User Customisable VOTING POLL

Done by

Yuhanth P

XII - JEE

INDEX

S.No.	Title	Page No.
1	Feasibility Study	1
2	Errors and their types	2
3	Testing	3
4	Maintenance	4
5	Project Synopsis	5
6	Project Details	6
7	System Requirements	6
8	Flowchart	7
9	Code	8
10	Output	47
11	Bibliography	60

FEASIBILITY STUDY

This feasibility study evaluates the development of polling software designed for businesses, educational institutions, and event organizers. The software aims to facilitate real-time feedback collection and decision-making. Initial assessments indicate strong market demand and potential profitability.

ECONOMIC FEASIBILITY

This economic feasibility study evaluates the financial viability of developing polling software targeted at businesses, educational institutions, and event organizers.

TECHNICAL FEASIBILITY

Technical feasibility centres on the existing computer system and its ability to support the proposed task. This involves financial consideration to accommodate technical enhancements. It is technically feasible because the technology needed to develop this software is easily available.

ERRORS AND THEIR TYPES

An error, sometimes called “A BUG” is anything in the code that prevents a program from compiling and running correctly. There are broadly three types of errors as follows:

1. Compile-time errors: Errors that occur during the compilation of a program are called compile-time errors. It has two types as follows:

a. Syntax error: It refers to formal rules governing the construction of valid statements in a language.

b. Semantics error: It refers to the set of rules which give the meaning of a statement.

2. Run-time Errors: Errors that occur during the execution of the program are run-time errors. Some run-time error stops the execution of the program which is then called program “Crashed”.

3. Logical Errors: Sometimes, even if you don’t encounter any error during compiling time and runtime, your program does not provide the correct result. This is because of the programmer’s mistaken analysis of the problem he or she is trying to solve. Such errors are called logical errors.

TESTING

Alpha Testing: The objective of this testing is to identify all possible issues or defects before releasing it into the market or to the user. It is conducted at the developer’s site.

Beta Testing: It is a formal type of software testing which is carried out by the customers. It is performed in a real environment before releasing the products into the market for the actual end-users. It is carried out to ensure that there are no major failures in the software or product and that it satisfies the business requirement. Beta Testing is successful when the customer accepts the software.

White Box Testing: It is software testing based on the knowledge of the internal logic of an application’s code. It is also known as Glass box Testing. Internal Software and code working should be known for performing this type of testing. These tests are based on the coverage of the code statements, branches, paths, conditions etc.

Black Box Testing: It is a software testing, method in which the internal structure or design of the item to be tested is not known to the tester. This method of testing can be applied virtually to every level of software testing.

MAINTENANCE

Programming maintenance refers to the modifications in the program. After it has been completed, to meet changing requirements or to take care of the errors that show up. There are four types of maintenance:

Corrective Maintenance:

When the program after compilation shows errors because of some unexpected situations, untested areas such errors are fixed by Corrective maintenance.

Adaptive Maintenance:

Changes in the environment in which an information system operates may lead to system management. To accommodate changing needs from time-to-time maintenance is done and is called Adaptive maintenance.

Preventive Maintenance:

If possible, errors could be anticipated before they occur, called preventive maintenance.

Perfective Maintenance:

In this rapidly changing world, information technology is the fastest-growing area. If the existing system is maintained to keep tuned with new features, facilities, and capabilities, it is said to be Perfective Maintenance.

PROJECT SYNOPSIS

Project Title: User customizable VOTING POLL

Problem definition:

Organizations today struggle to gather timely and actionable feedback due to inefficient traditional polling methods. Existing digital solutions often lack customization, real-time analytics, and user-friendly interfaces, leading to underutilization.

Reason for choosing this:

I wanted to create a user-customizable polling software with GUI allowing users to create and conduct polls.

Objective:

To create, host locally, and retrieve results of a poll.

Limitation:

It only functions on local computers, necessitating the consolidation of results from multiple devices after polling is completed.

PROJECT DETAILS

The poll software was developed by **Yuhanth** using Python.

Modules used:

- OS
- SYS
- CSV
- Pickle
- Random
- Tkinter
- Matplotlib
- Datetime
- Cryptography

SYSTEM REQUIREMENTS

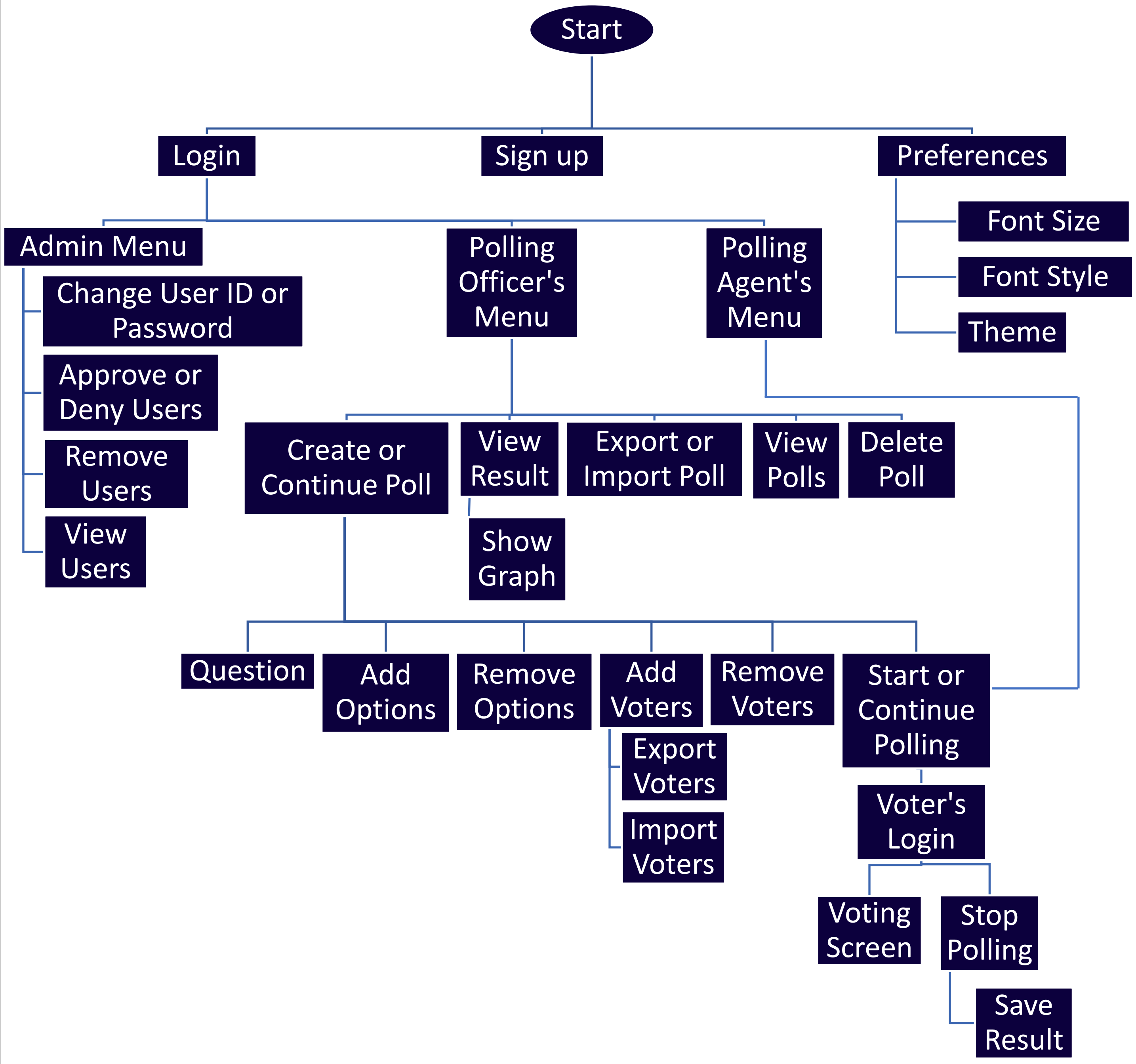
Hardware:

- A computer or laptop
- 2 GB RAM or higher
- Any processor
- At least 1 GB of free space

Software:

- Any operating system able to run .exe files
- Poll.exe file

Flowchart



Start
Login
Sign up
Preferences
Font Size
Font Style
Theme
Admin Menu
Change User ID or Password
Approve or Deny Users
Remove Users
View Users
Polling Officer's Menu
Polling Agent's Menu
Create or Continue Poll
View Result
Show Graph
Export or Import Poll
View Polls
Delete Poll
Question
Add Options
Remove Options
Add Voters
Export Voters
Import Voters
Remove Voters
Start or Continue Polling
Voter's Login
Voting Screen
Stop Polling
Save Result

