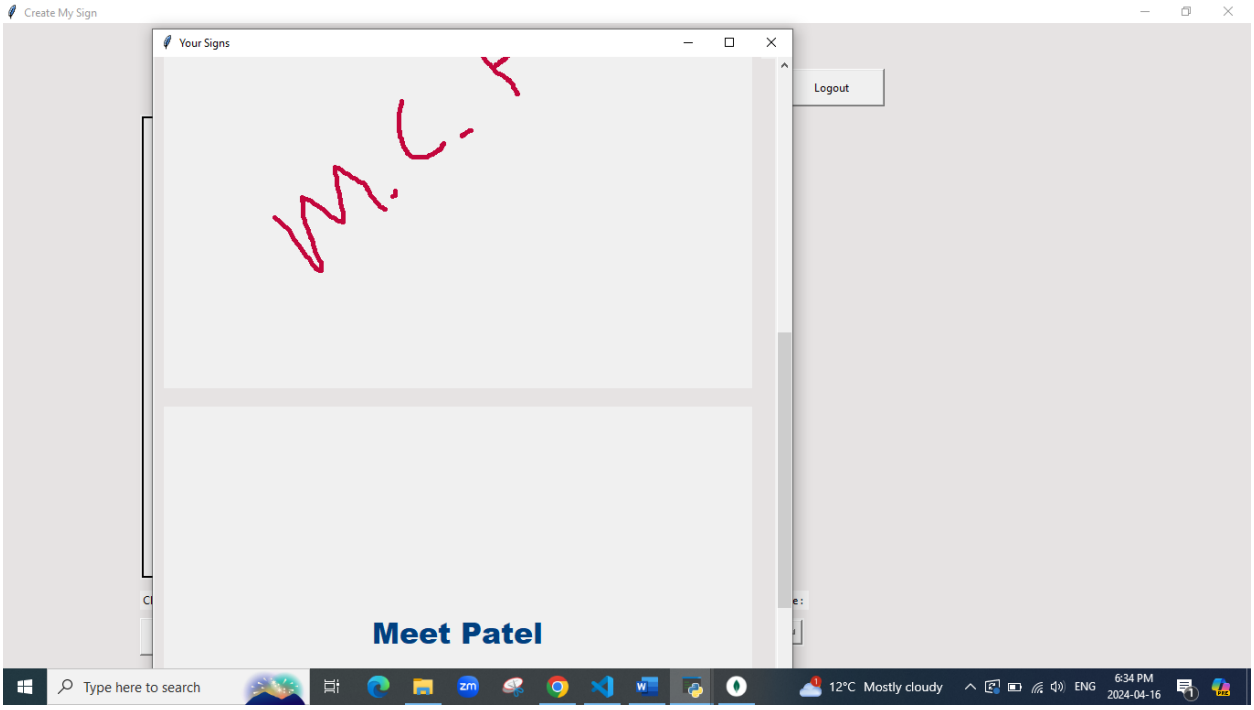




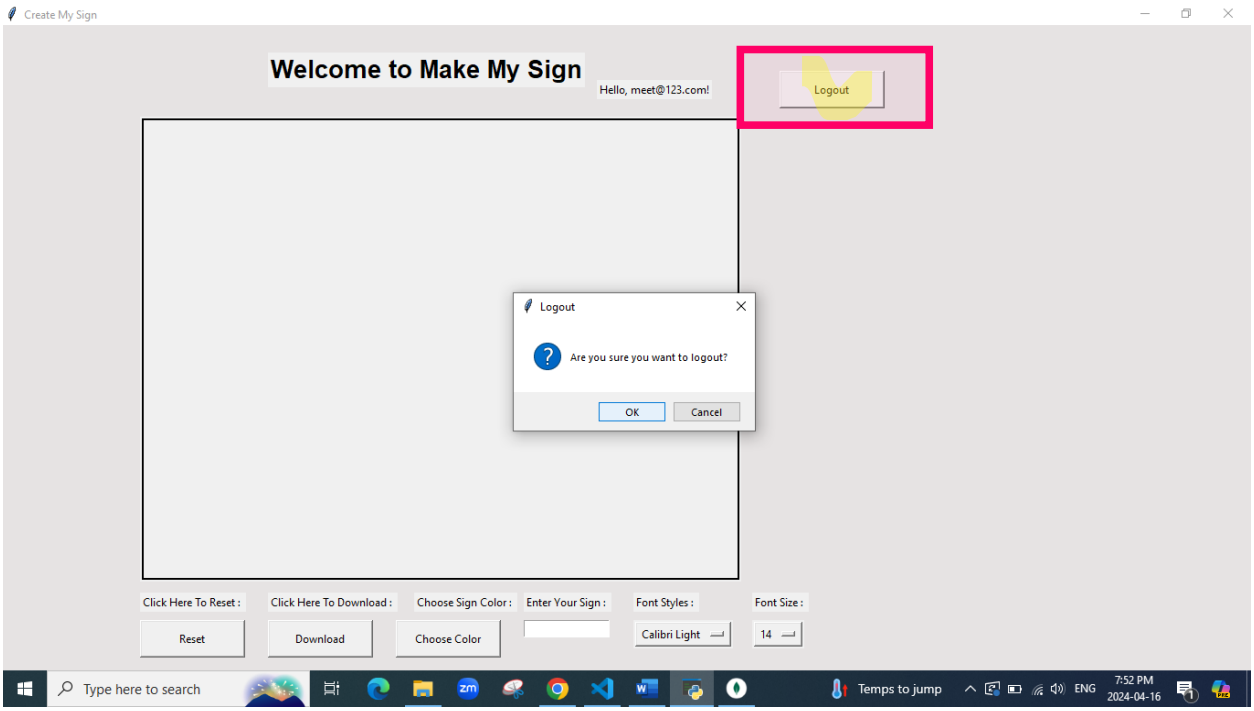
We can see that both signs are updated into your sign window.

Entering Text: Alternatively, you can enter text in the "Enter Your Sign" field. The entered text will appear on the canvas with default font and size.

Customizing Sign: Choose your preferred font style and size from the dropdown menus. Additionally, click on the "Choose Color" button to select the color for your signature.



Logout: To logout from your account, click on the "Logout" button. This will clear the canvas and reset the application to its initial state.


Create My Sign
Welcome to Make My Sign
Hello, meet@123.com!
Logout
Logout
Are you sure you want to logout?
OK
Cancel
Click Here To Reset :
Click Here To Download :
Choose Sign Color :
Enter Your Sign :
Font Styles :
Font Size :
Reset
Download
Choose Color
Calibri Light
14
Type here to search
Temps to jump
ENG
7:52 PM
2024-04-16

Challenges and Solutions:

GUI Layout Management

Challenge:
Managing the layout in a GUI application can be challenging, especially when scaling the interface to different window sizes or adding multiple components.

Solution:
Utilized the pack, grid, and place geometry managers from Tkinter effectively to organize the layout. Choose a place for precise control over the placement of elements, ensuring consistency across different screen resolutions. Responsive design principles were applied where possible to maintain usability.

Image Processing Capabilities

Challenge:
Integrating robust image processing capabilities for features such as adding text to images, adjusting font sizes, and saving the final product without losing quality.

Solution:
Incorporated the Python Imaging Library (PIL) to handle image processing tasks. Developed custom functions to dynamically adjust text placement and size based on user input, ensuring high-quality outputs. Additionally, optimized image handling routines to minimize memory usage and processing time.

Future Enhancement:

In the future, we aim to enhance CreateMySignApp by developing mobile versions, enabling cloud storage for user designs, and introducing new design tools and templates. Additionally, we plan to expand language support, implement collaborative features, and improve accessibility. We also intend to integrate direct social media sharing capabilities to increase the app’s versatility and user engagement. These improvements will make the app more accessible and enjoyable for a wider audience.

Conclusion:

In conclusion, CreateMySignApp offers users a simple yet powerful tool for creating personalized digital signatures. Through its intuitive interface and robust features, users can easily design and download their unique signatures. The app provides convenience and flexibility, allowing users to customize their signatures with various fonts, colors, and styles. Users can access their signatures any time by log in into the app.

Thanks!