Code

```
import os
import sys
import csv
import pickle
import random
import tkinter as TK
from tkinter import
messagebox,Toplevel,ttk,Tk,Entry,Button,Label,OptionMenu,filedialog,Frame,Listbox
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.ticker import MaxNLocator
from datetime import datetime as DT
from cryptography.fernet import Fernet
def MAIN():
    global key,main_dir,LoggedIn_Flag,current_poll
    app_dir=os.getcwd()
    if not os.path.exists('YUHANTH Poll_Project_Data'):
        os.mkdir('YUHANTH Poll_Project_Data')
        main_dir=app_dir+'\\YUHANTH Poll_Project_Data'
        os.chdir(main_dir)
        with open("secret.key", "wb") as key_file:
            key_file.write(Fernet.generate_key())
    main_dir=app_dir+'\\YUHANTH Poll_Project_Data'
    os.chdir(main_dir)
    try:
        key=open("secret.key", "rb").read()
    except:
        with open("secret.key", "wb") as key_file:
            key_file.write(Fernet.generate_key())
        key=open("secret.key", "rb").read()
        with open('Preferences.dat','wb') as f:
            pickle.dump(encrypt_msg(str([5,'Arial','Default','admin','admin'])),f)
    if not os.path.exists('NonApprovedUsers.csv'):
        open('NonApprovedUsers.csv','x').close()
    if not os.path.exists('User_Details.csv'):
        open('User_Details.csv','x').close()
    if not os.path.exists('Preferences.dat'):
        with open('Preferences.dat','wb') as f:
            pickle.dump(encrypt_msg(str([5,'Arial','Default','admin','admin'])),f)
    LoggedIn_Flag=False
    current_poll=None
    UserDictRefresh()
    FontAndThemeRefresher()
    TkinWelcome()
def encrypt_msg(message):
    fernet = Fernet(key)
    encrypted_message = fernet.encrypt(message.encode())
    return encrypted_message
def decrypt_msg(encrypted_message):
    fernet = Fernet(key)
    decrypted_message = fernet.decrypt(encrypted_message).decode()
    return decrypted_message
def PreferenceLoader():
    global fontsize,fontstyle,theme,AdminUID,AdminPass
    with open('Preferences.dat','rb') as f:
        try:
            PreferenceList=eval(decrypt_msg(pickle.load(f)))
        except:
            PreferenceList=[5,'Arial','Default','admin','admin']
    fontsize=PreferenceList[0]
    fontstyle=PreferenceList[1]
    theme=PreferenceList[2]
    AdminUID=PreferenceList[3]
    AdminPass=PreferenceList[4]
def PreferenceWriter():
    os.chdir(main_dir)
    with open('Preferences.dat','wb') as f:
        pickle.dump(encrypt_msg(str([fontsize,fontstyle,theme,AdminUID,AdminPass])
),f)
def FontAndThemeRefresher():
    global
MainTitleTextFont,TitleTextFont,TextFont1,TextFont2,TextFont3,TextFont4,ButtonFont
1,ButtonFont2,ButtonFont2Bold,ButtonFont3,ColourCodeMain,ColourCodeButtonBG,Colour
CodeButtonFG,ColourCodeText
    PreferenceLoader()
    MainTitleTextFont=(fontstyle,fontsize*12,'bold')
    TitleTextFont=(fontstyle,fontsize*12,'bold')
    TextFont1=(fontstyle,fontsize*8)
    TextFont2=(fontstyle,fontsize*6)
    TextFont3=(fontstyle,fontsize*5)
    TextFont4=(fontstyle,fontsize*4)
    ButtonFont1=(fontstyle,fontsize*8)
    ButtonFont2=(fontstyle,fontsize*6)
    ButtonFont2Bold=(fontstyle,fontsize*6,'bold')
    ButtonFont3=(fontstyle,fontsize*5)
    if theme=='Default':
        ColourCodeMain='#0C003D'
        ColourCodeButtonBG='#92C7CF'
        ColourCodeButtonFG='#000000'
        ColourCodeText='#FFFFFF'
    elif theme=='Light':
        ColourCodeMain='#5e5e5e'
        ColourCodeButtonBG='#000000'
        ColourCodeButtonFG='#5e5e5e'
```

```
            ColourCodeText='#000000'
        elif theme=='Dark':
            ColourCodeMain='#000000'
            ColourCodeButtonBG='#ababab'
            ColourCodeButtonFG='#000000'
            ColourCodeText='#ababab'
    def IncreaseFontSize():
        global fontsize
        fontsize+=1
        PreferenceWriter()
        FontAndThemeRefresher()
        Preferences_Window.destroy()
        TkinPreferences()
    def DecreaseFontSize():
        global fontsize
        fontsize=max(2,fontsize-1)
        PreferenceWriter()
        FontAndThemeRefresher()
        Preferences_Window.destroy()
        TkinPreferences()
    def ChangeFontStyle(*a):
        global fontstyle,theme
        fontstyle=FontStyle_var.get()
        theme=Theme_var.get()
        PreferenceWriter()
        FontAndThemeRefresher()
        Preferences_Window.destroy()
        TkinPreferences()
    def UserDictRefresh():
        global users_dict,NonApprovedUsers_Dict
        os.chdir(main_dir)
        users_dict={}
        NonApprovedUsers_Dict={}
        with open('User_Details.csv') as f:
            r=csv.reader(f)
            for i in r:
                users_dict[decrypt_msg(eval(i[0]))]=[decrypt_msg(eval(i[1])),decrypt_m
sg(eval(i[2])),decrypt_msg(eval(i[3]))]
        with open('NonApprovedUsers.csv') as f:
            r=csv.reader(f)
            for i in r:
                NonApprovedUsers_Dict[decrypt_msg(eval(i[0]))]=[decrypt_msg(eval(i[1])
),decrypt_msg(eval(i[2]))]
    def SignUp():
        UserDictRefresh()
        name_flag=u_ID_flag=pass_flag=False
        name=name_entry.get()
        if len(name)==0 or name.isspace():
            messagebox.showerror('Error','Field can\'t be empty(Name)')
        else:
            name_flag=True
        u_ID=userID_entry.get()
        if len(u_ID)==0 or u_ID.isspace():
            messagebox.showerror('Error','Field can\'t be empty(User ID)')
        else:
            u_ID_flag=True
        if u_ID in users_dict or u_ID in NonApprovedUsers_Dict or u_ID==AdminUID:
            messagebox.showerror('Error','User ID already exists('+u_ID+')')
            u_ID_flag=False
        pass_word=password_entry.get()
        if len(pass_word)==0 or pass_word.isspace():
            messagebox.showerror('Error','Field can\'t be empty(Password)')
        else:
            pass_flag=True
        pass_word_c=confirm_password_entry.get()
        if name_flag==True and u_ID_flag==True and pass_flag==True:
            if pass_word==pass_word_c:
                with open('NonApprovedUsers.csv','a',newline='') as f:
                    w=csv.writer(f)
                    w.writerow([encrypt_msg(str(u_ID)),encrypt_msg(str(pass_word)),enc
rypt_msg(str(name))])
                    SignUpWindow.destroy()
                    messagebox.showinfo('Signed up successfully','Signed up
successfully ('+name+')\nPlease wait for the admin to approve')
                    WelcomeWindow.deiconify()
            else:
                messagebox.showerror('Error','Passwords do not match')
def LogIn():
    global LogIn_user_ID,LoggedIn_Flag,current_PASS,current_designation
    UserDictRefresh()
    if not os.path.exists('User_Details.csv'):
            open('User_Details.csv','x').close()
    with open('User_Details.csv') as f:
            r=csv.reader(f)
            for i in r:
                users_dict[i[0]]=[i[1],i[2],i[3]]
    if LoggedIn_Flag==False:
        LogIn_user_ID=UserID_entry.get()
        password=Password_entry.get()
        if len(LogIn_user_ID)==0 or LogIn_user_ID.isspace():
            messagebox.showerror('Error','Field can\'t be empty(User ID)')
        elif LogIn_user_ID==AdminUID and password==AdminPass:
            login_button.config(state=TK.DISABLED)
            LoginWindow.withdraw()
            LoggedIn_Flag=True
            LoginWindow.unbind('<Return>')
            TkinAdminMenu()
        elif LogIn_user_ID==AdminUID:
            messagebox.showerror('Log','Incorrect password')
        elif LogIn_user_ID in users_dict:
            if password==users_dict[LogIn_user_ID][0]:
                current_PASS=users_dict[LogIn_user_ID][0]
                LoginWindow.withdraw()
                if users_dict[LogIn_user_ID][2]=='Polling officer':
                    login_button.config(state=TK.DISABLED)
                    LoggedIn_Flag=True
                    LoginWindow.unbind('<Return>')
                    POfficerMenu()
                elif users_dict[LogIn_user_ID][2]=='Polling agent':
                    login_button.config(state=TK.DISABLED)
                    LoggedIn_Flag=True
                    LoginWindow.unbind('<Return>')
                    TkinPAgent()
                else:
                    messagebox.showerror('Error','Designation error')
                    LoginWindow.deiconify()
                current_designation=users_dict[LogIn_user_ID][2]
            else:
                messagebox.showerror('Log','Incorrect password')
        elif LogIn_user_ID in NonApprovedUsers_Dict:
            messagebox.showerror('Log','Wait for Admin to approve or deny your
request')
        else:
            messagebox.showerror('Log','**No such user ('+LogIn_user_ID+')**')
def ChangeUIDAdmin():
    global AdminUID
    UserDictRefresh()
    NewUID_flag=CurrentPass_flag=False
    NewUID=NewUID_entry.get()
    if len(NewUID)==0 or NewUID.isspace():
        messagebox.showerror('Error','Field can\'t be empty (Current password)')
    else:
        NewUID_flag=True
    CurrentPass=CurrentPassword_entry.get()
    if len(CurrentPass)==0 or CurrentPass.isspace():
        messagebox.showerror('Error','Field can\'t be empty (New password)')
    else:
        CurrentPass_flag=True
    if NewUID_flag==True and CurrentPass_flag==True:
        if CurrentPass==AdminPass:
            if NewUID not in users_dict and NewUID not in NonApprovedUsers_Dict:
                AdminUID=NewUID
                PreferenceWriter()
                PreferenceLoader()
                ChangeUIDWindow.destroy()
                messagebox.showinfo('Successfully changed','User ID changed 
successfully')
                Admin_window.deiconify()
            else:
                messagebox.showerror('Error','User ID already exists('+NewUID+')')
        else:
            messagebox.showerror('Error','Current password incorrect')
def ChangePassAdmin():
    global AdminPass
    CurrentPass_flag=NewPass_flag=False
    CurrentPass=CurrentPassword_entry.get()
    if len(CurrentPass)==0 or CurrentPass.isspace():
        messagebox.showerror('Error','Field can\'t be empty (Current password)')
    else:
        CurrentPass_flag=True
    NewPass=NewPassword_entry.get()
    if len(NewPass)==0 or NewPass.isspace():
        messagebox.showerror('Error','Field can\'t be empty (New password)')
    else:
        NewPass_flag=True
    ReNewPass=ReNewPassword_entry.get()
    if CurrentPass_flag==True and NewPass_flag==True:
        if CurrentPass==AdminPass:
            if NewPass==ReNewPass:
                AdminPass=NewPass
                PreferenceWriter()
                PreferenceLoader()
                ChangePassWindow.destroy()
                messagebox.showinfo('Password changed successfully','Password 
changed successfully')
                Admin_window.deiconify()
            else:
                messagebox.showerror('Error','New passwords do not match')
        else:
            messagebox.showerror('Error','Current password incorrect')
def ApproveUser():
    UserDictRefresh()
    try:
        selected_index = Non_Approved_user_listbox.curselection()[0]
        Approve_UID = Non_Approved_user_listbox.get(selected_index)
        designation=designation_var.get()
        if designation=='Polling officer' or designation=='Polling agent':
            password=NonApprovedUsers_Dict[Approve_UID][0]
            name=NonApprovedUsers_Dict[Approve_UID][1]
            with open('User_Details.csv','a',newline='') as f:
                w=csv.writer(f)
                w.writerow([encrypt_msg(Approve_UID),encrypt_msg(password),encrypt
_msg(name),encrypt_msg(designation)])
            NonApprovedUsers_Dict.pop(Approve_UID)
            with open('NonApprovedUsers.csv','w',newline='') as f:
                w=csv.writer(f)
                for i in NonApprovedUsers_Dict:
                    w.writerow([encrypt_msg(i),encrypt_msg(NonApprovedUsers_Dict[i
][0]),encrypt_msg(NonApprovedUsers_Dict[i][1])])
            messagebox.showinfo('Successfull','Successfully approved')
            ApproveUserWindow.destroy()
            TkinApproveUser()
        else:
            messagebox.showerror('Error','Designation not selected')
    except IndexError:
        messagebox.showerror('Error','No user selected')
def DenyUser():
    UserDictRefresh()
    try:
        selected_index = Non_Approved_user_listbox.curselection()[0]
        Deny_UID = Non_Approved_user_listbox.get(selected_index)
        NonApprovedUsers_Dict.pop(Deny_UID)
        with open('NonApprovedUsers.csv','w',newline='') as f:
            w=csv.writer(f)
            for i in NonApprovedUsers_Dict:
                w.writerow([encrypt_msg(i),encrypt_msg(NonApprovedUsers_Dict[i][0]
),encrypt_msg(NonApprovedUsers_Dict[i][1])])
        messagebox.showinfo('Successfull','Successfully denied')
        ApproveUserWindow.destroy()
        TkinApproveUser()
    except IndexError:
        messagebox.showerror('Error','No user selected')
def RemoveUser():
    rem_u_name=Rem_UID_entry.get()
    rem_flag=False
    if len(rem_u_name)==0 or rem_u_name.isspace():
        messagebox.showerror('Error','Field can\'t be empty(Name)')
    else:
        with open('User_Details.csv','w',newline='') as f:
            w=csv.writer(f)
            for i in users_dict:
                if i==rem_u_name:
                    rem_flag=True
                    RemUserWindow.destroy()
                    messagebox.showinfo('Added Successfully','User removed
successfully ('+rem_u_name+')')
                    Admin_window.deiconify()
                    continue
                w.writerow([i,users_dict[i][0],users_dict[i][1],users_dict[i][2]])
        if rem_flag==False:
            messagebox.showerror('Error','User not found')
def POfficerMenu():
    global current_poll
    if not os.path.exists('Polls'):
        os.mkdir('Polls')
    os.chdir(main_dir+'\\Polls')
    TkinPollSelector()
def DisablePollSlectorWindowButtons():
    Logout_button.config(state=TK.DISABLED)
    CreatePoll_button.config(state=TK.DISABLED)
    ContinuePoll_button.config(state=TK.DISABLED)
    ViewResult_button.config(state=TK.DISABLED)
    ImportPoll_button.config(state=TK.DISABLED)
    ExportPoll_button.config(state=TK.DISABLED)
    ViewPolls_button.config(state=TK.DISABLED)
    DeletePoll_button.config(state=TK.DISABLED)
def EnablePollSlectorWindowButtons():
    Logout_button.config(state=TK.NORMAL)
    CreatePoll_button.config(state=TK.NORMAL)
    ContinuePoll_button.config(state=TK.NORMAL)
    ViewResult_button.config(state=TK.NORMAL)
    ImportPoll_button.config(state=TK.NORMAL)
    ExportPoll_button.config(state=TK.NORMAL)
    ViewPolls_button.config(state=TK.NORMAL)
    DeletePoll_button.config(state=TK.NORMAL)
def CreatePoll():
    global current_poll
    os.chdir(main_dir+'\\Polls')
    file=filename_entry.get()
    if len(file)==0 or file.isspace():
        DisablePollSlectorWindowButtons()
        messagebox.showerror('Error','Field can\'t be empty (Poll_Name)')
        EnablePollSlectorWindowButtons()
    else:
        try:
            os.mkdir(file)
            os.chdir(file)
            current_poll=file
            Creation_Time=DT.now().strftime('%d %B %Y')+' at 
'+DT.now().strftime('%H:%M:%S')
            with open(current_poll+'_Info.txt','w') as f:
                f.write(users_dict[LogIn_user_ID][1]+'\n'+Creation_Time)
            messagebox.showinfo('Created Successfully','Poll created successfully 
('+file+')')
            TkinPOfficerMenu()
        except FileExistsError:
            DisablePollSlectorWindowButtons()
            messagebox.showerror('Error','Poll already exists ('+file+')')
            EnablePollSlectorWindowButtons()
def ContinuePoll():
    global current_poll
    os.chdir(main_dir+'\\Polls')
    file=filename_entry.get()
    if len(file)==0 or file.isspace():
        DisablePollSlectorWindowButtons()
        messagebox.showerror('Error','Field can\'t be empty (Poll_Name)')
        EnablePollSlectorWindowButtons()
    else:
        try:
            os.chdir(file)
            current_poll=file
            if current_designation=='Polling officer':
                TkinPOfficerMenu()
            elif current_designation=='Polling agent':
                PAgent_Window.withdraw()
                TkinVoterLogin()
        except FileNotFoundError:
            if current_designation=='Polling officer':
                DisablePollSlectorWindowButtons()
                messagebox.showerror('Error','Poll does not exist ('+file+')')
                EnablePollSlectorWindowButtons()
            elif current_designation=='Polling agent':
                ContinuePoll_button.config(state=TK.DISABLED)
                messagebox.showerror('Error','Poll does not exist ('+file+')')
                ContinuePoll_button.config(state=TK.NORMAL)
def ViewResult():
    global current_poll
    os.chdir(main_dir+'\\Polls')
    file=filename_entry.get()
    if len(file)==0 or file.isspace():
        DisablePollSlectorWindowButtons()
        messagebox.showerror('Error','Field can\'t be empty (Poll_Name)')
        EnablePollSlectorWindowButtons()
    else:
        try:
            os.chdir(file)
            current_poll=file
            TkinViewResult()
        except FileNotFoundError:
            DisablePollSlectorWindowButtons()
            messagebox.showerror('Error','Poll does not exist ('+file+')')
            EnablePollSlectorWindowButtons()
def ExportPoll():
    global current_poll
    os.chdir(main_dir+'\\Polls')
    file=filename_entry.get()
    if len(file)==0 or file.isspace():
        DisablePollSlectorWindowButtons()
        messagebox.showerror('Error','Field can\'t be empty (Poll_Name)')
        EnablePollSlectorWindowButtons()
    else:
        DisablePollSlectorWindowButtons()
        try:
            os.chdir(file)
            current_poll=file
            ExportLis=[]
            if os.path.exists(current_poll+'_Question.txt'):
                with open(current_poll+'_Question.txt') as f:
                    Que=f.read()
                    ExportLis.append(Que)
            else:
                ExportLis.append(None)
            if os.path.exists(current_poll+'_Options.txt'):
                with open(current_poll+'_Options.txt') as f:
                    Opts=f.read()
                    ExportLis.append(Opts)
            else:
                ExportLis.append(None)
            if os.path.exists(current_poll+'_VotersDetails.csv'):
                VotersDetailsLis=[]
                with open(current_poll+'_VotersDetails.csv') as f:
                    r=csv.reader(f)
                    for i in r:
                        VotersDetailsLis.append(i)
                ExportLis.append(VotersDetailsLis)
            else:
                ExportLis.append(None)
            save_path=filedialog.asksaveasfilename(defaultextension=".yuhpoll",fil
etypes=[("Poll file", "*.yuhpoll")],title="Save As")
            if save_path:
                with open(save_path,'wb') as f:
                    pickle.dump(ExportLis,f)
                messagebox.showinfo('Export Successful','Successfully exported')
            else:
                messagebox.showerror('Error','Save path not selected')
        except FileNotFoundError:
            messagebox.showerror('Error','Poll does not exist ('+file+')')
        EnablePollSlectorWindowButtons()
def ImportPoll():
    global current_poll
    os.chdir(main_dir+'\\Polls')
    file=filename_entry.get()
    if len(file)==0 or file.isspace():
        DisablePollSlectorWindowButtons()
        messagebox.showerror('Error','Field can\'t be empty (Poll_Name)')
        EnablePollSlectorWindowButtons()
    else:
        DisablePollSlectorWindowButtons()
        try:
            current_poll=file
            open_path=filedialog.askopenfilename(title="Open
File",filetypes=[("Poll file", "*.yuhpoll")])
            if open_path:
                with open(open_path,'rb') as f:
                    ImportLis=pickle.load(f)
                if len(ImportLis)==3:
                    os.mkdir(file)
                    os.chdir(file)
                    if not ImportLis[0]==None:
                        with open(current_poll+'_Question.txt','w') as f:
                            f.write(ImportLis[0])
                    if not ImportLis[1]==None:
                        with open(current_poll+'_Options.txt','w') as f:
                            f.write(ImportLis[1])
                    if not ImportLis[2]==None:
                        with
open(current_poll+'_VotersDetails.csv','w',newline='') as f:
                            w=csv.writer(f)
                            w.writerows(ImportLis[2])
                Import_Time=DT.now().strftime('%d %B %Y')+' at
'+DT.now().strftime('%H:%M:%S')
                with open(current_poll+'_Info.txt','w') as f:
                    f.write(' (Imported)
'+users_dict[LogIn_user_ID][1]+'\n'+Import_Time)
                messagebox.showinfo('Import Successful','Successfully imported to
('+file+')')
                TkinPOfficerMenu()
            else:
                messagebox.showerror('Error','Import file not selected')
        except FileExistsError:
            messagebox.showerror('Error','Poll already exists ('+file+')')
        EnablePollSlectorWindowButtons()
def ViewPolls():
    global Polls
    Polls=os.listdir(main_dir+'\\Polls')
def DeletePoll():
    os.chdir(main_dir+'\\Polls')
    file=filename_entry.get()
    if len(file)==0 or file.isspace():
        DisablePollSlectorWindowButtons()
        messagebox.showerror('Error','Field can\'t be empty (Poll_Name)')
```

```
            EnablePollSlectorWindowButtons()
        else:
            try:
                os.remove(file)
                messagebox.showinfo('Deleted successfully','Poll deleted successfully
('+file+')')
            except FileNotFoundError:
                DisablePollSlectorWindowButtons()
                messagebox.showerror('Error','Poll does not exist ('+file+')')
                EnablePollSlectorWindowButtons()
            except:
                DisablePollSlectorWindowButtons()
                messagebox.showerror('Error','Error deleting poll ('+file+')')
                EnablePollSlectorWindowButtons()
def CurrentPollRefresh():
    global
CurrentQue,CurrentOpt_lis,PollInfo_textLis,NumOfOpts,NumOfVoters,LogText
    if os.path.exists(current_poll+'_Question.txt'):
        with open(current_poll+'_Question.txt') as q_f:
            CurrentQue=q_f.read()
    else:
        CurrentQue='**NOT ADDED**'
    if os.path.exists(current_poll+'_Options.txt'):
        with open(current_poll+'_Options.txt') as o_f:
            CurrentOpt_lis=[]
            for i in o_f.readlines():
                CurrentOpt_lis.append(i)
            for i in range(len(CurrentOpt_lis)):
                CurrentOpt_lis[i]=CurrentOpt_lis[i][:-1]
    else:
        CurrentOpt_lis=['**Options not added**']
    if os.path.exists(current_poll+'_VotersDetails.csv'):
        with open(current_poll+'_VotersDetails.csv') as v_f:
            Voterlis=[]
            r=csv.reader(v_f)
            for i in r:
                Voterlis.append(i[1])
        NumOfVoters=str(len(Voterlis))
    else:
        NumOfVoters='**Voters Not Added**'
    if os.path.exists(current_poll+'_Info.txt'):
        with open(current_poll+'_Info.txt') as f:
            Info=f.readlines()
    else:
        Info=('N.A.','N.A.')
    PollInfo_textLis=[]
    PollInfo_textLis.append('Poll created by '+Info[0]+'\n')
    PollInfo_textLis.append('Poll created on '+Info[1]+'\n')
```

```
        PollInfo_textLis.append('\n')
        PollInfo_textLis.append('Question : '+CurrentQue+'\n')
        PollInfo_textLis.append('\n')
        if CurrentOpt_lis==['**Options not added**']:
            NumOfOpts=0
        else:
            NumOfOpts=len(CurrentOpt_lis)
        PollInfo_textLis.append('Number of Options : '+str(NumOfOpts)+'\n')
        PollInfo_textLis.append('Options : ')
        for i in CurrentOpt_lis:
            PollInfo_textLis.append('\n\t'+i)
        PollInfo_textLis.append('\n\n')
        PollInfo_textLis.append('Number of Voters : '+NumOfVoters)
    def Question():
        que = Question_entry.get()
        if len(que)==0 or que.isspace():
            messagebox.showerror('Error','Field can\'t be empty(Question)')
        else:
            with open(current_poll+'_Question.txt','w') as f:
                f.write(que)
            messagebox.showinfo('Updated Successfully','Question updated
    successfully')
        TkinPOMenuRefresh()
    def AddOptions():
        options_lis=[]
        if not os.path.exists(current_poll+'_Options.txt'):
            open(current_poll+'_Options.txt','x').close()
        with open(current_poll+'_Options.txt') as f:
            for i in f.readlines():
                options_lis.append(i)
        option = Option_entry.get()
        if len(option)==0 or option.isspace():
            messagebox.showerror('Error','Field can\'t be empty(Option)')
        elif option+'\n' not in options_lis:
            options_lis.append(option+'\n')
            options_lis.sort()
            with open(current_poll+'_Options.txt','w') as f:
                for i in options_lis:
                    f.write(i)
            messagebox.showinfo('Added successfully','Option added
    successfully('+option+')')
        else:
            messagebox.showerror('Error','Option already entered('+option+')')
        TkinPOMenuRefresh()
    def RemoveOption():
        options_list=[]
        try:
            with open(current_poll+'_Options.txt') as f:
```

```
            for i in f.readlines():
                options_list.append(i)
        for i in range(len(options_list)):
            options_list[i]=options_list[i][:-1]
        rem=OptRem_var.get()
        if rem in options_list:
            options_list.remove(rem)
            if len(options_list)>0:
                with open(current_poll+'_Options.txt','w') as f:
                    for i in options_list:
                        f.write(i+'\n')
            else:
                os.remove(current_poll+'_Options.txt')
            messagebox.showinfo('Removed successfully','Option removed 
successfully('+rem+')')
        else:
            messagebox.showerror('Error','Error while removing')
    except FileNotFoundError:
        messagebox.showerror('Error','Options not added ('+current_poll+')')
    TkinPOMenuRefresh()
def AddVoters():
    voterid_lis=[]
    voters_dict={}
    if not os.path.exists(current_poll+'_VotersDetails.csv'):
        open(current_poll+'_VotersDetails.csv','x').close()
    f=open(current_poll+'_VotersDetails.csv')
    r=csv.reader(f)
    for i in r:
        voterid_lis.append(i[1])
        voters_dict[i[1]]=i[0]
    f.close()
    voter = Voter_entry.get()
    if len(voter)==0 or voter.isspace():
        messagebox.showerror('Error','Field can\'t be empty(Voter_Name)')
    else:
        with open(current_poll+'_VotersDetails.csv','a',newline='') as f:
            w=csv.writer(f)
            while True:
                voter_id=str(random.randrange(100000,1000000))
                if voter_id not in voterid_lis:
                    voterid_lis.append(voter_id)
                    break
            voters_dict[voter_id]=voter
            w.writerow([voter,voter_id])
        messagebox.showinfo('Added successfully','Voter added 
successfully\n'+voter+ ' your voter ID is : '+voter_id+'\nPLEASE REMEMBER\n')
    TkinPOMenuRefresh()
def RemoveVoter():
    voters_dict={}
    voted_dict={}
    if os.path.exists(current_poll+'_VotersDetails.csv'):
        with open(current_poll+'_VotersDetails.csv') as f:
            r=csv.reader(f)
            for i in r:
                voters_dict[i[1]]=i[0]
        if os.path.exists(current_poll+'_VotedData.csv'):
            with open(current_poll+'_VotedData.csv') as f:
                r=csv.reader(f)
                for i in r:
                    voted_dict[i[0]]=(i[1],i[2])
        rem=RemVoter_entry.get()
        if rem in voters_dict and rem not in voted_dict:
            RemName=voters_dict[rem]
            del voters_dict[rem]
            if len(voters_dict)>0:
                with open(current_poll+'_VotersDetails.csv','w',newline='') as f:
                    w=csv.writer(f)
                    for i in voters_dict:
                        w.writerow([voters_dict[i],i])
            else:
                os.remove(current_poll+'_VotersDetails.csv')
            messagebox.showinfo('Removed successfully','Voter removed successfully
'+RemName+'('+rem+')')
        elif rem in voters_dict and rem in voted_dict:
            messagebox.showerror('Error','\n'+voted_dict[rem][0]+' ('+rem+') has
already voted on '+voted_dict[rem][1])
        else:
            messagebox.showerror('Error','Invalid voter ID.')
    else:
        messagebox.showerror('Error','No voter added.')
    TkinPOMenuRefresh()
def ExportVoters():
    if os.path.exists(current_poll+'_VotersDetails.csv'):
        save_path=filedialog.asksaveasfilename(defaultextension=".csv",filetypes=[
("Comma-separated values File", "*.csv")],title="Save As")
        VotersDetailsLis=[]
        with open(current_poll+'_VotersDetails.csv') as f:
            r=csv.reader(f)
            for i in r:
                VotersDetailsLis.append(i)
        if save_path:
            with open(save_path,'a',newline='') as f:
                w=csv.writer(f)
                w.writerows(VotersDetailsLis)
            messagebox.showinfo('Exported successfully','Voters details exported
successfully')
```

```
            else:
                messagebox.showerror('Error','Save path not selected')
        else:
            messagebox.showerror('Error','No voter added.')
        TkinPOMenuRefresh()
    def ImportVoters():
        messagebox.showwarning('Warning','Importing voters will replace existing
voters!')
        open_path=filedialog.askopenfilename(title="Open File",filetypes=[("Comma-
separated values File", "*.csv")])
        if open_path:
            ImportFlag=True
            voteridlis=[]
            voterslis=[]
            with open(open_path) as f:
                r=csv.reader(f)
                for i in r:
                    if len(i)==2 and i[1].isdigit():
                        if i[1] not in voteridlis:
                            voterslis.append(i)
                            voteridlis.append(i[1])
                        else:
                            ImportFlag=False
                            messagebox.showerror('Error while importing voters','Voter
ID repeated')
                            break
                    else:
                        ImportFlag=False
                        messagebox.showerror('Error while importing voters','CSV file
not in correct format')
                        break
            if ImportFlag==True:
                with open(current_poll+'_VotersDetails.csv','w',newline='') as f:
                    w=csv.writer(f)
                    w.writerows(voterslis)
                messagebox.showinfo('Imported successfully','Voters details imported
successfully')
        else:
            messagebox.showerror('Error','Import file not selected')
        TkinPOMenuRefresh()
    def Polling(VID,VName):
        try:
            vote=Options_var.get()
            Vote_Time=DT.now().strftime('%d %B %Y')+' at
'+DT.now().strftime('%H:%M:%S')
            vote_dict[vote]=str(int(vote_dict[vote])+1)
            voted_dict[VID]=[VName,Vote_Time]
            with open(current_poll+'_PollData.csv','w',newline='') as f:
```

```
            w=csv.writer(f)
            for i in vote_dict:
                w.writerow([i,vote_dict[i]])
        with open(current_poll+'_VotedData.csv','a',newline='') as f:
            w=csv.writer(f)
            w.writerow([VID,VName,Vote_Time])
        messagebox.showinfo('Vote recorded',VName+' ('+VID+') vote recorded.')
    except:
        messagebox.showerror('Error','Vote was not recorded.\nPlease try again.')
    PollingWindow.destroy()
    TkinVoterLogin()
def SaveResult():
    try:
        open(current_poll+'_Result.txt','x').close()
        saved_list=[]
        vote_dict={}
        with open(current_poll+'_PollData.csv')as f:
            r=csv.reader(f)
            for i in r:
                vote_dict[i[0]]=i[1]
        f=open(current_poll+'_Result.txt','a')
        votes_list=[]
        for i in vote_dict:
            votes_list.append(int(vote_dict[i]))
        while len(votes_list)>0:
            max_vote=max(votes_list)
            for i in vote_dict:
                if int(vote_dict[i])==max_vote and i not in saved_list:
                    f.write(i+'-'+str(max_vote)+'\n')
                    saved_list.append(i)
            votes_list.remove(max_vote)
        f.close()
        PostPollingWindow.destroy()
        messagebox.showinfo('Saved successfully','Result saved successfully
('+current_poll+')')
        if current_designation=='Polling officer':
            POfficer_Window.deiconify()
        elif current_designation=='Polling agent':
            PAgent_Window.deiconify()
    except:
        messagebox.showerror('Error','Error while saving result')
def Tkin_closing():
    messagebox.showerror('Error','Use the Back / Logout / Exit button.')
def TkinWelcome():
    global WelcomeWindow
    WelcomeWindow = Tk()
    WelcomeWindow.config(bg=ColourCodeMain)
    WelcomeWindow.title("POLL\t-YUHANTH")
    Label(WelcomeWindow,text='Welcome to the Voting 
Poll!',font=MainTitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fonts
ize*7)
    WELCOMEButtons_frame = Frame(WelcomeWindow,bg=ColourCodeMain)
    WELCOMEButtons_frame.pack(pady=fontsize*4)
    LoginButton = 
Button(WELCOMEButtons_frame,text="Login",command=TkinLogIn,font=ButtonFont2,bg=Col
ourCodeButtonBG,fg=ColourCodeButtonFG)
    LoginButton.pack(side=TK.LEFT, padx=fontsize*3)
    SignUpButton = Button(WELCOMEButtons_frame,text="Sign 
up",command=TkinSignup,font=ButtonFont2,bg=ColourCodeButtonBG,fg=ColourCodeButtonF
G)
    SignUpButton.pack(side=TK.RIGHT, padx=fontsize*3)
    preferences_button = 
Button(WelcomeWindow,text="Preferences",command=TkinPreferences,font=ButtonFont3,b
g=ColourCodeButtonBG,fg=ColourCodeButtonFG)
    preferences_button.pack(padx=fontsize*4,side=TK.TOP,anchor=TK.NE)
    exit_button = 
Button(WelcomeWindow,text="Exit",command=sys.exit,font=ButtonFont3,bg=ColourCodeBu
ttonBG,fg=ColourCodeButtonFG)
    exit_button.pack(pady=fontsize*3)
    Label(WelcomeWindow,text='\t\t\t\t\t-
Yuhanth',font=TextFont2,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*1)
    Label(WelcomeWindow,text='\t\t\t\t\t XII -
JEE',font=TextFont2,bg=ColourCodeMain,fg=ColourCodeText).pack()
    WelcomeWindow.attributes('-fullscreen', True)
    WelcomeWindow.protocol("WM_DELETE_WINDOW",Tkin_closing)
    WelcomeWindow.mainloop()
def TkinLogIn():
    global UserID_entry,Password_entry,LoginWindow,login_button,LogIn_on_enter
    def Back():
        LoginWindow.destroy()
        WelcomeWindow.deiconify()
    def LogIn_on_enter(event):
        LogIn()
    WelcomeWindow.withdraw()
    LoginWindow=Toplevel(WelcomeWindow)
    LoginWindow.config(bg=ColourCodeMain)
    LoginWindow.title("POLL\t-YUHANTH")
    Label(LoginWindow,text='Welcome to the Voting 
Poll!',font=MainTitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fonts
ize*7)
    Label(LoginWindow,text="Login to 
continue",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*2
)
    back_button = 
Button(LoginWindow,text="Back",command=Back,font=ButtonFont3,bg=ColourCodeButtonBG
,fg=ColourCodeButtonFG)
```

```
    back_button.pack(side=TK.TOP,anchor=TK.NE,padx=fontsize*4)
    UserIDEntryFrame = Frame(LoginWindow,bg=ColourCodeMain)
    UserIDEntryFrame.pack(pady=fontsize*5)
    Label(UserIDEntryFrame,text="User ID
:",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(side=TK.LEFT,
padx=fontsize*3)
    UserID_entry = Entry(UserIDEntryFrame,font=TextFont3,width=fontsize*4)
    UserID_entry.pack(side=TK.RIGHT, padx=fontsize*3)
    PasswordEntryFrame = Frame(LoginWindow,bg=ColourCodeMain)
    PasswordEntryFrame.pack(pady=fontsize*4)
    Label(PasswordEntryFrame,text="Password
:",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(side=TK.LEFT,
padx=fontsize*3)
    Password_entry =
Entry(PasswordEntryFrame,font=TextFont3,show="*",width=fontsize*4)
    Password_entry.pack(side=TK.RIGHT, padx=fontsize*3)
    login_button =
Button(LoginWindow,text="Login",command=LogIn,font=ButtonFont2Bold,bg=ColourCodeBu
ttonBG,fg=ColourCodeButtonFG)
    login_button.pack(pady=fontsize*2)
    LoginWindow.bind('<Return>', LogIn_on_enter)
    LoginWindow.resizable(False, False)
    LoginWindow.attributes('-fullscreen', True)
    LoginWindow.protocol("WM_DELETE_WINDOW",Tkin_closing)
def TkinSignup():
    global
SignUpWindow,name_entry,userID_entry,designation_var,designation_dropdown,password
_entry,confirm_password_entry
    def Back():
        SignUpWindow.destroy()
        WelcomeWindow.deiconify()
    WelcomeWindow.withdraw()
    SignUpWindow = Toplevel(WelcomeWindow)
    SignUpWindow.config(bg=ColourCodeMain)
    SignUpWindow.title("Sign up")
    Label(SignUpWindow,text='Welcome to the Voting
Poll!',font=MainTitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fonts
ize*7)
    Label(SignUpWindow,text="Sign up to
continue",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3
)
    back_button =
Button(SignUpWindow,text="Back",command=Back,font=ButtonFont3,bg=ColourCodeButtonB
G,fg=ColourCodeButtonFG)
    back_button.pack(side=TK.TOP,anchor=TK.NE,padx=fontsize*4)
    NameEntryFrame = Frame(SignUpWindow,bg=ColourCodeMain)
    NameEntryFrame.pack(pady=fontsize*4)
    Label(NameEntryFrame,text="Name
:",font=TextFont2,bg=ColourCodeMain,fg=ColourCodeText).pack(side=TK.LEFT,
padx=fontsize*3)
    name_entry = Entry(NameEntryFrame,font=TextFont2)
    name_entry.pack(side=TK.RIGHT, padx=fontsize*3)
    UsedIDEntryFrame = Frame(SignUpWindow,bg=ColourCodeMain)
    UsedIDEntryFrame.pack(pady=fontsize*4)
    Label(UsedIDEntryFrame,text="User ID
:",font=TextFont2,bg=ColourCodeMain,fg=ColourCodeText).pack(side=TK.LEFT,
padx=fontsize*3)
    userID_entry = Entry(UsedIDEntryFrame,font=TextFont2)
    userID_entry.pack(side=TK.RIGHT, padx=fontsize*3)
    PassEntryFrame = Frame(SignUpWindow,bg=ColourCodeMain)
    PassEntryFrame.pack(pady=fontsize*4)
    Label(PassEntryFrame,text="Create Password
:",font=TextFont2,bg=ColourCodeMain,fg=ColourCodeText).pack(side=TK.LEFT,
padx=fontsize*3)
    password_entry = Entry(PassEntryFrame,show="*",font=TextFont2)
    password_entry.pack(side=TK.RIGHT, padx=fontsize*3)
    RePassEntryFrame = Frame(SignUpWindow,bg=ColourCodeMain)
    RePassEntryFrame.pack(pady=fontsize*4)
    Label(RePassEntryFrame,text="Confirm Password
:",font=TextFont2,bg=ColourCodeMain,fg=ColourCodeText).pack(side=TK.LEFT,
padx=fontsize*3)
    confirm_password_entry = Entry(RePassEntryFrame,show="*",font=TextFont2)
    confirm_password_entry.pack(side=TK.RIGHT, padx=fontsize*3)
    submit_button =
Button(SignUpWindow,text="Submit",font=ButtonFont2Bold,command=SignUp,bg=ColourCod
eButtonBG,fg=ColourCodeButtonFG)
    submit_button.pack(pady=fontsize*2)
    SignUpWindow.attributes('-fullscreen', True)
    SignUpWindow.protocol("WM_DELETE_WINDOW",Tkin_closing)
def TkinPreferences():
    global Preferences_Window,FontStyle_var,Theme_var
    def Back():
        Preferences_Window.destroy()
        WelcomeWindow.destroy()
        PreferenceWriter()
        TkinWelcome()
    WelcomeWindow.withdraw()
    Preferences_Window=Toplevel(WelcomeWindow)
    Preferences_Window.config(bg=ColourCodeMain)
    Preferences_Window.title("Preferences Window")
    Label(Preferences_Window,text="Preference
Menu",font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3
)
    Label(Preferences_Window,text="Make sure '] [' is inside the
screen",font=TextFont3,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3)
```

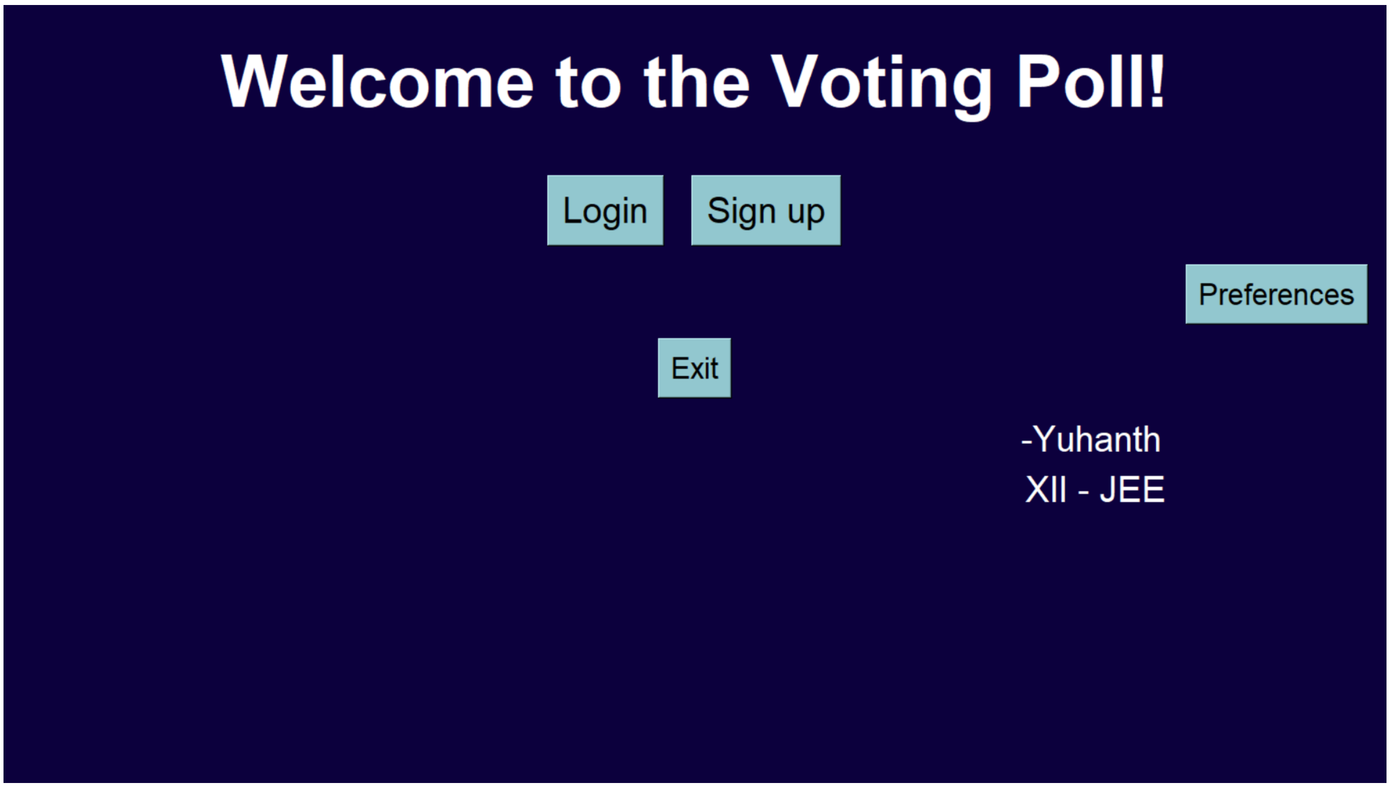
```
    Label(Preferences_Window,text="]\t\t\t\t[",font=TitleTextFont,bg=ColourCodeMai
n,fg=ColourCodeText).pack(pady=fontsize*3)
    back_button =
Button(Preferences_Window,text="Back",command=Back,font=ButtonFont3,bg=ColourCodeB
uttonBG,fg=ColourCodeButtonFG)
    back_button.pack(side=TK.TOP,anchor=TK.NE,padx=fontsize*4)
    FontSizeButtons_frame = Frame(Preferences_Window,bg=ColourCodeMain)
    FontSizeButtons_frame.pack(pady=fontsize*4)
    IncreaseFont_button = Button(FontSizeButtons_frame,text="Increase Font Size
(+)",command=IncreaseFontSize,font=ButtonFont2,bg=ColourCodeButtonBG,fg=ColourCode
ButtonFG)
    IncreaseFont_button.pack(side=TK.LEFT, padx=fontsize*3)
    DecreaseFont_button = Button(FontSizeButtons_frame,text="Decrease Font Size (-
)",command=DecreaseFontSize,font=ButtonFont2,bg=ColourCodeButtonBG,fg=ColourCodeBu
ttonFG)
    DecreaseFont_button.pack(side=TK.RIGHT, padx=fontsize*3)
    FontFamilySelection_frame = Frame(Preferences_Window,bg=ColourCodeMain)
    FontFamilySelection_frame.pack(pady=fontsize*4)
    Label(FontFamilySelection_frame,text="Font family
:",font=TextFont2,bg=ColourCodeMain,fg=ColourCodeText).pack(side=TK.LEFT,
padx=fontsize*3)
    FontStyle_var = TK.StringVar(Preferences_Window)
    FontStyle_var.set(fontstyle)
    FontStyle_var.trace('w',ChangeFontStyle)
    FontStyle_dropdown = OptionMenu(FontFamilySelection_frame, FontStyle_var,
*['Arial','Courier New','Georgia','Impact','Lucida Console'])
    FontStyle_dropdown.config(font=TextFont2,width=fontsize*4,bg=ColourCodeButtonB
G,fg=ColourCodeButtonFG)
    FontStyle_dropdown['menu'].config(font=TextFont3)
    FontStyle_dropdown['menu'].config(activebackground=ColourCodeButtonBG,
activeforeground=ColourCodeButtonFG)
    FontStyle_dropdown.pack(side=TK.RIGHT, padx=fontsize*3)
    ThemeSelection_frame = Frame(Preferences_Window,bg=ColourCodeMain)
    ThemeSelection_frame.pack(pady=fontsize*4)
    Label(ThemeSelection_frame,text="Theme
:",font=TextFont2,bg=ColourCodeMain,fg=ColourCodeText).pack(side=TK.LEFT,
padx=fontsize*3)
    Theme_var = TK.StringVar(Preferences_Window)
    Theme_var.set(theme)
    Theme_var.trace('w',ChangeFontStyle)
    Themes_dropdown = OptionMenu(ThemeSelection_frame, Theme_var,
*['Default','Light','Dark'])
    Themes_dropdown.config(font=TextFont2,width=fontsize*4,bg=ColourCodeButtonBG,f
g=ColourCodeButtonFG)
    Themes_dropdown['menu'].config(font=TextFont3)
    Themes_dropdown['menu'].config(activebackground=ColourCodeButtonBG,
activeforeground=ColourCodeButtonFG)
    Themes_dropdown.pack(side=TK.RIGHT, padx=fontsize*3)
    Preferences_Window.attributes('-fullscreen', True)
    Preferences_Window.protocol("WM_DELETE_WINDOW",Tkin_closing)
def TkinAdminMenu():
    global Admin_window
    def AdminLogOut():
        global LoggedIn_Flag
        Admin_window.destroy()
        LoggedIn_Flag=False
        LoginWindow.deiconify()
        login_button.config(state=TK.NORMAL)
        LoginWindow.bind('<Return>', LogIn_on_enter)
    Admin_window = Toplevel(LoginWindow)
    Admin_window.config(bg=ColourCodeMain)
    Admin_window.title("Admin Window")
    Label(Admin_window,text="Admin
Menu",font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3
)
    Label(Admin_window,text="Logged in as
Admin",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3)
    logout_button =
Button(Admin_window,text="Logout",command=AdminLogOut,font=ButtonFont3,bg=ColourCo
deButtonBG,fg=ColourCodeButtonFG)
    logout_button.pack(padx=fontsize*4,side=TK.TOP,anchor=TK.NE)
    Admin_windowButtons_frame1 = Frame(Admin_window,bg=ColourCodeMain)
    Admin_windowButtons_frame1.pack(pady=fontsize*4)
    ChangeUID_button = Button(Admin_windowButtons_frame1,text="Change User
ID",font=ButtonFont1,command=TkinChangeUID,bg=ColourCodeButtonBG,fg=ColourCodeButt
onFG)
    ChangeUID_button.pack(side=TK.LEFT, padx=fontsize*3)
    ChangePassword_button = Button(Admin_windowButtons_frame1,text="Change
Password",font=ButtonFont1,command=TkinChangePass,bg=ColourCodeButtonBG,fg=ColourC
odeButtonFG)
    ChangePassword_button.pack(side=TK.RIGHT, padx=fontsize*3)
    ApproveUser_button = Button(Admin_window,text="Approve or Deny
User",font=ButtonFont1,command=TkinApproveUser,bg=ColourCodeButtonBG,fg=ColourCode
ButtonFG)
    ApproveUser_button.pack(pady=fontsize*3)
    RemUser_button = Button(Admin_window,text="Remove
User",font=ButtonFont1,command=TkinRemoveUser,bg=ColourCodeButtonBG,fg=ColourCodeB
uttonFG)
    RemUser_button.pack(pady=fontsize*3)
    Viewusers_button = Button(Admin_window,text="View
Users",font=ButtonFont1,command=TkinViewUsers,bg=ColourCodeButtonBG,fg=ColourCodeB
uttonFG)
    Viewusers_button.pack(pady=fontsize*3)
    Admin_window.attributes('-fullscreen', True)
    Admin_window.protocol("WM_DELETE_WINDOW",Tkin_closing)
def TkinChangeUID():
    global ChangeUIDWindow,NewUID_entry,CurrentPassword_entry
    def Back():
        ChangeUIDWindow.destroy()
        Admin_window.deiconify()
    Admin_window.withdraw()
    ChangeUIDWindow = Toplevel(Admin_window)
    ChangeUIDWindow.config(bg=ColourCodeMain)
    ChangeUIDWindow.title("Change User ID (Admin)")
    Label(ChangeUIDWindow,text="Change User
Id",font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3)
    Label(ChangeUIDWindow,text="Logged in as
Admin",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3)
    back_button =
Button(ChangeUIDWindow,text="Back",command=Back,font=ButtonFont3,bg=ColourCodeButt
onBG,fg=ColourCodeButtonFG)
    back_button.pack(side=TK.TOP,anchor=TK.NE,padx=fontsize*4)
    NewUID = Frame(ChangeUIDWindow,bg=ColourCodeMain)
    NewUID.pack(pady=fontsize*4)
    Label(NewUID,text="New User ID
:",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(side=TK.LEFT,
padx=fontsize*3)
    NewUID_entry = Entry(NewUID,font=TextFont3,width=fontsize*4)
    NewUID_entry.pack(side=TK.RIGHT, padx=fontsize*3)
    CurrentPassEntryFrame = Frame(ChangeUIDWindow,bg=ColourCodeMain)
    CurrentPassEntryFrame.pack(pady=fontsize*4)
    Label(CurrentPassEntryFrame,text="Current Password
:",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(side=TK.LEFT,
padx=fontsize*3)
    CurrentPassword_entry =
Entry(CurrentPassEntryFrame,font=TextFont3,show="*",width=fontsize*4)
    CurrentPassword_entry.pack(side=TK.RIGHT, padx=fontsize*3)
    submit_button =
Button(ChangeUIDWindow,text="Submit",font=ButtonFont2,command=ChangeUIDAdmin,bg=Co
lourCodeButtonBG,fg=ColourCodeButtonFG)
    submit_button.pack(pady=fontsize*2)
    ChangeUIDWindow.attributes('-fullscreen', True)
    ChangeUIDWindow.protocol("WM_DELETE_WINDOW",Tkin_closing)
def TkinChangePass():
    global
ChangePassWindow,CurrentPassword_entry,NewPassword_entry,ReNewPassword_entry
    def Back():
        ChangePassWindow.destroy()
        Admin_window.deiconify()
    Admin_window.withdraw()
    ChangePassWindow = Toplevel(Admin_window)
    ChangePassWindow.config(bg=ColourCodeMain)
    ChangePassWindow.title("Change Password (Admin)")
    Label(ChangePassWindow,text="Change 
Password",font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsi
ze*3)
    Label(ChangePassWindow,text="Logged in as 
Admin",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3)
    back_button = 
Button(ChangePassWindow,text="Back",command=Back,font=ButtonFont3,bg=ColourCodeBut
tonBG,fg=ColourCodeButtonFG)
    back_button.pack(side=TK.TOP,anchor=TK.NE,padx=fontsize*4)
    CurrentPassEntryFrame = Frame(ChangePassWindow,bg=ColourCodeMain)
    CurrentPassEntryFrame.pack(pady=fontsize*4)
    Label(CurrentPassEntryFrame,text="Current Password 
:",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(side=TK.LEFT, 
padx=fontsize*3)
    CurrentPassword_entry = 
Entry(CurrentPassEntryFrame,font=TextFont3,show="*",width=fontsize*4)
    CurrentPassword_entry.pack(side=TK.RIGHT, padx=fontsize*3)
    NewPassEntryFrame = Frame(ChangePassWindow,bg=ColourCodeMain)
    NewPassEntryFrame.pack(pady=fontsize*4)
    Label(NewPassEntryFrame,text="New Password 
:",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(side=TK.LEFT, 
padx=fontsize*3)
    NewPassword_entry = 
Entry(NewPassEntryFrame,font=TextFont3,show="*",width=fontsize*4)
    NewPassword_entry.pack(side=TK.RIGHT, padx=fontsize*3)
    ReNewPassEntryFrame = Frame(ChangePassWindow,bg=ColourCodeMain)
    ReNewPassEntryFrame.pack(pady=fontsize*4)
    Label(ReNewPassEntryFrame,text="Re-enter New Password 
:",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(side=TK.LEFT, 
padx=fontsize*3)
    ReNewPassword_entry = 
Entry(ReNewPassEntryFrame,font=TextFont3,show="*",width=fontsize*4)
    ReNewPassword_entry.pack(side=TK.RIGHT, padx=fontsize*3)
    submit_button = 
Button(ChangePassWindow,text="Submit",font=ButtonFont2,command=ChangePassAdmin,bg=
ColourCodeButtonBG,fg=ColourCodeButtonFG)
    submit_button.pack(pady=fontsize*2)
    ChangePassWindow.attributes('-fullscreen', True)
    ChangePassWindow.protocol("WM_DELETE_WINDOW",Tkin_closing)
def TkinApproveUser():
    global ApproveUserWindow,Non_Approved_user_listbox,designation_var
    def Back():
        ApproveUserWindow.destroy()
        Admin_window.deiconify()
    Admin_window.withdraw()
    ApproveUserWindow = Toplevel(Admin_window)
    ApproveUserWindow.config(bg=ColourCodeMain)
    ApproveUserWindow.title("Approve User")
    Label(ApproveUserWindow,text="Approve
User",font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3
)
    back_button =
Button(ApproveUserWindow,text="Back",command=Back,font=ButtonFont3,bg=ColourCodeBu
ttonBG,fg=ColourCodeButtonFG)
    back_button.pack(side=TK.TOP,anchor=TK.NE,padx=fontsize*4)
    Non_Approved_user_listbox = Listbox(ApproveUserWindow,
selectmode=TK.SINGLE,font=TextFont2,height=fontsize,width=fontsize*7,bg=ColourCode
Main,fg=ColourCodeText)
    for i in NonApprovedUsers_Dict:
        Non_Approved_user_listbox.insert(TK.END, i)
    Non_Approved_user_listbox.pack(pady=10)
    DesignationEntryFrame = Frame(ApproveUserWindow,bg=ColourCodeMain)
    DesignationEntryFrame.pack(pady=fontsize*4)
    Label(DesignationEntryFrame,text="Designation
:",font=TextFont2,bg=ColourCodeMain,fg=ColourCodeText).pack(side=TK.LEFT,
padx=fontsize*3)
    designations=('Polling officer','Polling agent')
    designation_var = TK.StringVar(ApproveUserWindow)
    designation_var.set('Select designation')
    designation_dropdown = OptionMenu(DesignationEntryFrame, designation_var,
*designations)
    designation_dropdown.config(font=TextFont2,width=int(fontsize*3.6))
    designation_dropdown['menu'].config(font=TextFont2)
    designation_dropdown['menu'].config(activebackground=ColourCodeButtonBG,
activeforeground=ColourCodeButtonFG)
    designation_dropdown.pack(pady=fontsize*3)
    Buttons_frame = Frame(ApproveUserWindow,bg=ColourCodeMain)
    Buttons_frame.pack(pady=fontsize*4)
    Approve_button =
Button(Buttons_frame,text="Approve",command=ApproveUser,font=ButtonFont2,bg=Colour
CodeButtonBG,fg=ColourCodeButtonFG)
    Approve_button.pack(side=TK.LEFT, padx=fontsize*3)
    Deny_button =
Button(Buttons_frame,text="Deny",command=DenyUser,font=ButtonFont2,bg=ColourCodeBu
ttonBG,fg=ColourCodeButtonFG)
    Deny_button.pack(side=TK.RIGHT, padx=fontsize*3)
    ApproveUserWindow.attributes('-fullscreen', True)
    ApproveUserWindow.protocol("WM_DELETE_WINDOW",Tkin_closing)
def TkinRemoveUser():
    global RemUserWindow,Rem_UID_entry
    UserDictRefresh()
    def Back():
        RemUserWindow.destroy()
        Admin_window.deiconify()
    Admin_window.withdraw()
    RemUserWindow = Toplevel(Admin_window)
    RemUserWindow.config(bg=ColourCodeMain)
    RemUserWindow.title("Add User")
    Label(RemUserWindow,text="Remove
User",font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3
)
    back_button =
Button(RemUserWindow,text="Back",command=Back,font=ButtonFont3,bg=ColourCodeButton
BG,fg=ColourCodeButtonFG)
    back_button.pack(side=TK.TOP,anchor=TK.NE,padx=fontsize*4)
    Label(RemUserWindow,text="Enter User ID to remove
user",font=TextFont3,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3)
    Label(RemUserWindow,text="User
ID",font=TextFont2,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3)
    Rem_UID_entry = Entry(RemUserWindow,font=TextFont2)
    Rem_UID_entry.pack(pady=fontsize)
    Remove_button =
Button(RemUserWindow,text="Submit",font=ButtonFont2,command=RemoveUser,bg=ColourCo
deButtonBG,fg=ColourCodeButtonFG)
    Remove_button.pack(pady=fontsize*2)
    RemUserWindow.attributes('-fullscreen', True)
    RemUserWindow.protocol("WM_DELETE_WINDOW",Tkin_closing)
def TkinViewUsers():
    def Back():
        ViewUsersWindow.destroy()
        Admin_window.deiconify()
    Admin_window.withdraw()
    UserDictRefresh()
    ViewUsersWindow = Toplevel(Admin_window)
    ViewUsersWindow.config(bg=ColourCodeMain)
    ViewUsersWindow.title("View Users")
    Label(ViewUsersWindow,text="Users:",font=TitleTextFont,bg=ColourCodeMain,fg=Co
lourCodeText).pack(pady=fontsize*3)
    back_button =
Button(ViewUsersWindow,text="Back",command=Back,font=ButtonFont3,bg=ColourCodeButt
onBG,fg=ColourCodeButtonFG)
    back_button.pack(side=TK.TOP,anchor=TK.NE,padx=fontsize*4)
    users_text = TK.Text(ViewUsersWindow,font=TextFont2)
    users_text.pack(pady=fontsize*3)
    for i in users_dict:
        name,user_id,designation = users_dict[i][1],i,users_dict[i][2]
        users_text.insert(TK.END,f"Name: {name}\nUser ID: {user_id}\nDesignation:
{designation}\n\n")
    users_text.config(state=TK.DISABLED)
    ViewUsersWindow.attributes('-fullscreen', True)
    ViewUsersWindow.protocol("WM_DELETE_WINDOW",Tkin_closing)
def TkinPollSelector():
    global
LoginWindow,login_button,PollSelectorWindow,filename_entry,CreatePoll_button,Conti
```

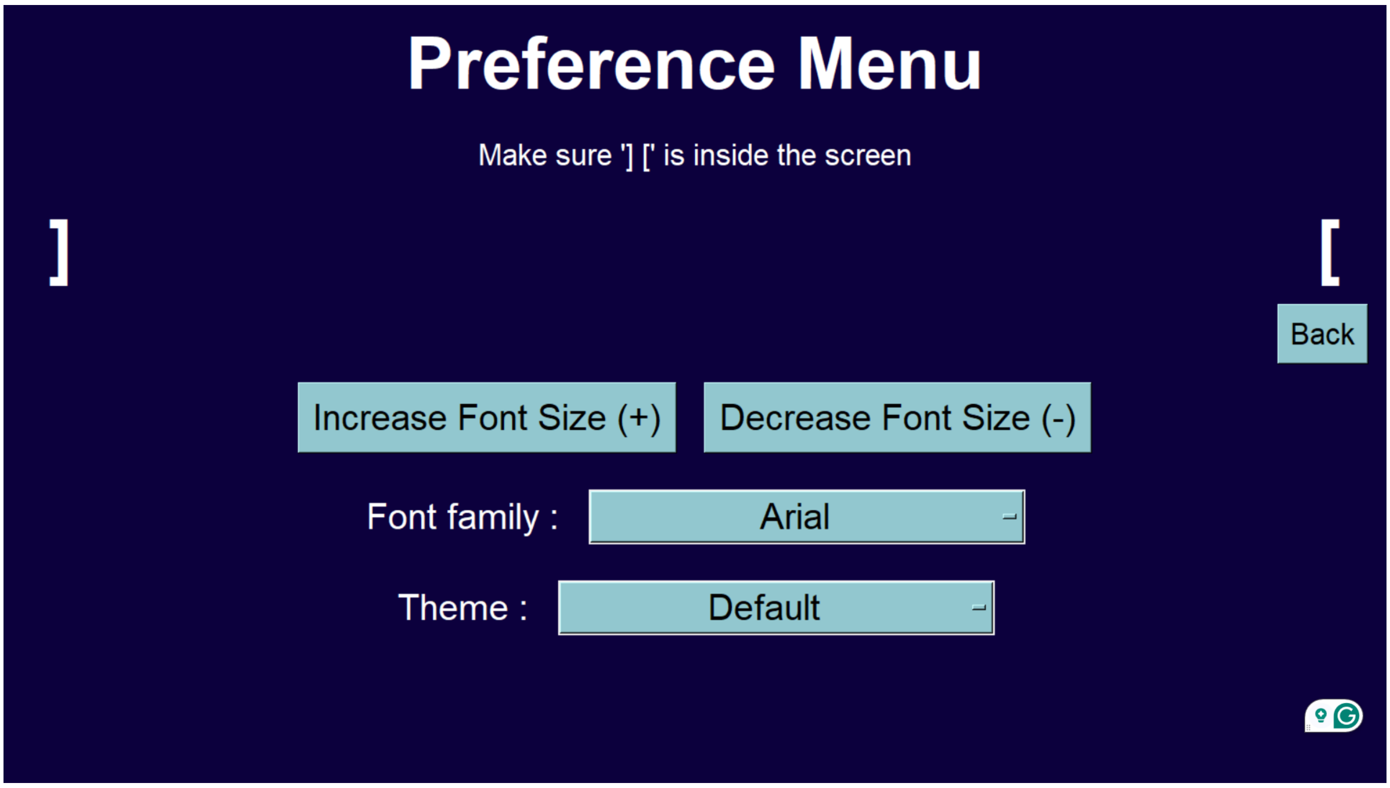
```
nuePoll_button,ViewResult_button,Logout_button,ImportPoll_button,ExportPoll_button
,ViewPolls_button,DeletePoll_button
    def POfficerLogOut():
        global LoggedIn_Flag
        LoggedIn_Flag=False
        PollSelectorWindow.destroy()
        LoginWindow.deiconify()
        login_button.config(state=TK.NORMAL)
        LoginWindow.bind('<Return>', LogIn_on_enter)
    PollSelectorWindow = Toplevel(LoginWindow)
    PollSelectorWindow.config(bg=ColourCodeMain)
    PollSelectorWindow.title("Poll Officer Window")
    Label(PollSelectorWindow,text="Polling officer's
Menu",font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3
)
    Label(PollSelectorWindow,text="Logged in as
"+users_dict[LogIn_user_ID][1],font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText)
.pack(pady=fontsize*3)
    Logout_button =
Button(PollSelectorWindow,text="Logout",command=POfficerLogOut,font=ButtonFont3,bg
=ColourCodeButtonBG,fg=ColourCodeButtonFG)
    Logout_button.pack(padx=fontsize*4,side=TK.TOP,anchor=TK.NE)
    Label(PollSelectorWindow,text="Enter name of the
poll",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3)
    filename_entry = Entry(PollSelectorWindow,font=TextFont2)
    filename_entry.pack(pady=fontsize*3)
    PSMenuButtons_frame1 = Frame(PollSelectorWindow,bg=ColourCodeMain)
    PSMenuButtons_frame1.pack(pady=fontsize*4)
    CreatePoll_button = Button(PSMenuButtons_frame1,text="Create
Poll",font=ButtonFont2,command=CreatePoll,bg=ColourCodeButtonBG,fg=ColourCodeButto
nFG)
    CreatePoll_button.pack(side=TK.LEFT, padx=fontsize*3)
    ContinuePoll_button = Button(PSMenuButtons_frame1,text="Continue
Poll",font=ButtonFont2,command=ContinuePoll,bg=ColourCodeButtonBG,fg=ColourCodeBut
tonFG)
    ContinuePoll_button.pack(side=TK.LEFT, padx=fontsize*3)
    ViewResult_button = Button(PSMenuButtons_frame1,text="View
Result",font=ButtonFont2,command=ViewResult,bg=ColourCodeButtonBG,fg=ColourCodeBut
tonFG)
    ViewResult_button.pack(side=TK.RIGHT, padx=fontsize*3)
    PSMenuButtons_frame2 = Frame(PollSelectorWindow,bg=ColourCodeMain)
    PSMenuButtons_frame2.pack(pady=fontsize*4)
    ImportPoll_button = Button(PSMenuButtons_frame2,text="Import
Poll",font=ButtonFont2,command=ImportPoll,bg=ColourCodeButtonBG,fg=ColourCodeButto
nFG)
    ImportPoll_button.pack(side=TK.LEFT, padx=fontsize*3)
    ExportPoll_button = Button(PSMenuButtons_frame2,text="Export
Poll",font=ButtonFont2,command=ExportPoll,bg=ColourCodeButtonBG,fg=ColourCodeButto
nFG)
    ExportPoll_button.pack(side=TK.RIGHT, padx=fontsize*3)
    PSMenuButtons_frame3 = Frame(PollSelectorWindow,bg=ColourCodeMain)
    PSMenuButtons_frame3.pack(pady=fontsize*4)
    ViewPolls_button = Button(PSMenuButtons_frame3,text="View
Polls",font=ButtonFont2,command=TkinViewPolls,bg=ColourCodeButtonBG,fg=ColourCodeB
uttonFG)
    ViewPolls_button.pack(side=TK.LEFT, padx=fontsize*3)
    DeletePoll_button = Button(PSMenuButtons_frame3,text="Delete
Poll",font=ButtonFont2,command=DeletePoll,bg=ColourCodeButtonBG,fg=ColourCodeButto
nFG)
    DeletePoll_button.pack(side=TK.RIGHT, padx=fontsize*3)
    PollSelectorWindow.attributes('-fullscreen', True)
    PollSelectorWindow.protocol("WM_DELETE_WINDOW",Tkin_closing)
def TkinViewPolls():
    def Back():
        ViewPollsWindow.destroy()
        PollSelectorWindow.deiconify()
    ViewPolls()
    PollSelectorWindow.withdraw()
    ViewPollsWindow = Toplevel(PollSelectorWindow)
    ViewPollsWindow.config(bg=ColourCodeMain)
    ViewPollsWindow.title("Polls")
    Label(ViewPollsWindow,text="Polls",font=TitleTextFont,bg=ColourCodeMain,fg=Col
ourCodeText).pack(pady=fontsize)
    Label(ViewPollsWindow,text="Number of Polls :
"+str(len(Polls)),font=TextFont3,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fo
ntsize)
    back_button =
Button(ViewPollsWindow,text="Back",command=Back,font=ButtonFont3,bg=ColourCodeButt
onBG,fg=ColourCodeButtonFG)
    back_button.pack(pady=fontsize,side=TK.TOP,anchor=TK.NE)
    result_text = TK.Text(ViewPollsWindow,font=TextFont1,height=fontsize*1.75)
    result_text.pack(pady=fontsize*3.25)
    for i in Polls:
        result_text.insert(TK.END,i+'\n')
    result_text.config(state=TK.DISABLED)
    ViewPollsWindow.attributes('-fullscreen', True)
    ViewPollsWindow.resizable(False, False)
    ViewPollsWindow.protocol("WM_DELETE_WINDOW",Tkin_closing)
def TkinPOMenuRefresh():
    CurrentPollRefresh()
    QueLabel.config(text='Question : '+CurrentQue)
    NoOfOptLabel1.config(text='Number of options added : '+str(NumOfOpts))
    NoOfOptLabel2.config(text='Number of options added : '+str(NumOfOpts))
    OptRem_menu=OptRem_dropdown['menu']
    OptRem_menu.delete(0, 'end')
    for i in CurrentOpt_lis:
        OptRem_menu.add_command(label=i,command=TK._setit(OptRem_var, i))
    OptRem_var.set('Select option')
    VotersFlagLabel1.config(text='Number of Voters added : '+NumOfVoters)
    VotersFlagLabel2.config(text='Number of Voters added : '+NumOfVoters)
    PollInfo_text.config(state=TK.NORMAL)
    PollInfo_text.delete("1.0", TK.END)
    for i in PollInfo_textLis:
        PollInfo_text.insert(TK.END,i)
    PollInfo_text.config(state=TK.DISABLED)
def TkinPOfficerMenu():
    def back():
        os.chdir(main_dir+'\\Polls')
        POfficer_Window.destroy()
        PollSelectorWindow.deiconify()
    PollSelectorWindow.withdraw()
    CurrentPollRefresh()
    global 
POfficer_Window,POMenuBack_Button,QueLabel,NoOfOptLabel1,NoOfOptLabel2,VotersFlagL
abel1,VotersFlagLabel2,Question_entry,Option_entry,OptRem_var,OptRem_dropdown,Vote
r_entry,RemVoter_entry,PollInfo_text,Polling_button
    POfficer_Window=Toplevel(PollSelectorWindow)
    POfficer_Window.config(bg=ColourCodeMain)
    POfficer_Window.title('Poll Menu')
    Label(POfficer_Window,text='Poll 
Menu',font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize)
    Label(POfficer_Window,text='Current Poll : 
'+current_poll,font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack()
    POMenuBack_Button = 
Button(POfficer_Window,text="Back",font=ButtonFont3,command=back,bg=ColourCodeButt
onBG,fg=ColourCodeButtonFG)
    POMenuBack_Button.pack(padx=fontsize*4,side=TK.TOP,anchor=TK.NE)
    POfficer_Notebook = ttk.Notebook(POfficer_Window,height=fontsize*85)
    style=ttk.Style()
    style.configure('TNotebook', background=ColourCodeMain)
    style.configure('TNotebook.Tab', background='black', 
padding=[fontsize*4,fontsize*4],font=TextFont3)
    style.configure('TFrame', background=ColourCodeMain)
    POfficer_Notebook.pack(expand=True,fill='both')
    QueTab = ttk.Frame(POfficer_Notebook,style='TFrame')
    QueLabel=Label(QueTab,text='Question : **NOT UPDATED 
(ERROR)**',font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText)
    QueLabel.pack(pady=fontsize*2)
    Label(QueTab,text="Enter question : 
",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*2)
    Question_entry = Entry(QueTab,width=fontsize*7,font=TextFont2)
    Question_entry.pack(pady=fontsize)
    UpdateQue_button = 
Button(QueTab,text="Update",font=ButtonFont2,command=Question,bg=ColourCodeButtonB
G,fg=ColourCodeButtonFG)
    UpdateQue_button.pack(pady=fontsize*2)
    AddOptsTab = ttk.Frame(POfficer_Notebook,style='TFrame')
    NoOfOptLabel1=Label(AddOptsTab,text="No of options added : **NOT UPDATED 
(ERROR)**",font=TextFont3,bg=ColourCodeMain,fg=ColourCodeText)
    NoOfOptLabel1.pack(pady=fontsize*2)
    Label(AddOptsTab,text="Add 
options",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*2)
    Label(AddOptsTab,text="Enter option : 
",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*2)
    Option_entry = Entry(AddOptsTab,width=fontsize*4,font=TextFont2)
    Option_entry.pack(pady=fontsize*2)
    AddOpt_button = 
Button(AddOptsTab,text="Submit",font=ButtonFont2,command=AddOptions,bg=ColourCodeB
uttonBG,fg=ColourCodeButtonFG)
    AddOpt_button.pack(pady=fontsize*2)
    RemOptTab = ttk.Frame(POfficer_Notebook,style='TFrame')
    NoOfOptLabel2=Label(RemOptTab,text="No of options added : **NOT UPDATED 
(ERROR)**",font=TextFont3,bg=ColourCodeMain,fg=ColourCodeText)
    NoOfOptLabel2.pack(pady=fontsize*2)
    Label(RemOptTab,text="Remove 
option",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*2)
    Label(RemOptTab,text="Select option : 
",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*2)
    OptRem_var = TK.StringVar(RemOptTab)
    OptRem_var.set('Select option')
    OptRem_dropdown = OptionMenu(RemOptTab, OptRem_var, *CurrentOpt_lis)
    OptRem_dropdown.config(font=TextFont2,width=fontsize*4)
    OptRem_dropdown['menu'].config(font=TextFont2)
    OptRem_dropdown['menu'].config(activebackground=ColourCodeButtonBG, 
activeforeground=ColourCodeButtonFG)
    OptRem_dropdown.pack(pady=fontsize*2)
    RemoveOpt_button = 
Button(RemOptTab,text="Remove",font=ButtonFont2,command=RemoveOption,bg=ColourCode
ButtonBG,fg=ColourCodeButtonFG)
    RemoveOpt_button.pack(pady=fontsize*2)
    AddVotersTab = ttk.Frame(POfficer_Notebook,style='TFrame')
    VotersFlagLabel1=Label(AddVotersTab,text="Voters : **NOT UPDATED 
(ERROR)**",font=TextFont3,bg=ColourCodeMain,fg=ColourCodeText)
    VotersFlagLabel1.pack(pady=fontsize*2)
    Label(AddVotersTab,text="Add 
Voters",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*2)
    Label(AddVotersTab,text="Enter name of the voter : 
",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*2)
    Voter_entry = Entry(AddVotersTab,width=fontsize*4,font=TextFont2)
    Voter_entry.pack(pady=fontsize*2)
    AddVotersTabButtons_frame = Frame(AddVotersTab,bg=ColourCodeMain)
    AddVotersTabButtons_frame.pack(pady=fontsize*4)
    ImportVoters_button = Button(AddVotersTabButtons_frame,text="Import 
Voters",font=ButtonFont2,command=ImportVoters,bg=ColourCodeButtonBG,fg=ColourCodeB
uttonFG)
    ImportVoters_button.pack(side=TK.LEFT,padx=fontsize*27)
    AddVoter_button = 
Button(AddVotersTabButtons_frame,text="Submit",font=ButtonFont2,command=AddVoters,
bg=ColourCodeButtonBG,fg=ColourCodeButtonFG)
    AddVoter_button.pack(side=TK.LEFT,padx=fontsize*27)
    ExportVoters_button = Button(AddVotersTabButtons_frame,text="Export 
Voters",font=ButtonFont2,command=ExportVoters,bg=ColourCodeButtonBG,fg=ColourCodeB
uttonFG)
    ExportVoters_button.pack(side=TK.RIGHT,padx=fontsize*27)
    RemVoterTab = ttk.Frame(POfficer_Notebook,style='TFrame')
    VotersFlagLabel2=Label(RemVoterTab,text="Voters : **NOT UPDATED 
(ERROR)**",font=TextFont3,bg=ColourCodeMain,fg=ColourCodeText)
    VotersFlagLabel2.pack(pady=fontsize*2)
    Label(RemVoterTab,text="Remove 
Voter",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*2)
    Label(RemVoterTab,text="Voter ID : 
",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*2)
    RemVoter_entry = Entry(RemVoterTab,width=int(fontsize*1.5),font=TextFont2)
    RemVoter_entry.pack(pady=fontsize*2)
    RemoveVoter_button = 
Button(RemVoterTab,text="Remove",font=ButtonFont2,command=RemoveVoter,bg=ColourCod
eButtonBG,fg=ColourCodeButtonFG)
    RemoveVoter_button.pack(pady=fontsize*2)
    PollInfoTab = ttk.Frame(POfficer_Notebook,style='TFrame')
    Label(PollInfoTab,text="Poll 
Info",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack()
    PollInfo_text = TK.Text(PollInfoTab,font=TextFont2,height=fontsize*5)
    PollInfo_text.pack()
    PollInfo_text.insert(TK.END,'**NOT UPDATED (ERROR)**')
    PollInfo_text.config(state=TK.DISABLED)

    POfficer_Notebook.add(QueTab,text='Question')
    POfficer_Notebook.add(AddOptsTab,text='Add Options')
    POfficer_Notebook.add(RemOptTab,text='Remove Option')
    POfficer_Notebook.add(AddVotersTab,text='Add Voters')
    POfficer_Notebook.add(RemVoterTab,text='Remove Voter')
    POfficer_Notebook.add(PollInfoTab,text='Poll Info')

    Polling_button = Button(POfficer_Window,text="Start or Continue 
polling",font=ButtonFont1,command=TkinVoterLogin,bg=ColourCodeButtonBG,fg=ColourCo
deButtonFG)
    Polling_button.pack(padx=fontsize*2,pady=fontsize*2)
    POfficer_Window.attributes('-fullscreen', True)
    POfficer_Window.protocol("WM_DELETE_WINDOW",Tkin_closing)
    TkinPOMenuRefresh()
def TkinViewResult():
    global ResultWindow,resultDict
    try:
        with open(current_poll+'_Result.txt')as f:
            result=f.readlines()
        resultDict={}
        for i in result:
            c=0
            for j in i[::-1]:
                if j=='-':
                    c+=1
                    break
                c+=1
            try:
                resultDict[i[:-c]]=int(i[1-c:-1])
            except:
                pass
        def Back():
            ResultWindow.destroy()
            os.chdir(main_dir+'\\Polls\\'+current_poll)
            PollSelectorWindow.deiconify()
        PollSelectorWindow.withdraw()
        UserDictRefresh()
        ResultWindow = Toplevel(PollSelectorWindow)
        ResultWindow.config(bg=ColourCodeMain)
        ResultWindow.title("Result - "+current_poll)
        Label(ResultWindow,text="Result - 
"+current_poll,font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack()
        back_button = 
Button(ResultWindow,text="Back",command=Back,font=ButtonFont3,bg=ColourCodeButtonB
G,fg=ColourCodeButtonFG)
        back_button.pack(side=TK.TOP,anchor=TK.NE)
        result_text = TK.Text(ResultWindow,font=TextFont1,height=fontsize*1.75)
        result_text.pack(pady=fontsize*3)
        for i in result:
            result_text.insert(TK.END,i)
        result_text.config(state=TK.DISABLED)
        ViewGraphButton = Button(ResultWindow,text="Show 
Graph",font=ButtonFont1,command=TkinResultGraph,bg=ColourCodeButtonBG,fg=ColourCod
eButtonFG)
        ViewGraphButton.pack(pady=fontsize*3)
        ResultWindow.attributes('-fullscreen', True)
        ResultWindow.resizable(False, False)
        ResultWindow.protocol("WM_DELETE_WINDOW",Tkin_closing)
    except FileNotFoundError:
        DisablePollSlectorWindowButtons()
        messagebox.showerror('Error','Result not saved ('+current_poll+').')
        EnablePollSlectorWindowButtons()
def TkinResultGraph():
    def Back():
        GraphWindow.destroy()
        ResultWindow.deiconify()
    fig = Figure(figsize=(fontsize*3, fontsize*1.25), dpi=100)
    ax = fig.add_subplot(111)
    ResultWindow.withdraw()
    GraphWindow=Toplevel(ResultWindow)
    GraphWindow.config(bg=ColourCodeMain)
    GraphWindow.title("Result Graph - "+current_poll)
    Label(GraphWindow,text="Result Graph - 
"+current_poll,font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack()
    back_button = 
Button(GraphWindow,text="Back",command=Back,font=ButtonFont3,bg=ColourCodeButtonBG
,fg=ColourCodeButtonFG)
    back_button.pack(side=TK.TOP,anchor=TK.NE)
    Graphcanvas = FigureCanvasTkAgg(fig, master=GraphWindow)
    Graphcanvas_widget = Graphcanvas.get_tk_widget()
    Graphcanvas_widget.pack(pady=fontsize*3)
    GraphWindow.attributes('-fullscreen', True)
    GraphWindow.resizable(False, False)
    GraphWindow.protocol("WM_DELETE_WINDOW",Tkin_closing)
    ax.clear()
    categories = list(resultDict.keys())
    values = list(resultDict.values())
    ax.bar(categories, values, color=ColourCodeMain)
    ax.set_title("Result Graph - "+current_poll)
    ax.set_xlabel('Options')
    ax.set_ylabel('Number of Votes')
    ax.yaxis.set_major_locator(MaxNLocator(integer=True))
    Graphcanvas.draw()
def TkinVoterLogin():
    global VoterLoginWindow,Voterid_entry
    Q_flag=O_flag=V_flag=False
    if current_designation=='Polling officer':
        Polling_button.config(state=TK.DISABLED)
        POMenuBack_Button.config(state=TK.DISABLED)
    elif current_designation=='Polling agent':
        ContinuePoll_button.config(state=TK.DISABLED)
    try:
        open(current_poll+'_Question.txt').close()
        Q_flag=True
    except FileNotFoundError:
        messagebox.showerror('Error','Qusetion not added ('+current_poll+')')
    try:
        open(current_poll+'_Options.txt').close()
        O_flag=True
    except FileNotFoundError:
        messagebox.showerror('Error','Options not added ('+current_poll+')')
    try:
        open(current_poll+'_VotersDetails.csv').close()
        V_flag=True
    except FileNotFoundError:
        messagebox.showerror('Error','Voters not added ('+current_poll+')')
    if current_designation=='Polling officer':
        Polling_button.config(state=TK.NORMAL)
        POMenuBack_Button.config(state=TK.NORMAL)
    elif current_designation=='Polling agent':
        ContinuePoll_button.config(state=TK.NORMAL)
    if Q_flag==True and O_flag==True and V_flag==True:
        if current_designation=='Polling officer':
            POfficer_Window.withdraw()
            VoterLoginWindow = Toplevel(POfficer_Window)
        elif current_designation=='Polling agent':
            VoterLoginWindow = Toplevel(PAgent_Window)
        VoterLoginWindow.title("Voter Login")
        VoterLoginWindow.config(bg=ColourCodeMain)
        Label(VoterLoginWindow,text="Voter 
Login",font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*
3)
        Label(VoterLoginWindow,text="Poll : 
"+current_poll,font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=f
ontsize*3)
        Label(VoterLoginWindow,text="Enter your Voter ID : 
",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3)
        Voterid_entry = Entry(VoterLoginWindow,width=fontsize*2,font=TextFont2)
        Voterid_entry.pack(pady=fontsize*3)
        Submit_button = 
Button(VoterLoginWindow,text="Submit",font=ButtonFont1,command=TkinPolling,bg=Colo
urCodeButtonBG,fg=ColourCodeButtonFG)
        Submit_button.pack(pady=fontsize*3)
        back_button = Button(VoterLoginWindow,text="Stop 
polling",command=TkinPostPolling,font=ButtonFont2,bg=ColourCodeButtonBG,fg=ColourC
odeButtonFG)
        back_button.pack(padx=fontsize*2,pady=fontsize*4,side=TK.TOP,anchor=TK.NE)
        VoterLoginWindow.attributes('-fullscreen', True)
        VoterLoginWindow.resizable(False,False)
        VoterLoginWindow.protocol("WM_DELETE_WINDOW",Tkin_closing)
    else:
        if current_designation=='Polling agent':
            PAgent_Window.deiconify()
def TkinPolling():
    global vote_dict,voted_dict,Options_var,PollingWindow
    with open(current_poll+'_Question.txt') as q_f:
        question=q_f.read()
    with open(current_poll+'_Options.txt') as o_f:
        opt_lis=[]
        for i in o_f.readlines():
            opt_lis.append(i)
        for i in range(len(opt_lis)):
            opt_lis[i]=opt_lis[i][:-1]
    with open(current_poll+'_VotersDetails.csv') as v_f:
        voters_dict={}
        voterid_lis=[]
        r=csv.reader(v_f)
        for i in r:
            voters_dict[i[1]]=i[0]
            voterid_lis.append(i[1])
    if not os.path.exists(current_poll+'_PollData.csv'):
        open(current_poll+'_PollData.csv','x').close()
        with open(current_poll+'_PollData.csv','w',newline='') as PD_f:
            for i in opt_lis:
                w=csv.writer(PD_f)
                w.writerow([i,0])
    if not os.path.exists(current_poll+'_VotedData.csv'):
        open(current_poll+'_VotedData.csv','x').close()
    vote_dict={}
    with open(current_poll+'_PollData.csv') as f:
        r=csv.reader(f)
        for i in r:
            vote_dict[i[0]]=i[1]
    voted_dict={}
    with open(current_poll+'_VotedData.csv') as f:
        r=csv.reader(f)
        for i in r:
            voted_dict[i[0]]=(i[1],i[2])
    id_ver=Voterid_entry.get()
    def PollingCaller():
        Polling(id_ver,voters_dict[id_ver])
    if id_ver in voterid_lis and id_ver not in voted_dict:
        VoterLoginWindow.destroy()
        if current_designation=='Polling officer':
            PollingWindow = Toplevel(POfficer_Window)
        elif current_designation=='Polling agent':
            PollingWindow = Toplevel(PAgent_Window)
        PollingWindow.config(bg=ColourCodeMain)
        PollingWindow.title("Polling")
        Label(PollingWindow,text=voters_dict[id_ver],font=TitleTextFont,bg=ColourC
odeMain,fg=ColourCodeText).pack(pady=fontsize*3)
        Label(PollingWindow,text='Please
vote',font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3
)
```

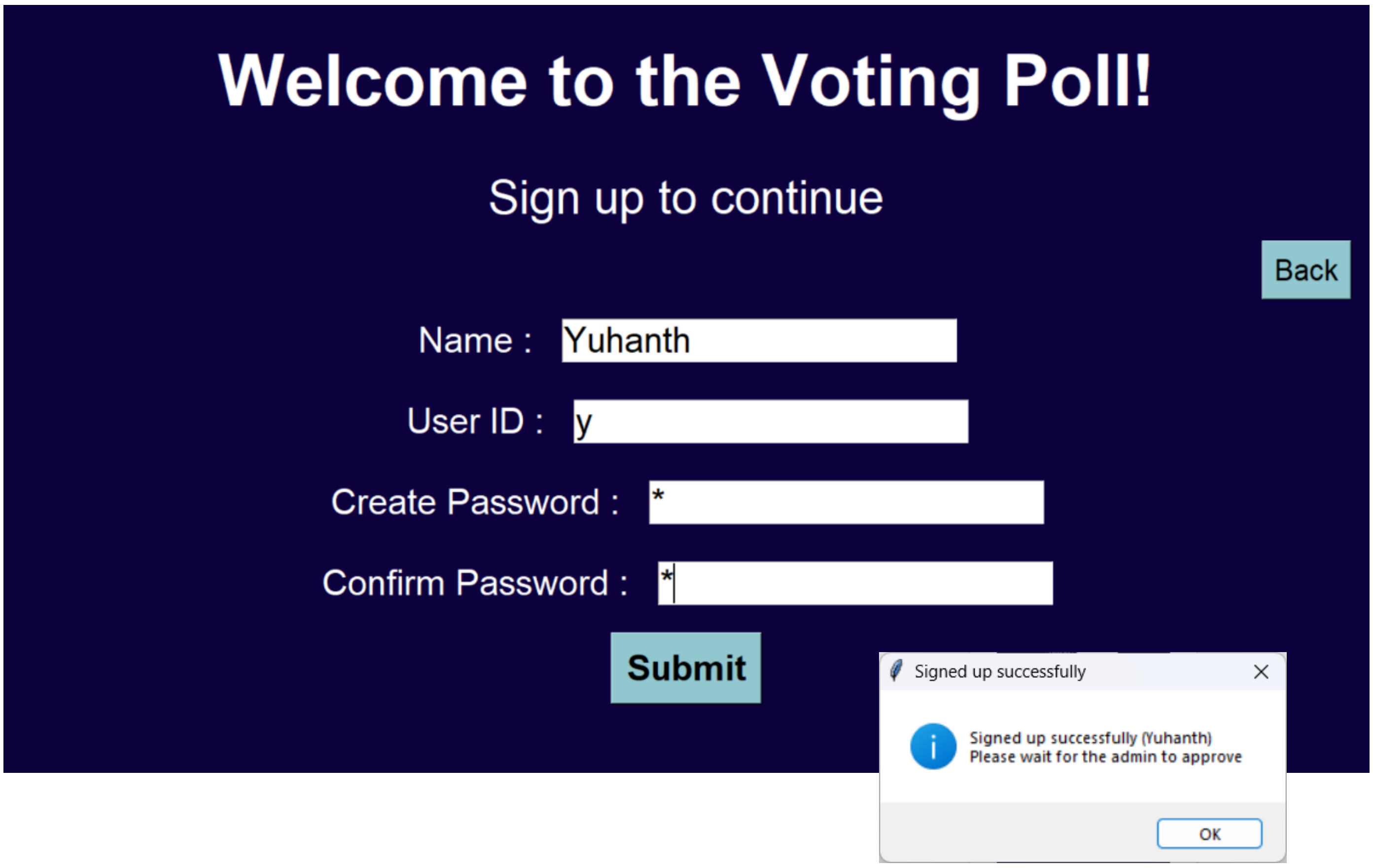
```
        Label(PollingWindow,text='Question :
'+question,font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*
3)
        Label(PollingWindow,text="Select option :
",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3)
        Options_var = TK.StringVar(PollingWindow)
        Options_var.set('Select option')
        Options_dropdown = OptionMenu(PollingWindow, Options_var, *opt_lis)
        Options_dropdown.config(font=TextFont1,width=int(fontsize*3.6))
        Options_dropdown['menu'].config(font=TextFont2)
        Options_dropdown['menu'].config(activebackground=ColourCodeButtonBG,
activeforeground=ColourCodeButtonFG)
        Options_dropdown.pack(pady=fontsize*3)
        Submit_button =
Button(PollingWindow,text="Submit",font=ButtonFont1,command=PollingCaller,bg=Colou
rCodeButtonBG,fg=ColourCodeButtonFG)
        Submit_button.pack(pady=fontsize*3)
        PollingWindow.attributes('-fullscreen', True)
        PollingWindow.resizable(False, False)
        PollingWindow.protocol("WM_DELETE_WINDOW",Tkin_closing)
    elif id_ver in voterid_lis and id_ver in voted_dict:
        messagebox.showerror('Error','\n'+voted_dict[id_ver][0]+' ('+id_ver+') has
already voted on '+voted_dict[id_ver][1])
    else:
        messagebox.showerror('Error','Invalid voter ID ('+id_ver+').')
def TkinPostPolling():
    global PostPollingWindow
    def back():
        PostPollingWindow.destroy()
        TkinVoterLogin()
    def SaveResultButtonAction():
        if Password_entry2.get()==current_PASS:
            if os.path.exists(current_poll+'_Result.txt'):
                PostPollingWindow.withdraw()
                TkinResultExists()
            else:
                SaveResult()
        else:
            SaveResult_button.config(state=TK.DISABLED)
            PollMenu_button.config(state=TK.DISABLED)
            messagebox.showerror('Log','Incorrect password')
            SaveResult_button.config(state=TK.NORMAL)
            PollMenu_button.config(state=TK.NORMAL)
    def PollMenuButtonAction():
        if Password_entry2.get()==current_PASS:
            PostPollingWindow.destroy()
            if current_designation=='Polling officer':
                POfficer_Window.deiconify()
            elif current_designation=='Polling agent':
                PAgent_Window.deiconify()
        else:
            SaveResult_button.config(state=TK.DISABLED)
            PollMenu_button.config(state=TK.DISABLED)
            messagebox.showerror('Log','Incorrect password')
            SaveResult_button.config(state=TK.NORMAL)
            PollMenu_button.config(state=TK.NORMAL)
    if current_designation=='Polling officer':
        PostPollingWindow = Toplevel(POfficer_Window)
    elif current_designation=='Polling agent':
        PostPollingWindow = Toplevel(PAgent_Window)
    VoterLoginWindow.destroy()
    PostPollingWindow.config(bg=ColourCodeMain)
    PostPollingWindow.title("Result Menu")
    Label(PostPollingWindow,text="Result
Menu",font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3
)
    Label(PostPollingWindow,text="Logged in as
"+users_dict[LogIn_user_ID][1],font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText)
.pack(pady=fontsize*3)
    Back_Button =
Button(PostPollingWindow,text="Back",font=ButtonFont3,command=back,bg=ColourCodeBu
ttonBG,fg=ColourCodeButtonFG)
    Back_Button.pack(padx=fontsize*4,side=TK.TOP,anchor=TK.NE)
    Label(PostPollingWindow,text="Password:",font=TextFont1,bg=ColourCodeMain,fg=C
olourCodeText).pack(pady=fontsize*3)
    Password_entry2 =
Entry(PostPollingWindow,show="*",width=fontsize*4,font=TextFont2)
    Password_entry2.pack(pady=fontsize*3)
    SaveResult_button = Button(PostPollingWindow,text="Save
Result",font=ButtonFont1,command=SaveResultButtonAction,bg=ColourCodeButtonBG,fg=C
olourCodeButtonFG)
    SaveResult_button.pack(pady=fontsize*3)
    PollMenu_button = Button(PostPollingWindow,text="Poll
Menu",font=ButtonFont1,command=PollMenuButtonAction,bg=ColourCodeButtonBG,fg=Colou
rCodeButtonFG)
    PollMenu_button.pack(pady=fontsize*3)
    PostPollingWindow.attributes('-fullscreen', True)
    PostPollingWindow.resizable(False, False)
    PostPollingWindow.protocol("WM_DELETE_WINDOW",Tkin_closing)
def TkinResultExists():
    def Overwrite():
        try:
            os.remove(current_poll+'_Result.txt')
            SaveResult()
            ResultExists_Window.destroy()
        except:
            messagebox.showerror('Error','Error while overwiting.')
    def DontOverwrite():
        ResultExists_Window.destroy()
        PostPollingWindow.deiconify()
    ResultExists_Window = Toplevel(PostPollingWindow)
    ResultExists_Window.config(bg=ColourCodeMain)
    ResultExists_Window.title("Result exists")
    Label(ResultExists_Window,text='Result already saved
('+current_poll+')',font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(p
ady=fontsize*3)
    Label(ResultExists_Window,text="Do you want to
overwrite?",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize
*3)
    Overwrite_Button=Button(ResultExists_Window,text="Yes
(Overwrite)",font=ButtonFont1,command=Overwrite,bg=ColourCodeButtonBG,fg=ColourCod
eButtonFG)
    Overwrite_Button.pack(pady=fontsize*3)
    DontOverwrite_Button=Button(ResultExists_Window,text="No
(Back)",font=ButtonFont1,command=DontOverwrite,bg=ColourCodeButtonBG,fg=ColourCode
ButtonFG)
    DontOverwrite_Button.pack(pady=fontsize*3)
    ResultExists_Window.attributes('-fullscreen', True)
    ResultExists_Window.resizable(False,False)
    ResultExists_Window.protocol("WM_DELETE_WINDOW",Tkin_closing)
def TkinPAgent():
    global PAgent_Window,filename_entry,ContinuePoll_button
    def Logout():
        global LoggedIn_Flag
        LoggedIn_Flag=False
        PAgent_Window.destroy()
        login_button.config(state=TK.NORMAL)
        LoginWindow.deiconify()
        LoginWindow.bind('<Return>', LogIn_on_enter)
    PAgent_Window = Toplevel(LoginWindow)
    PAgent_Window.config(bg=ColourCodeMain)
    PAgent_Window.title("Continue Poll")
    Label(PAgent_Window,text="Polling agent's
Menu",font=TitleTextFont,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3
)
    Label(PAgent_Window,text="Logged in as "+users_dict[LogIn_user_ID][1]+' -
Polling
agent',font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3)
    LogOut_button =
Button(PAgent_Window,text="Logout",command=Logout,font=ButtonFont3,bg=ColourCodeBu
ttonBG,fg=ColourCodeButtonFG)
    LogOut_button.pack(pady=fontsize*3,padx=fontsize*2,side=TK.TOP,anchor=TK.NE)
    Label(PAgent_Window,text="Enter name of the
poll",font=TextFont1,bg=ColourCodeMain,fg=ColourCodeText).pack(pady=fontsize*3)
```

```
    filename_entry = Entry(PAgent_Window,font=TextFont1)
    filename_entry.pack(pady=fontsize*3)
    ContinuePoll_button = Button(PAgent_Window,text="Continue
Poll",font=ButtonFont1,command=ContinuePoll,bg=ColourCodeButtonBG,fg=ColourCodeBut
tonFG)
    ContinuePoll_button.pack(pady=fontsize*3)
    PAgent_Window.attributes('-fullscreen',True)
    PAgent_Window.resizable(False,False)
    PAgent_Window.protocol("WM_DELETE_WINDOW",Tkin_closing)
MAIN()
```

Output


Welcome to the Voting Poll!
Login
Sign up
Preferences
Exit
-Yuhanth
XII - JEE


Preference Menu
Make sure '] [' is inside the screen
]
[
Back
Increase Font Size (+)
Decrease Font Size (-)
Font family :
Arial
Theme :
Default


Welcome to the Voting Poll!

Sign up to continue

Name : Yuhanth

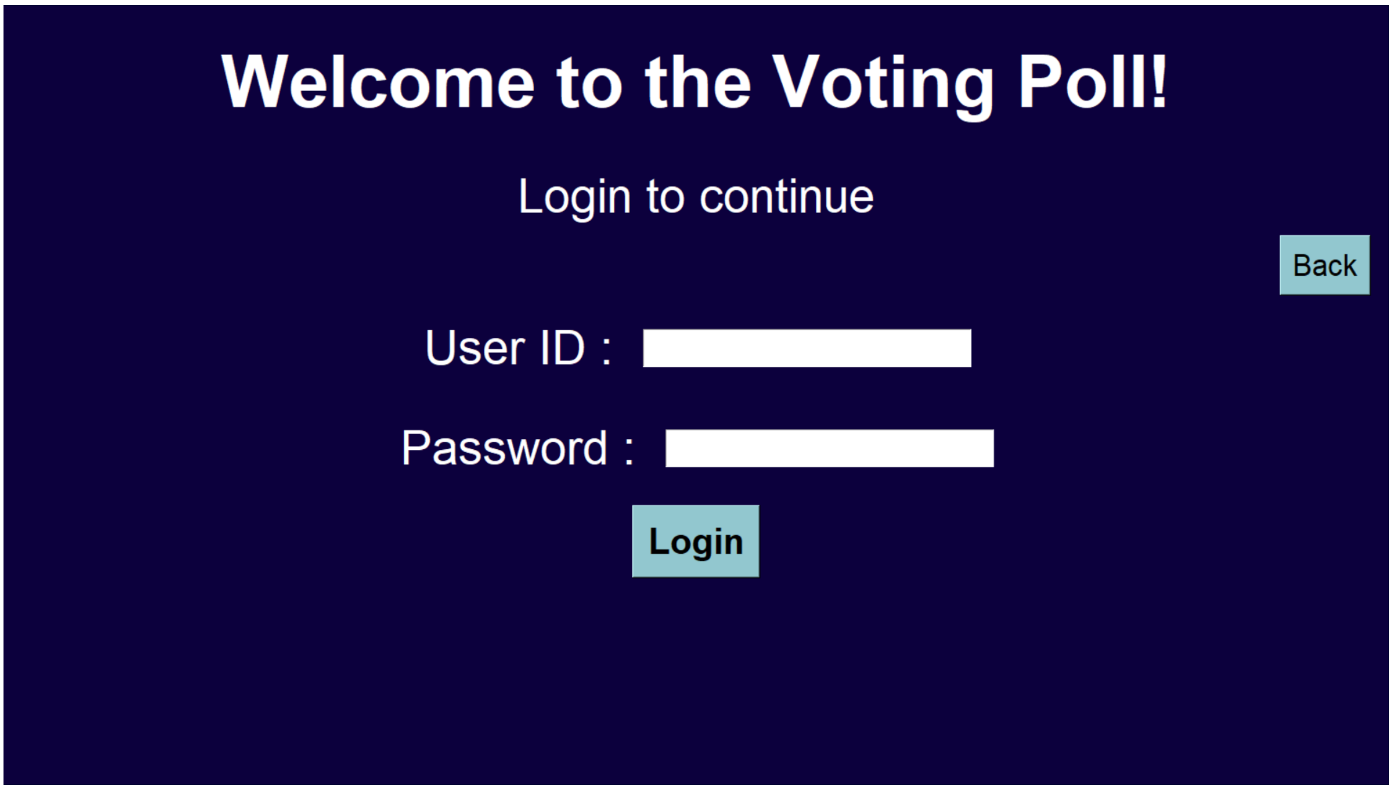
User ID : y

Create Password : *

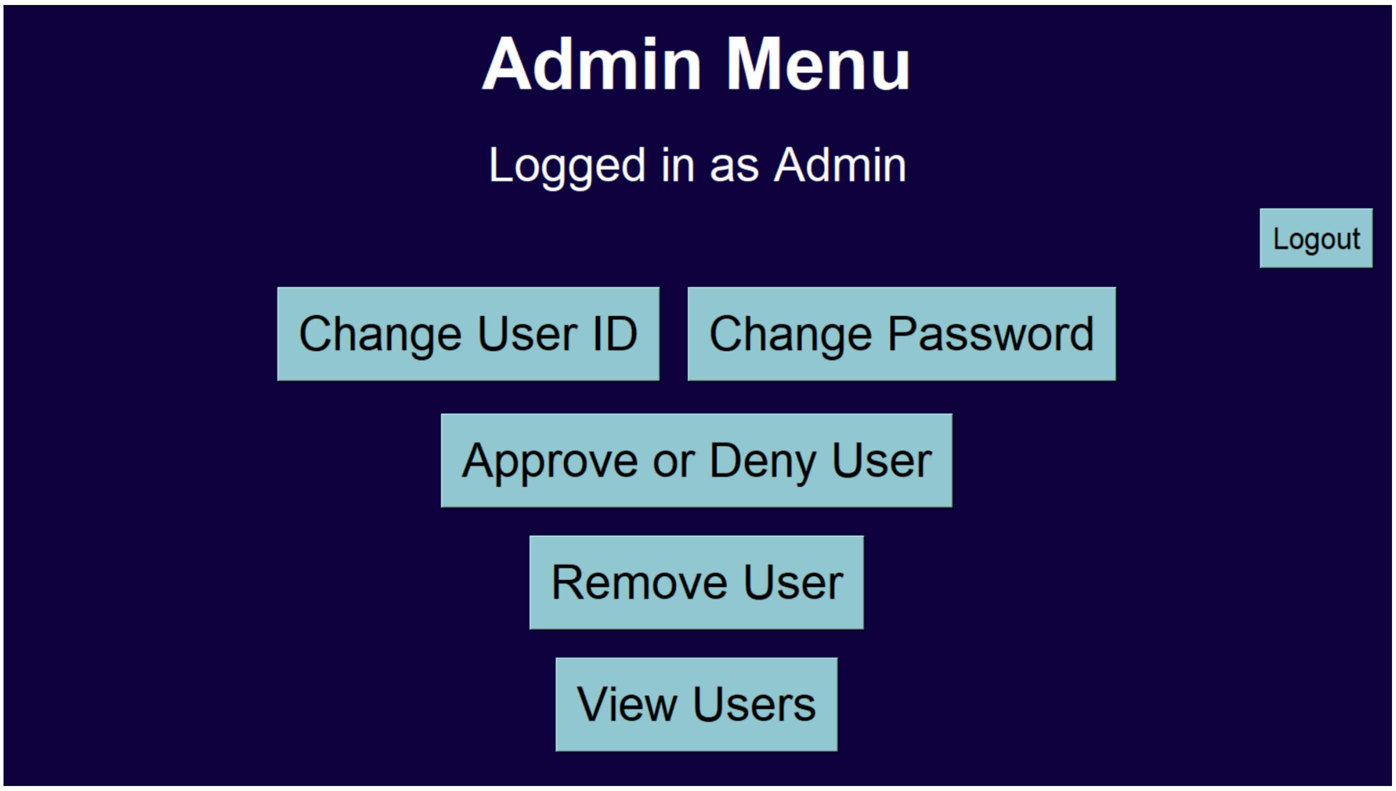
Confirm Password : *

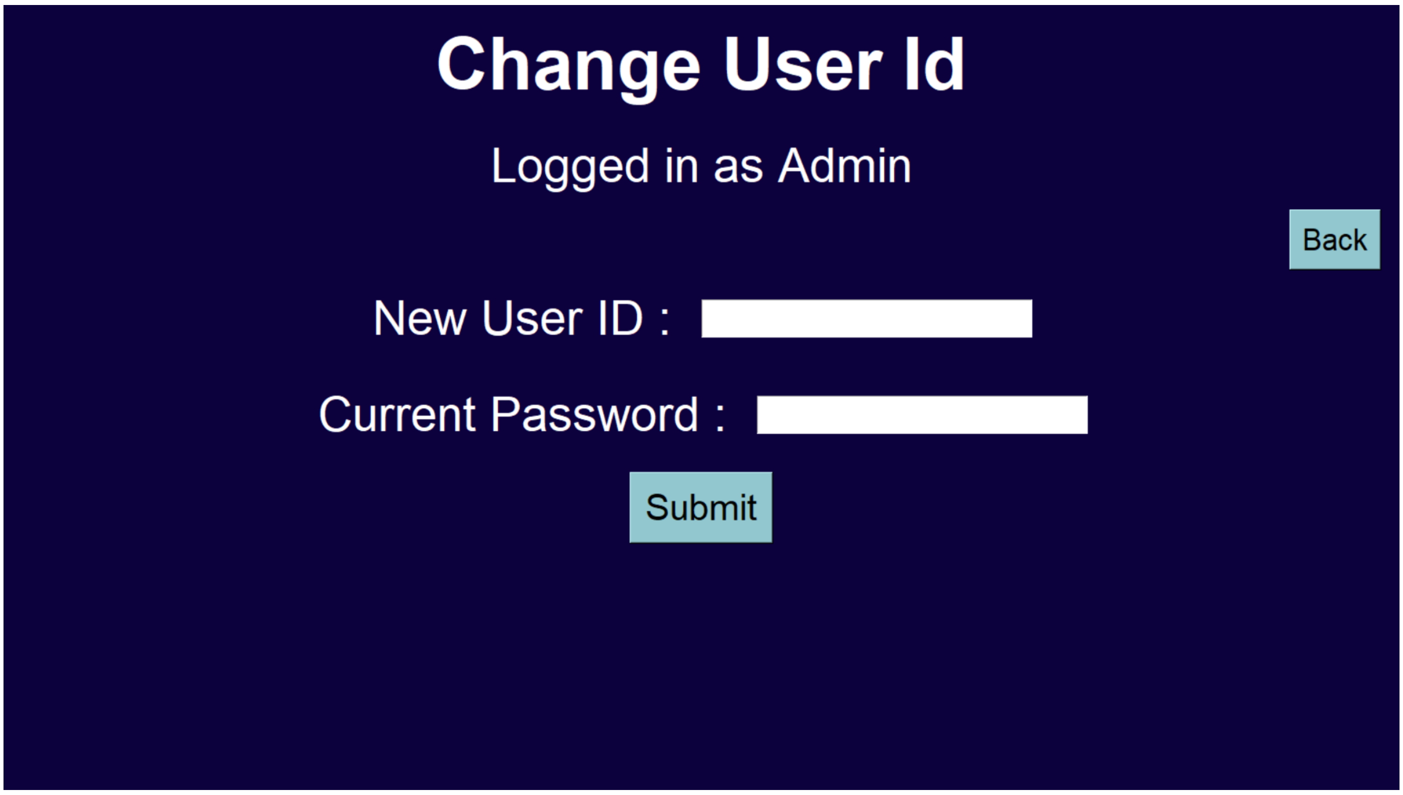
Signed up successfully

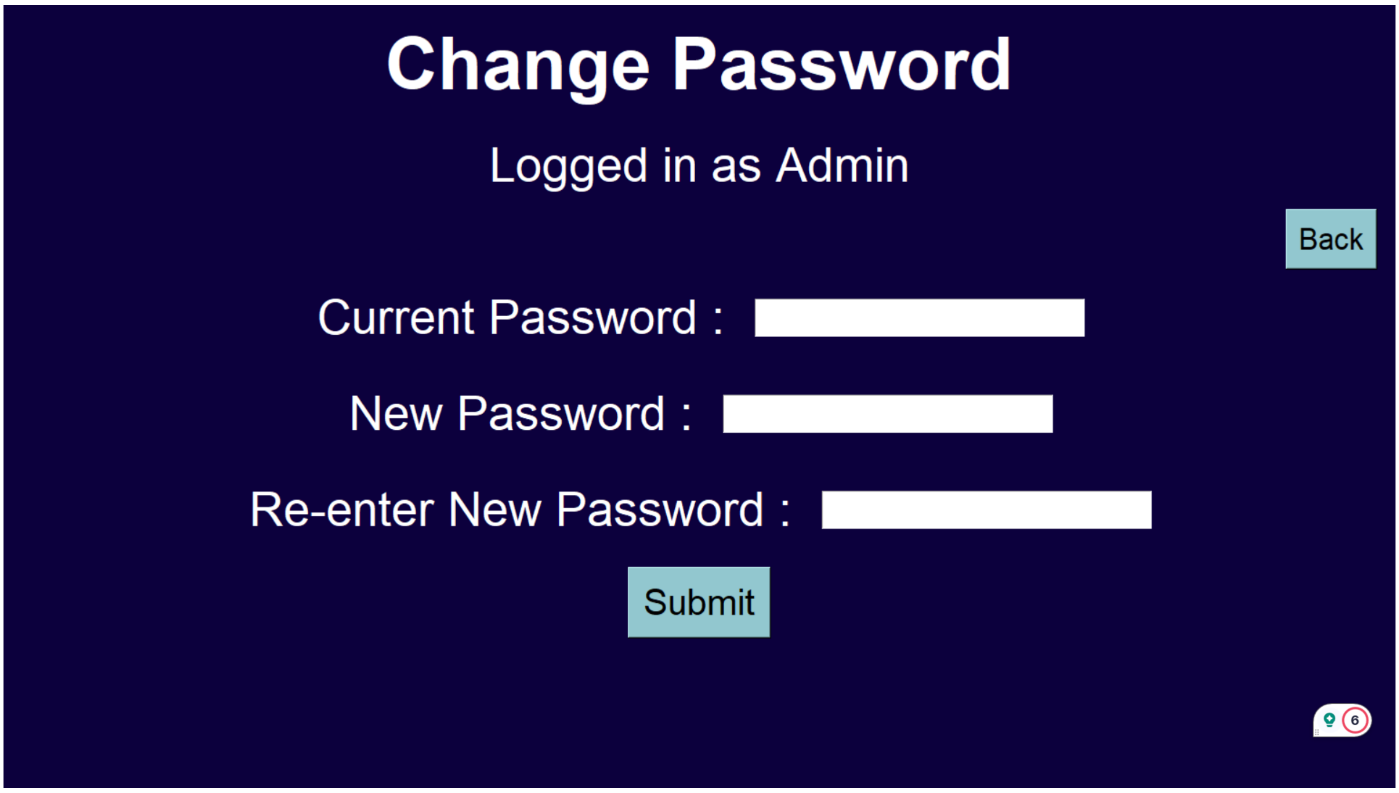
Signed up successfully (Yuhanth)
Please wait for the admin to approve


Welcome to the Voting Poll!

Login to continue


Admin Menu
Logged in as Admin
Logout
Change User ID
Change Password
Approve or Deny User
Remove User
View Users


Change User Id
Logged in as Admin
Back
New User ID :
Current Password :
Submit



Change Password

Logged in as Admin

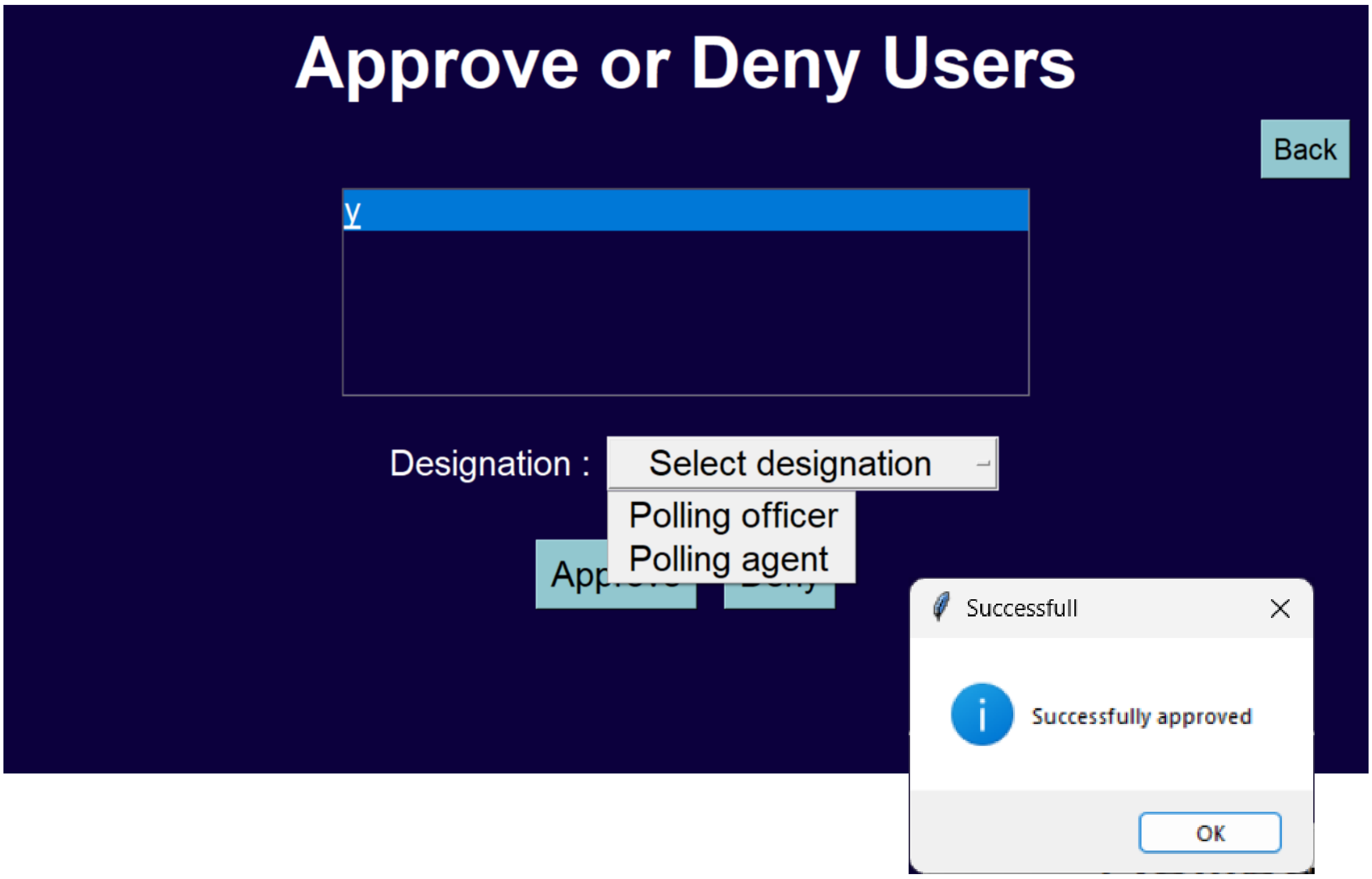
Back

Current Password :

New Password :

Re-enter New Password :

Submit



Approve or Deny Users

Back

y

Designation : Select designation

Polling officer

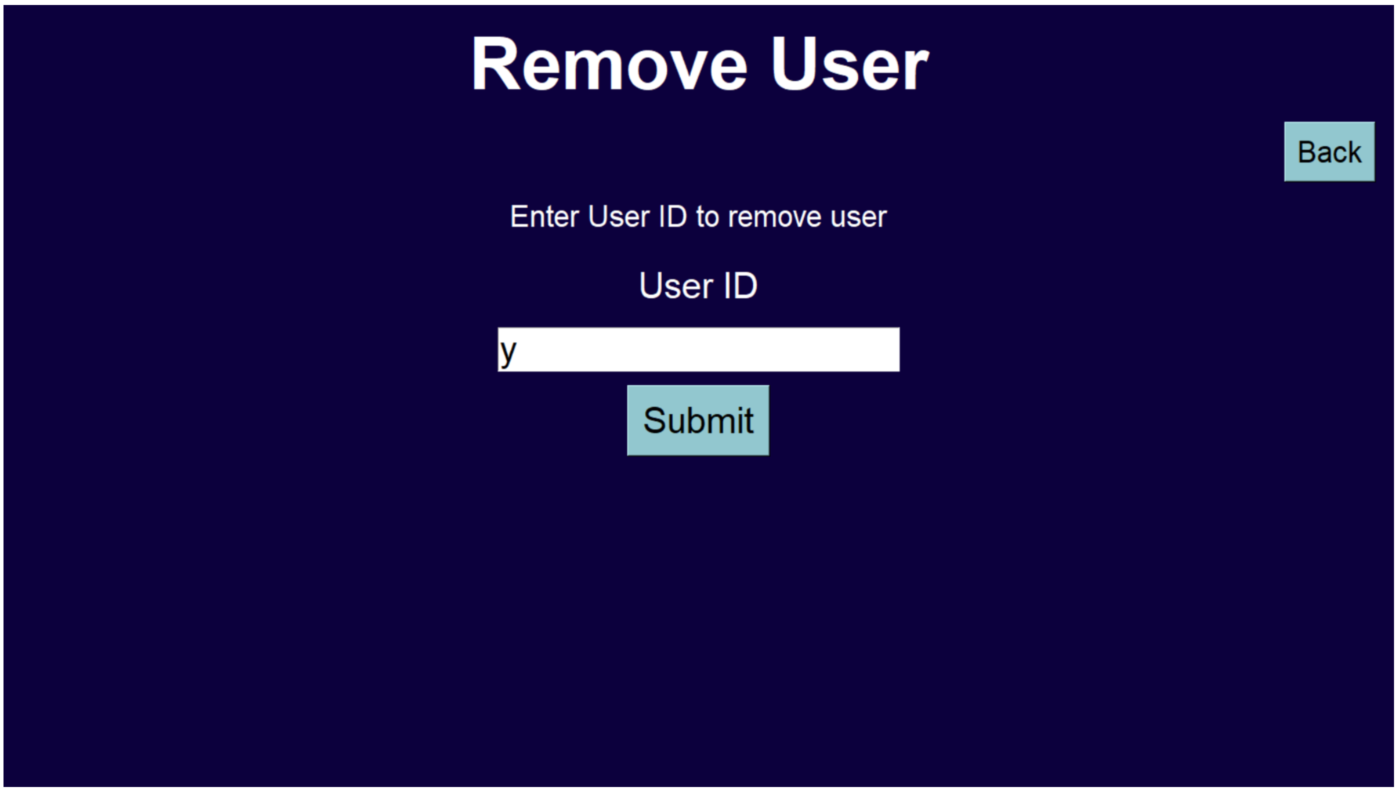
Polling agent

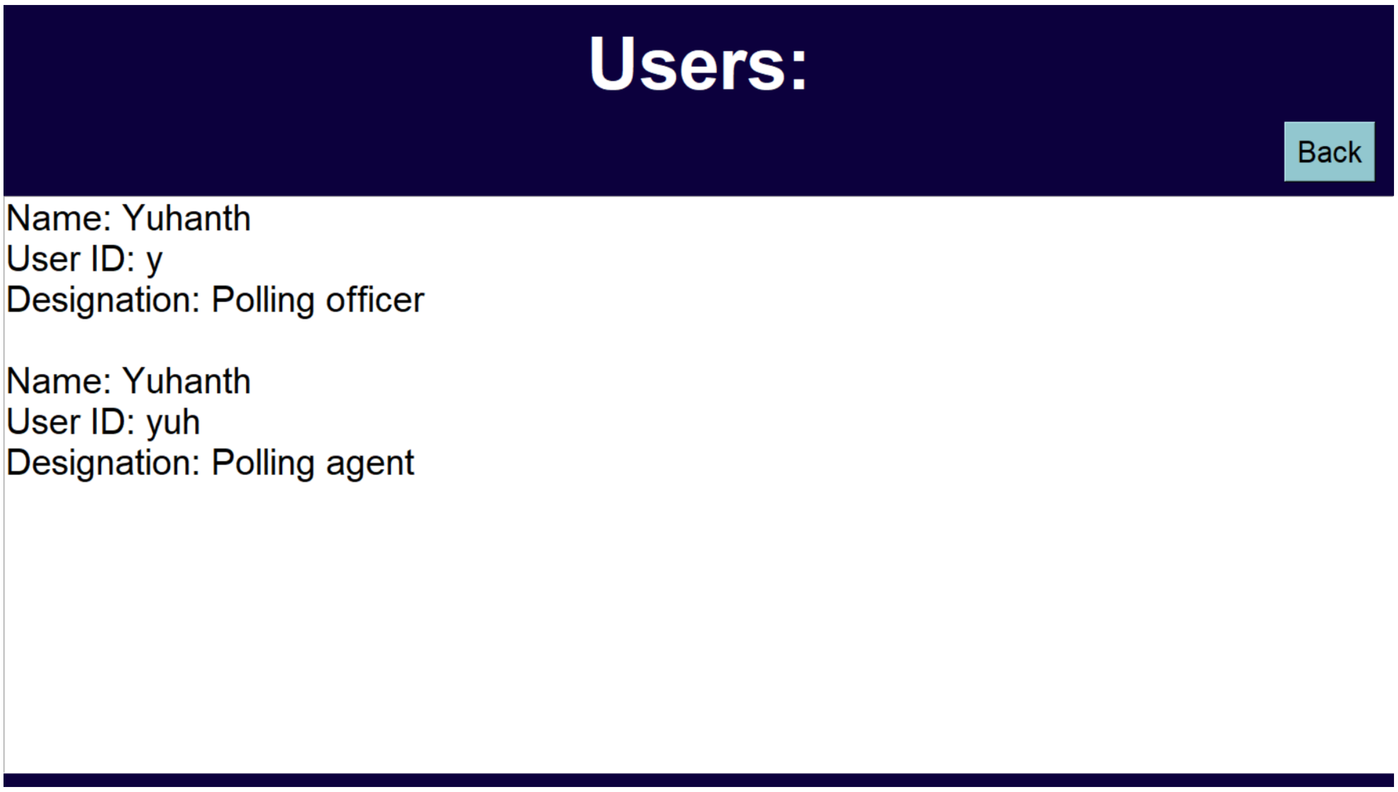
App

Successfull

Successfully approved

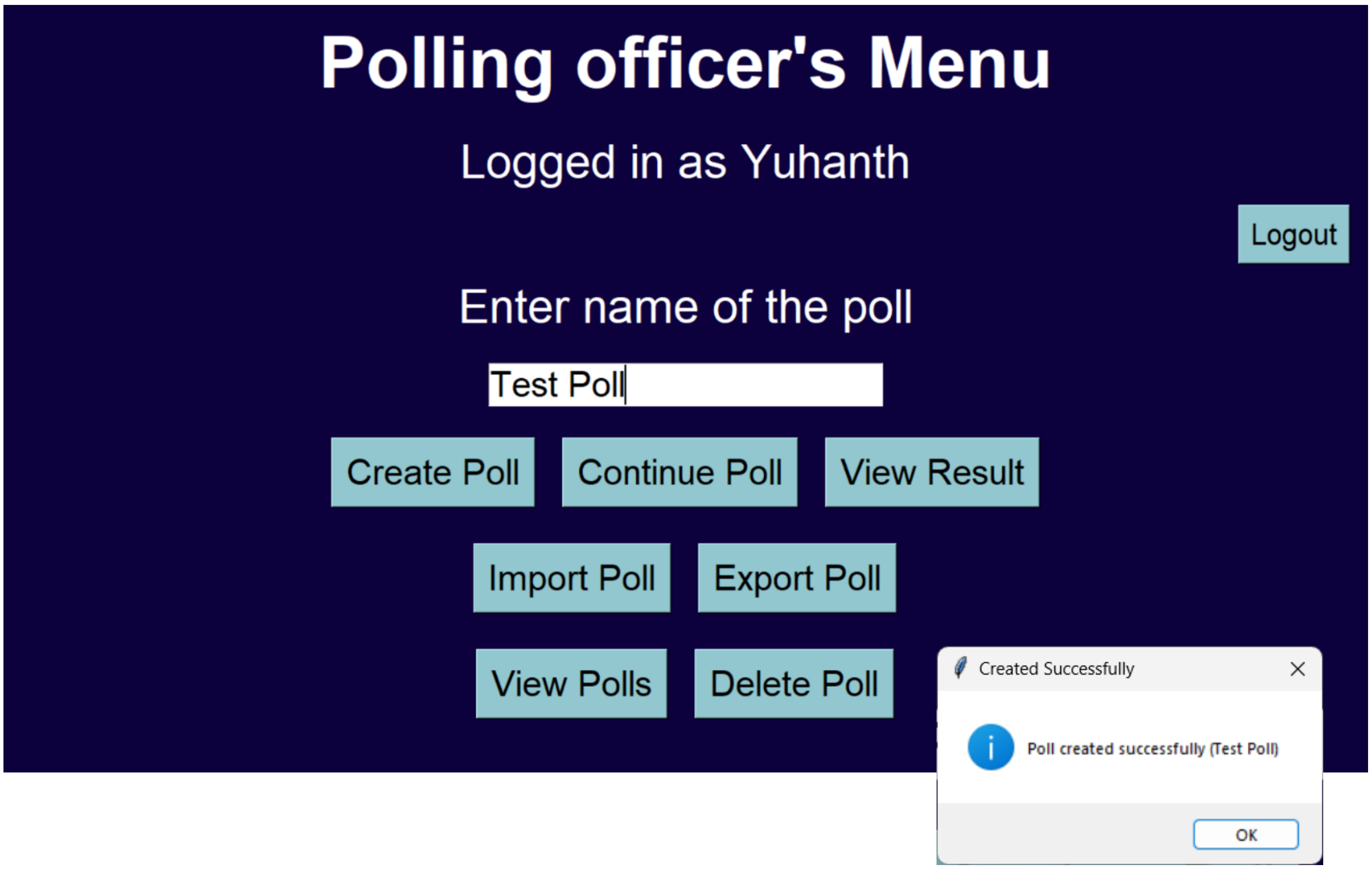
OK





Name: Yuhanth
User ID: y
Designation: Polling officer

Name: Yuhanth
User ID: yuh
Designation: Polling agent


Polling officer's Menu

Logged in as Yuhanth

Logout

Enter name of the poll

Test Poll

Create Poll | Continue Poll | View Result

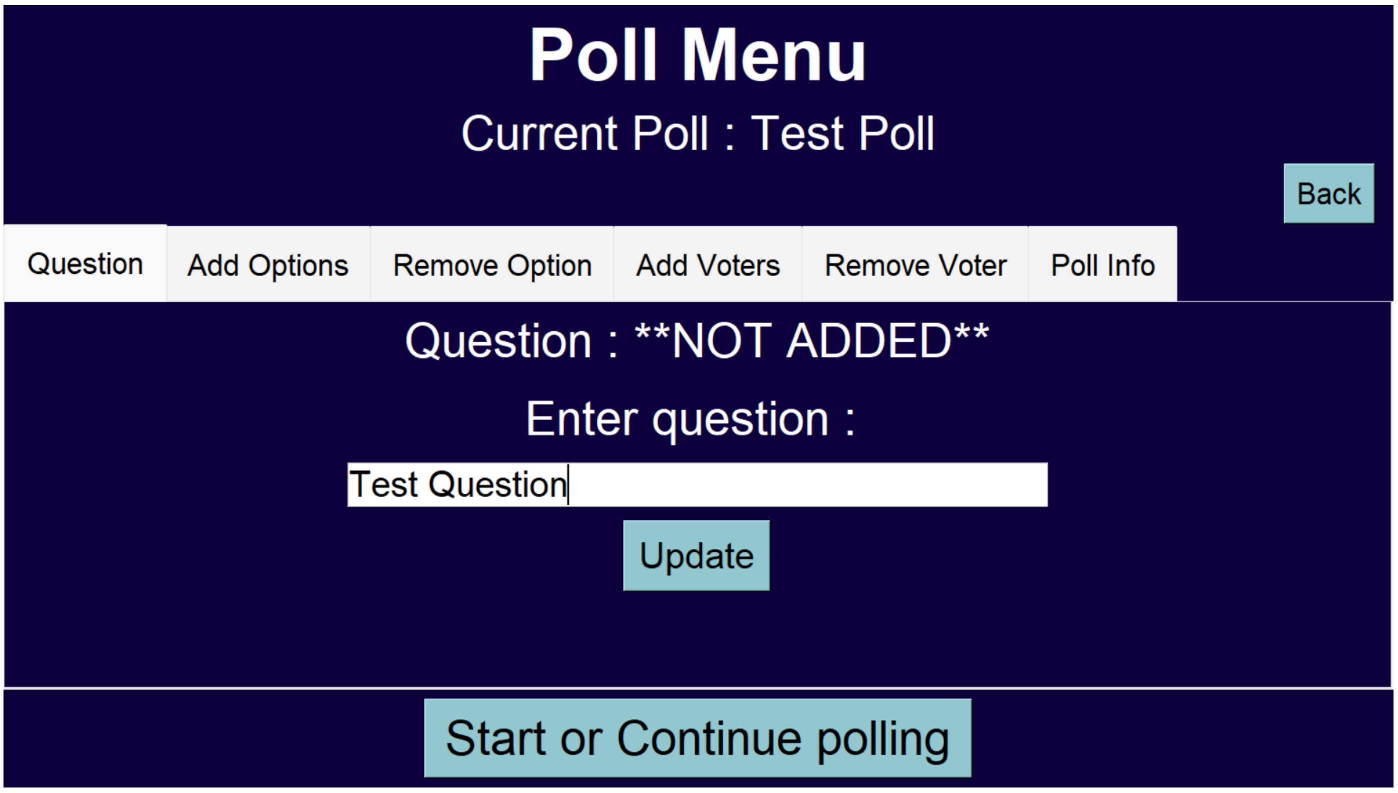
Import Poll | Export Poll

View Polls | Delete Poll

Created Successfully

Poll created successfully (Test Poll)

OK


Poll Menu

Current Poll : Test Poll

Back

Question | Add Options | Remove Option | Add Voters | Remove Voter | Poll Info

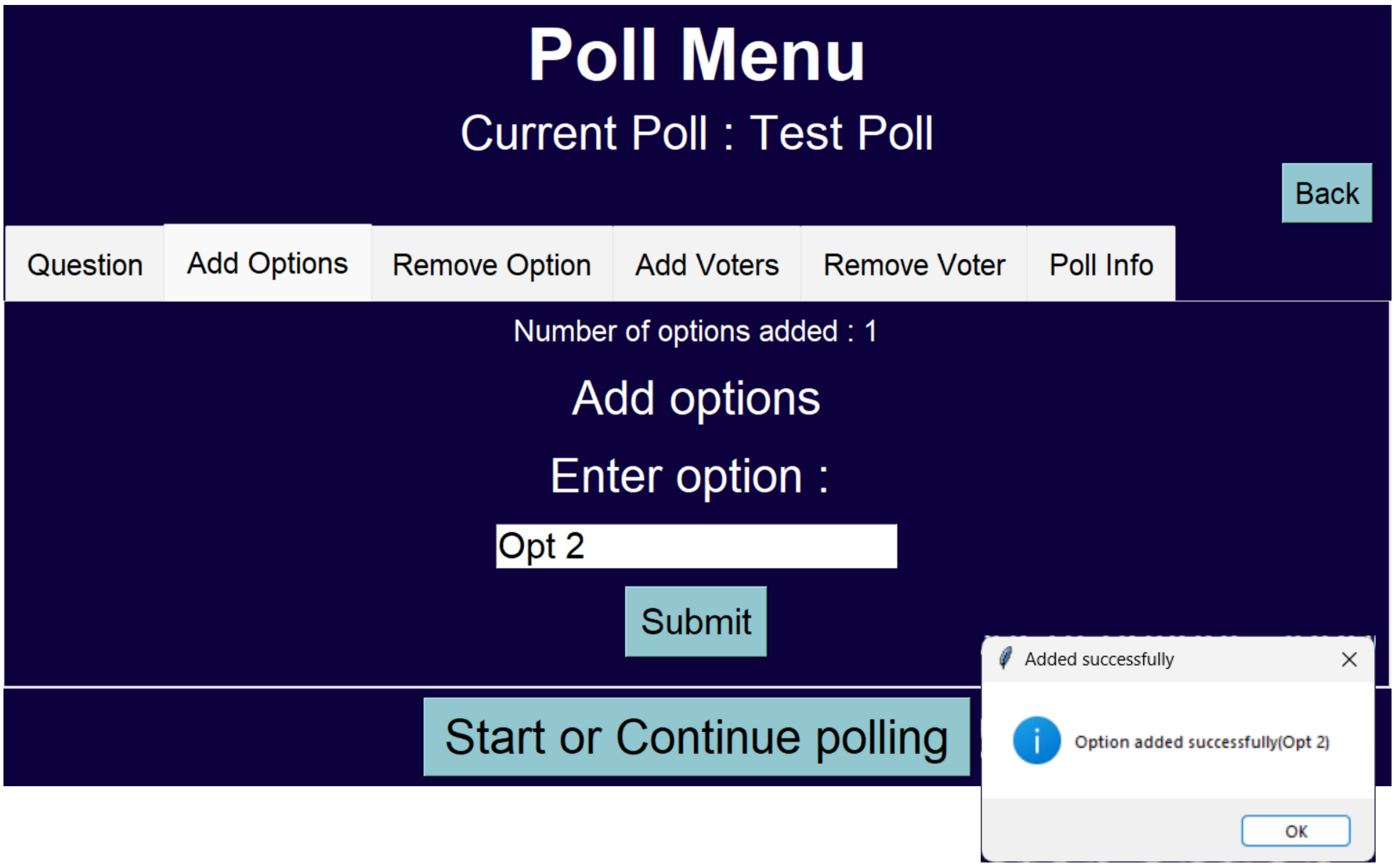
Question : **NOT ADDED**

Enter question :

Test Question

Update

Start or Continue polling



Poll Menu

Current Poll : Test Poll

Back

Question | Add Options | Remove Option | Add Voters | Remove Voter | Poll Info

Number of options added : 1

Add options

Enter option :

Opt 2

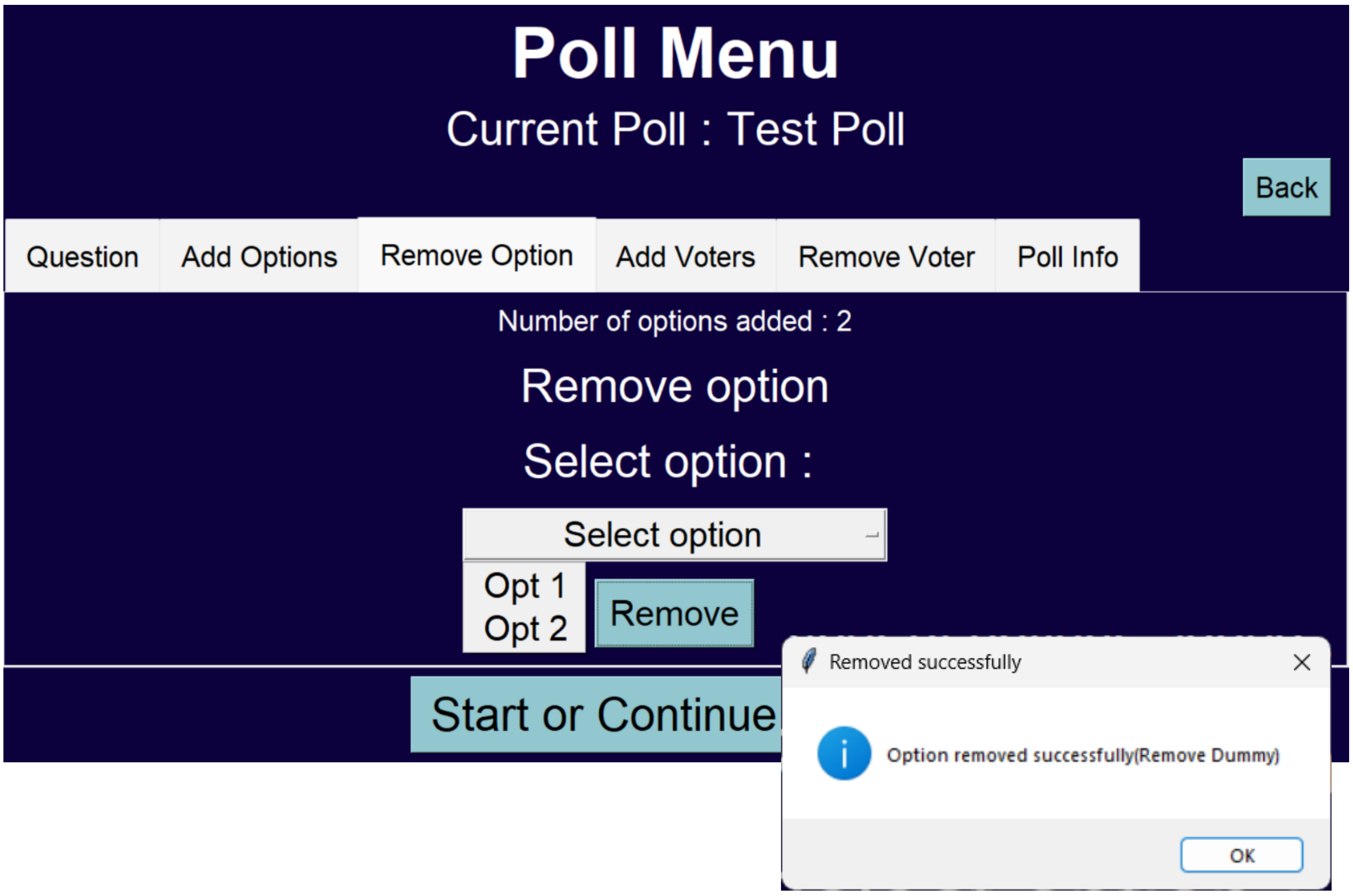
Submit

Start or Continue polling

Added successfully

Option added successfully(Opt 2)

OK



Poll Menu

Current Poll : Test Poll

Back

Question | Add Options | Remove Option | Add Voters | Remove Voter | Poll Info

Number of options added : 2

Remove option

Select option :

Select option

Opt 1
Opt 2

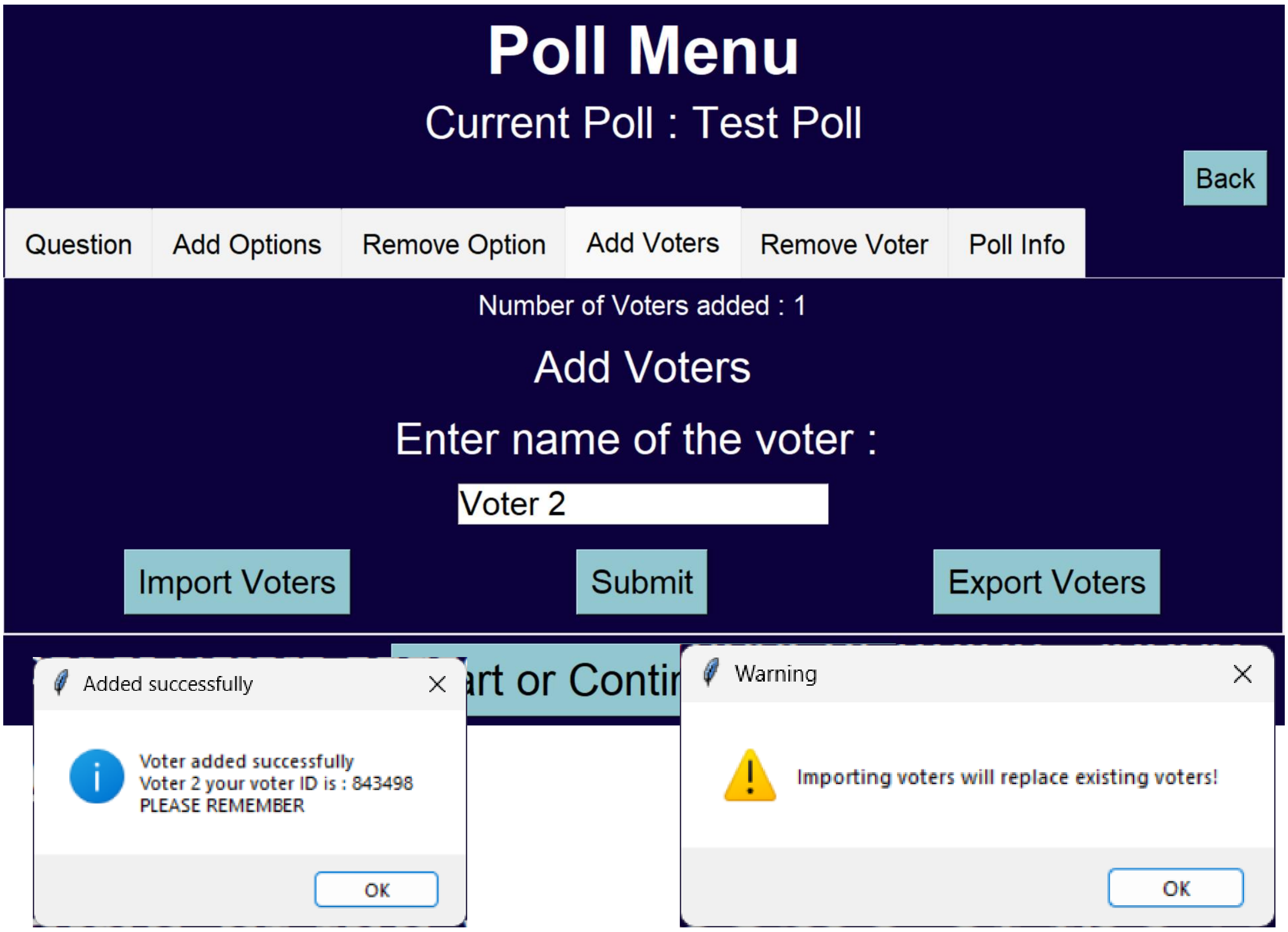
Remove

Start or Continue

Removed successfully

Option removed successfully(Remove Dummy)

OK



Poll Menu

Current Poll : Test Poll

Number of Voters added : 1

Add Voters

Enter name of the voter :

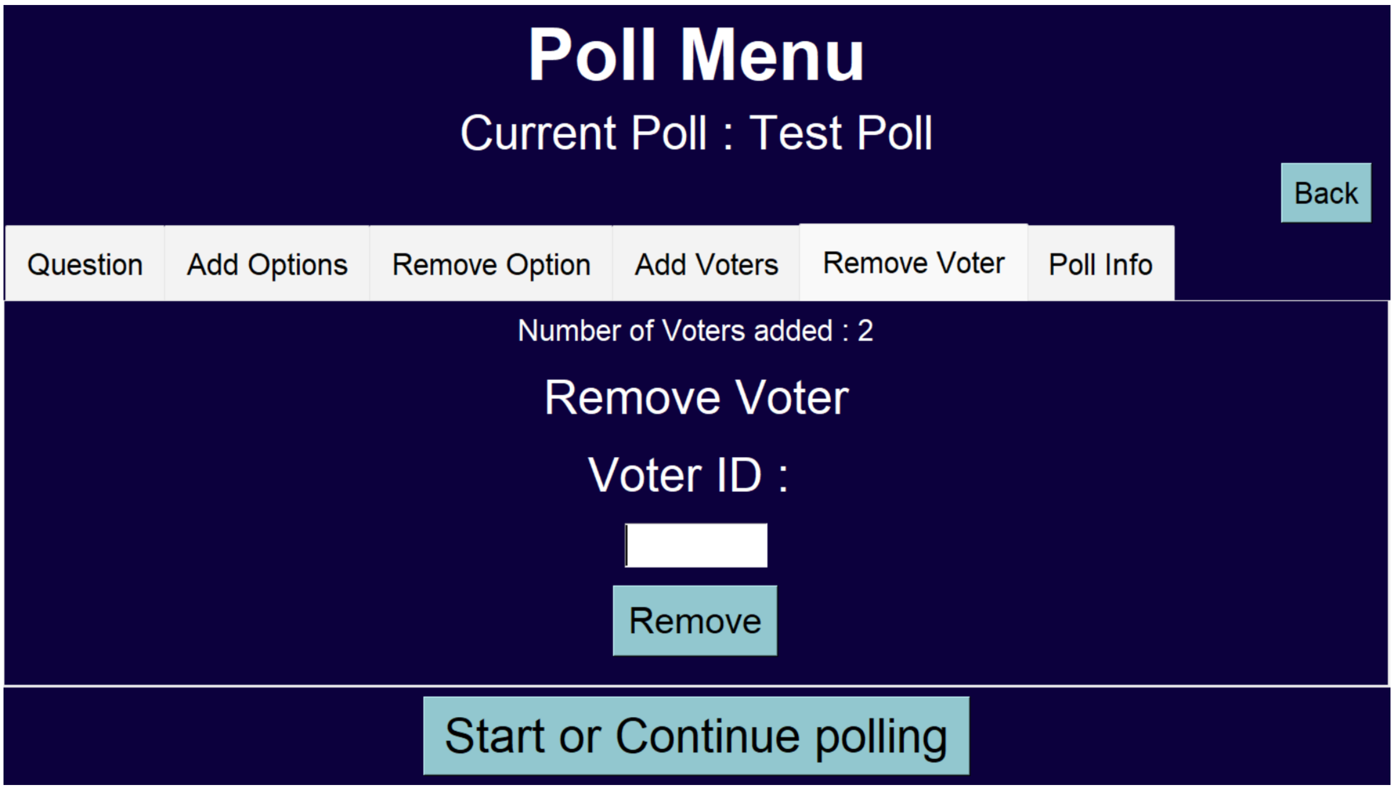
Voter 2

Added successfully

Voter added successfully
Voter 2 your voter ID is : 843498
PLEASE REMEMBER

Warning

Importing voters will replace existing voters!



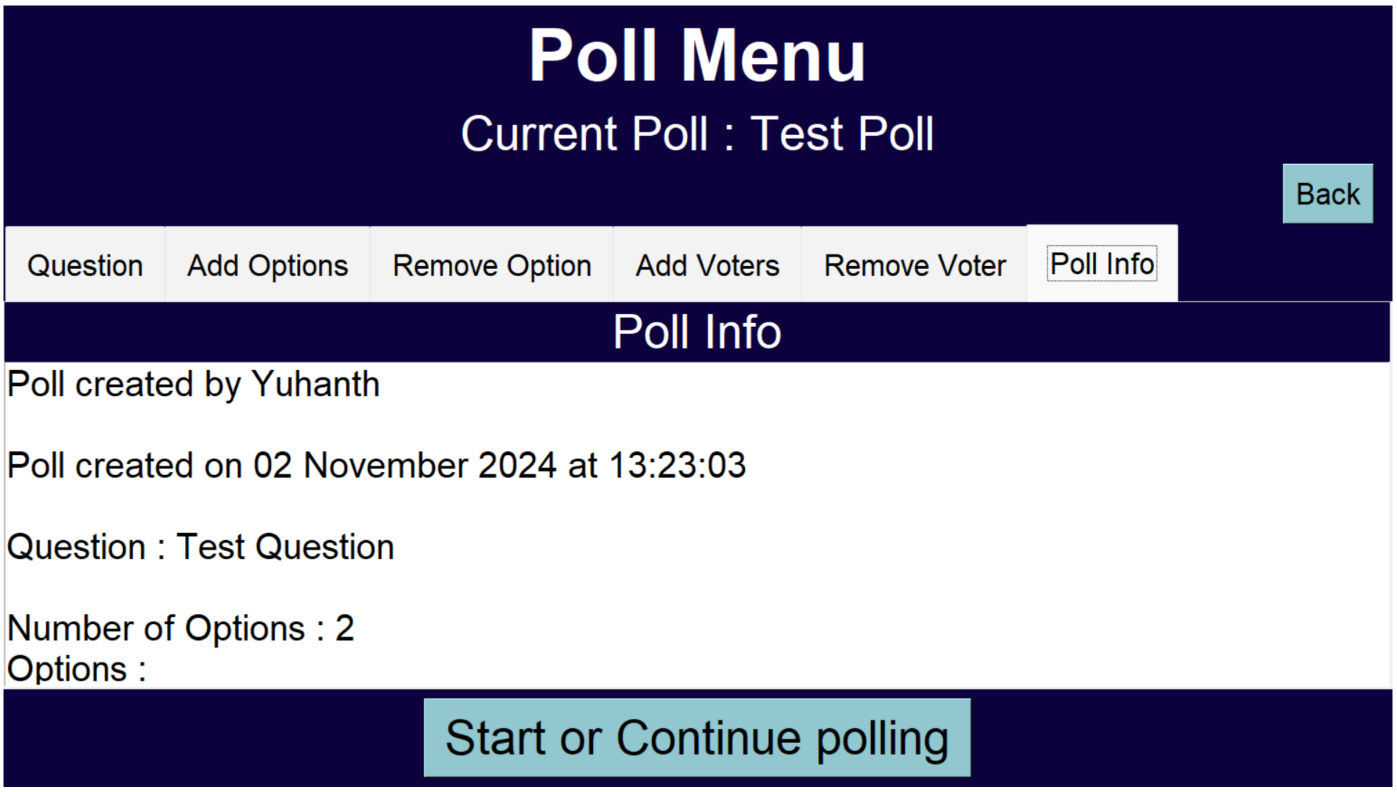
Poll Menu

Current Poll : Test Poll

Number of Voters added : 2

Remove Voter

Voter ID :

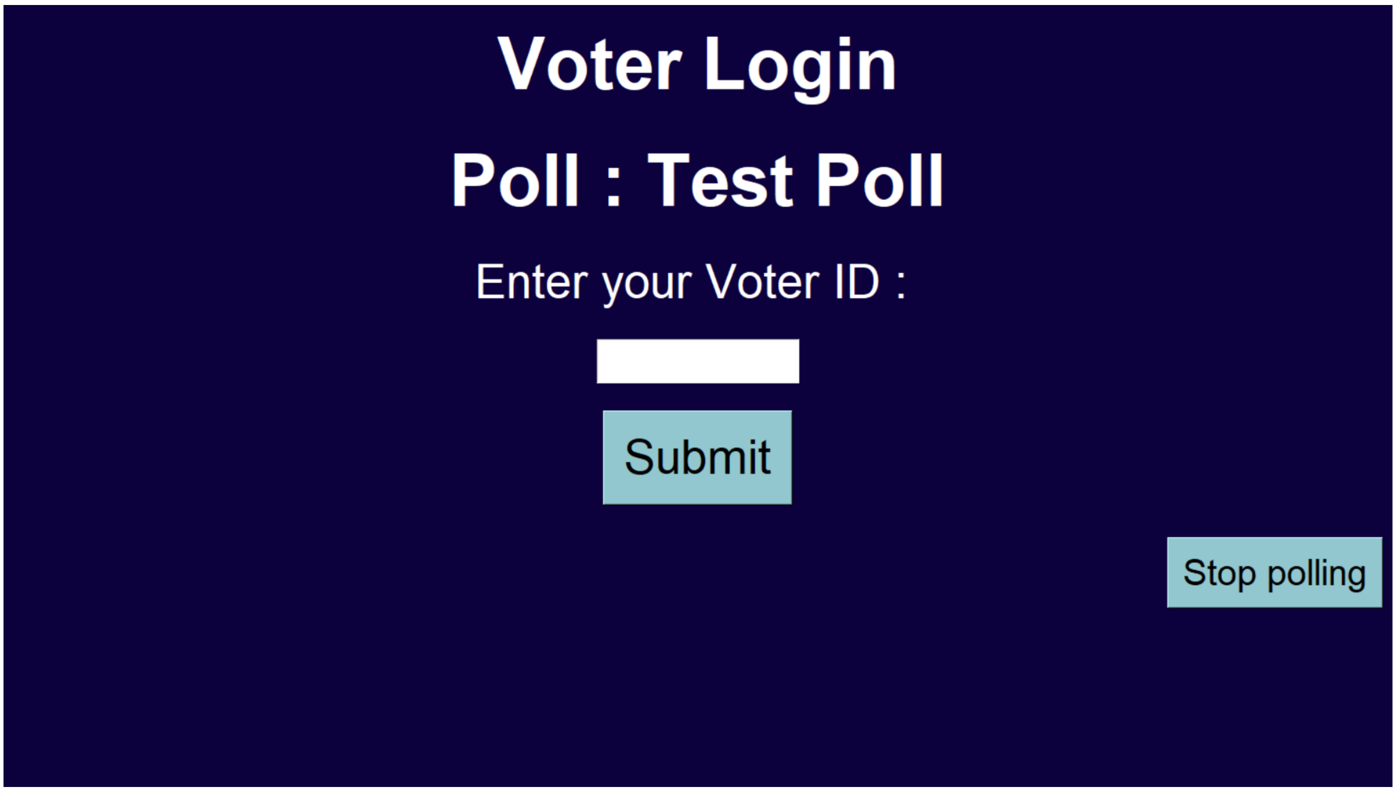


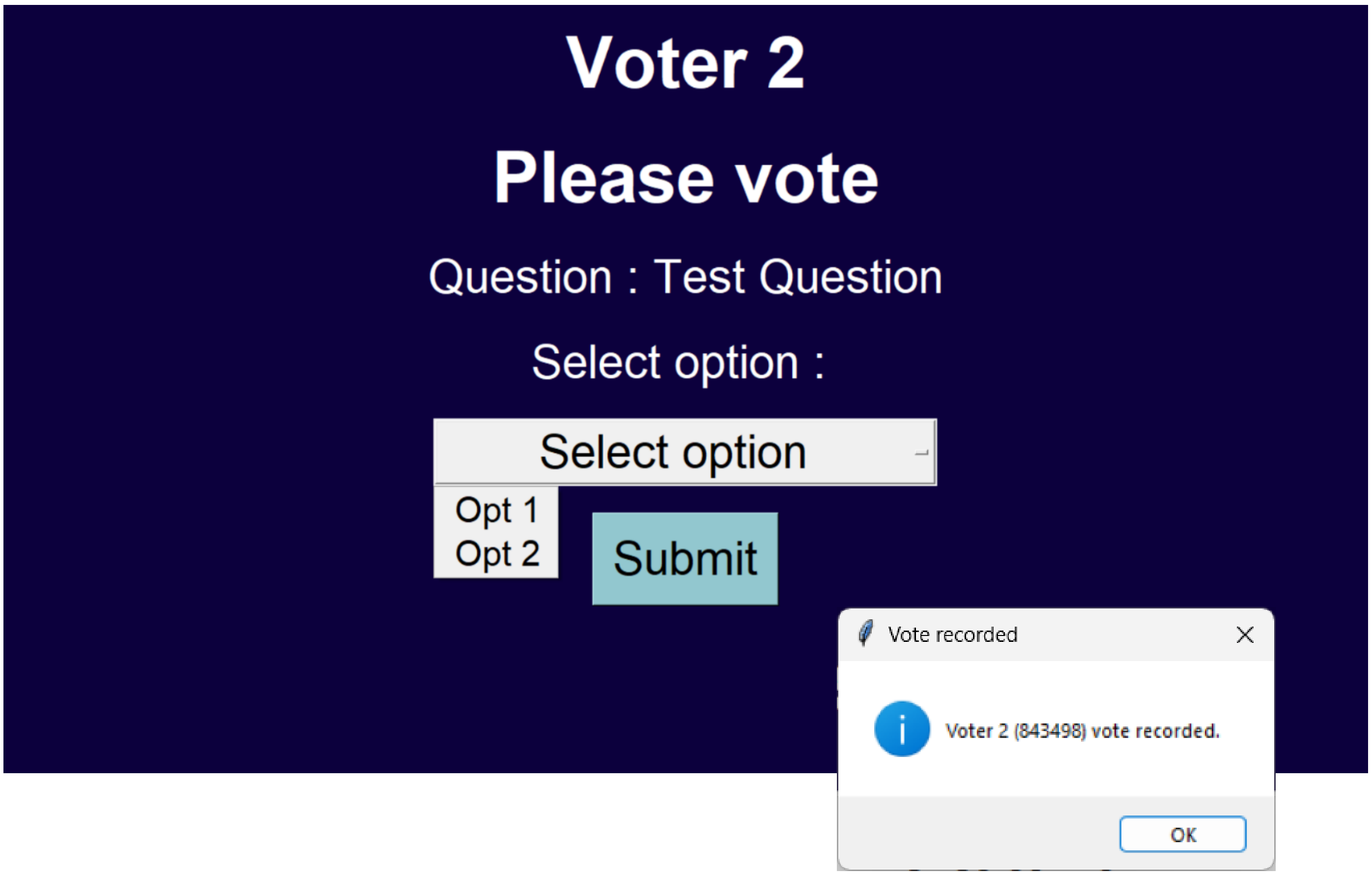
Poll created by Yuhanth

Poll created on 02 November 2024 at 13:23:03

Question : Test Question

Number of Options : 2
Options :





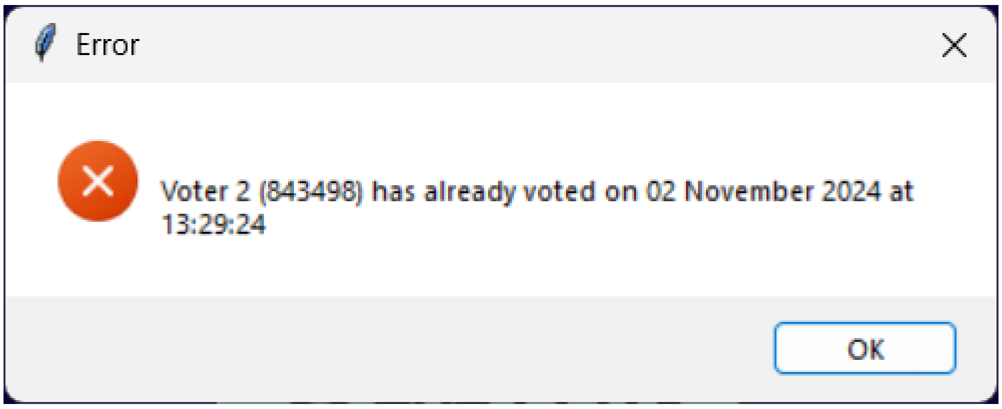
Voter 2

Please vote

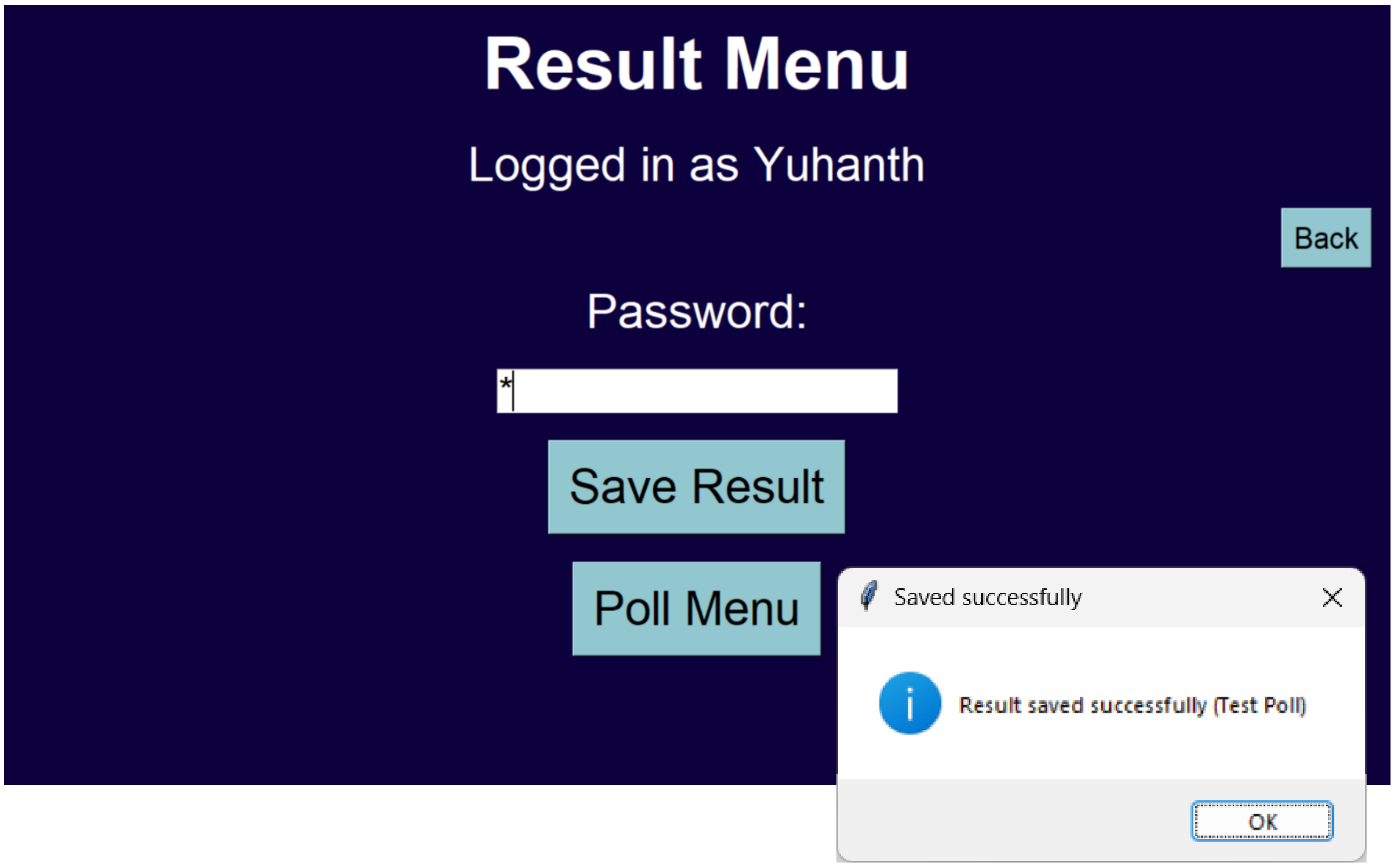
Question : Test Question

Select option :

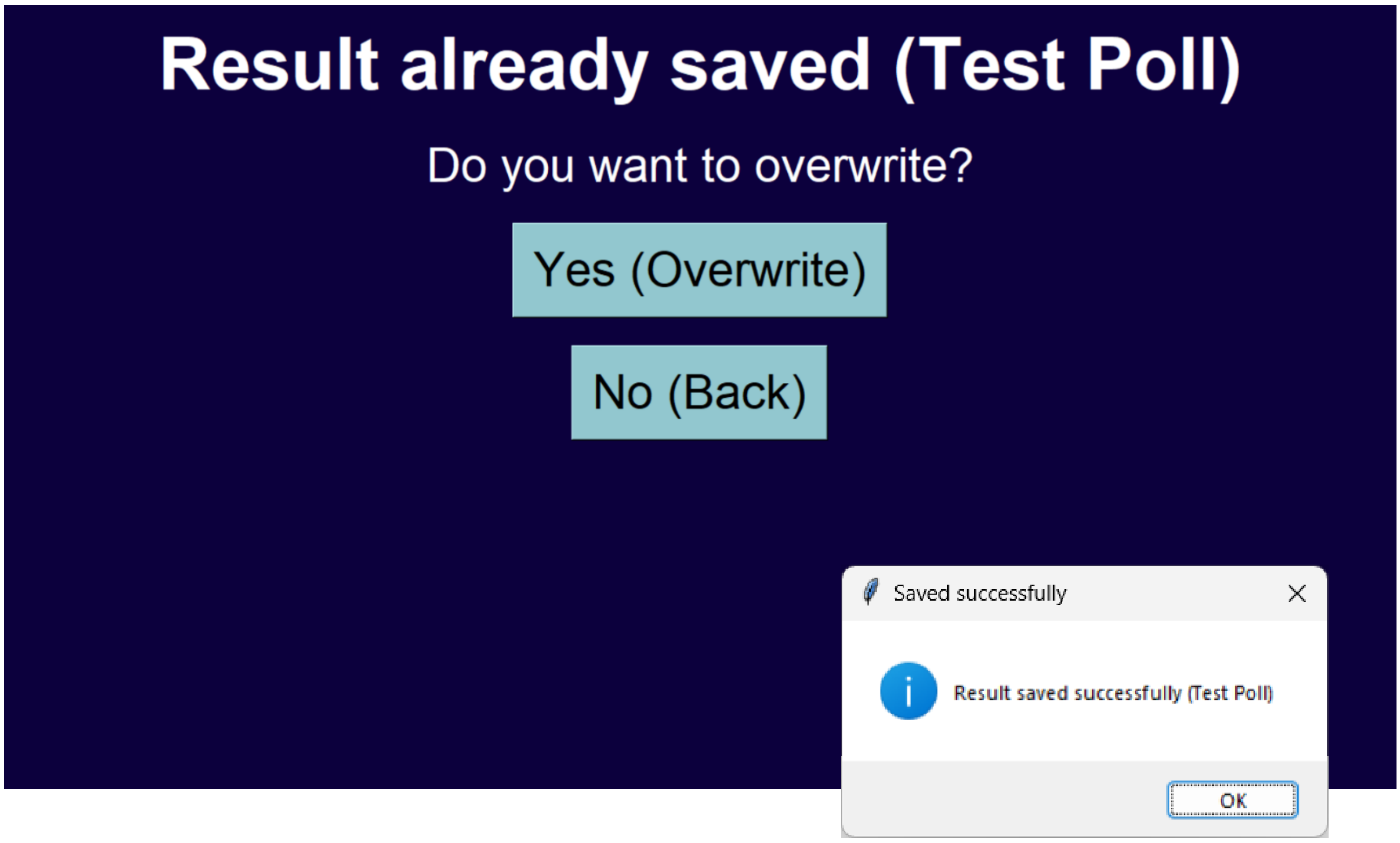
Voter 2 (843498) vote recorded.



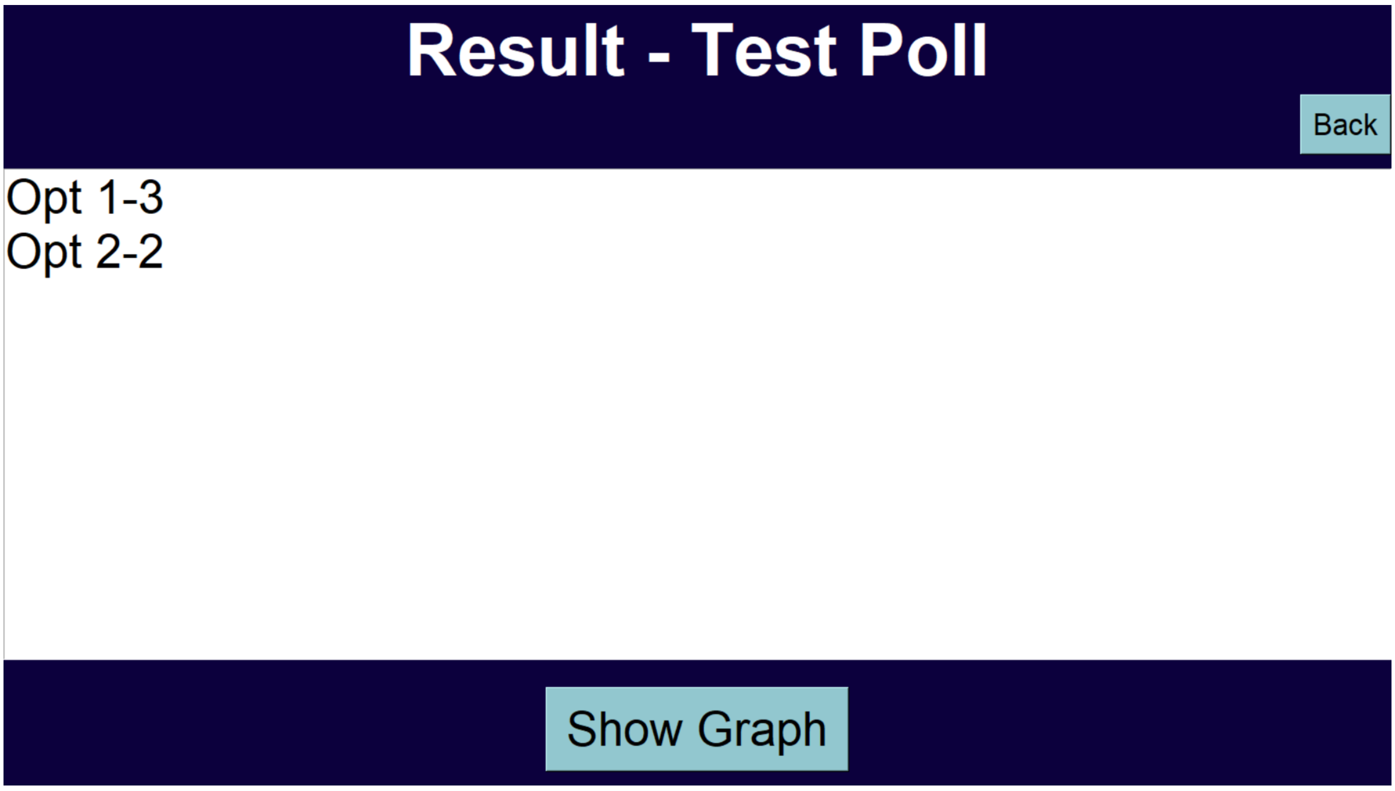
Voter 2 (843498) has already voted on 02 November 2024 at 13:29:24



Result saved successfully (Test Poll)



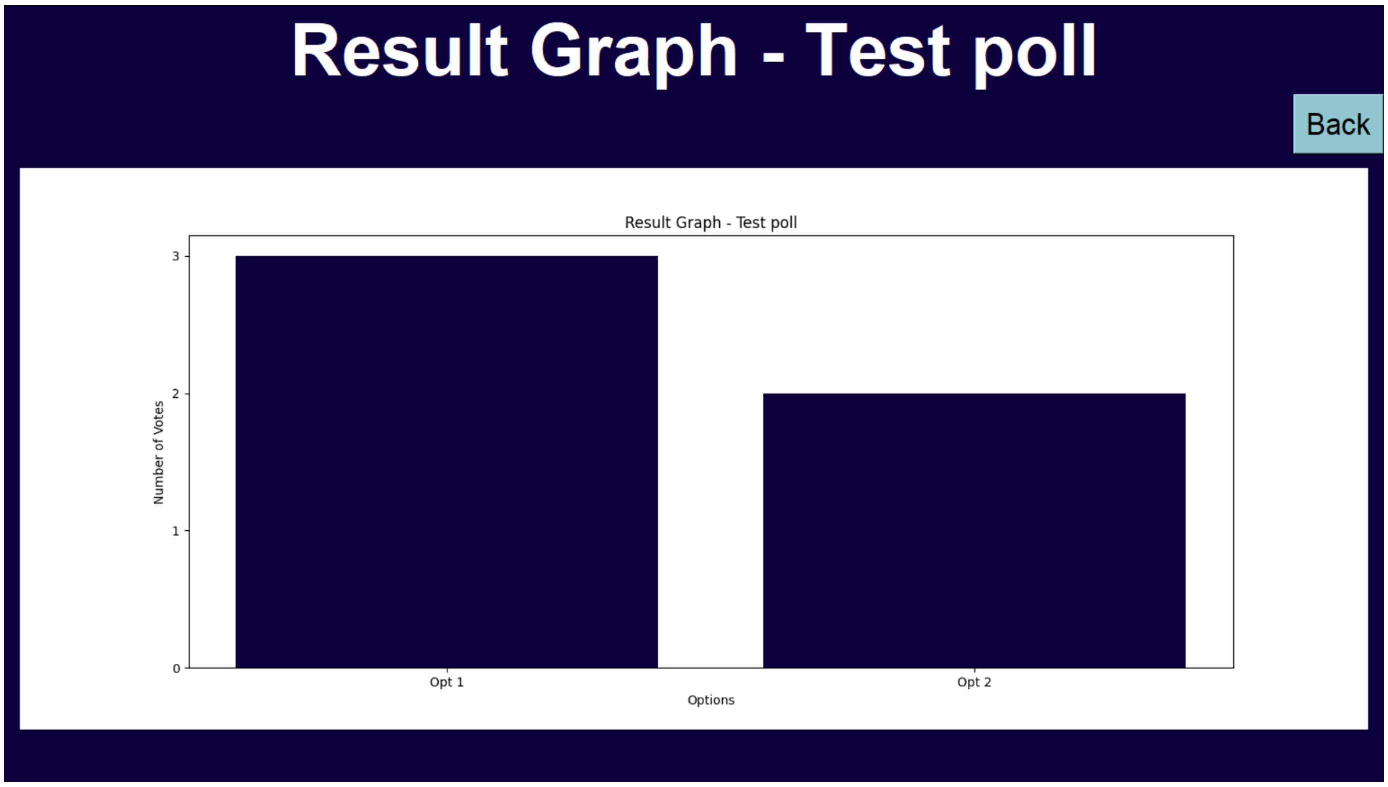
Result saved successfully (Test Poll)


Result - Test Poll

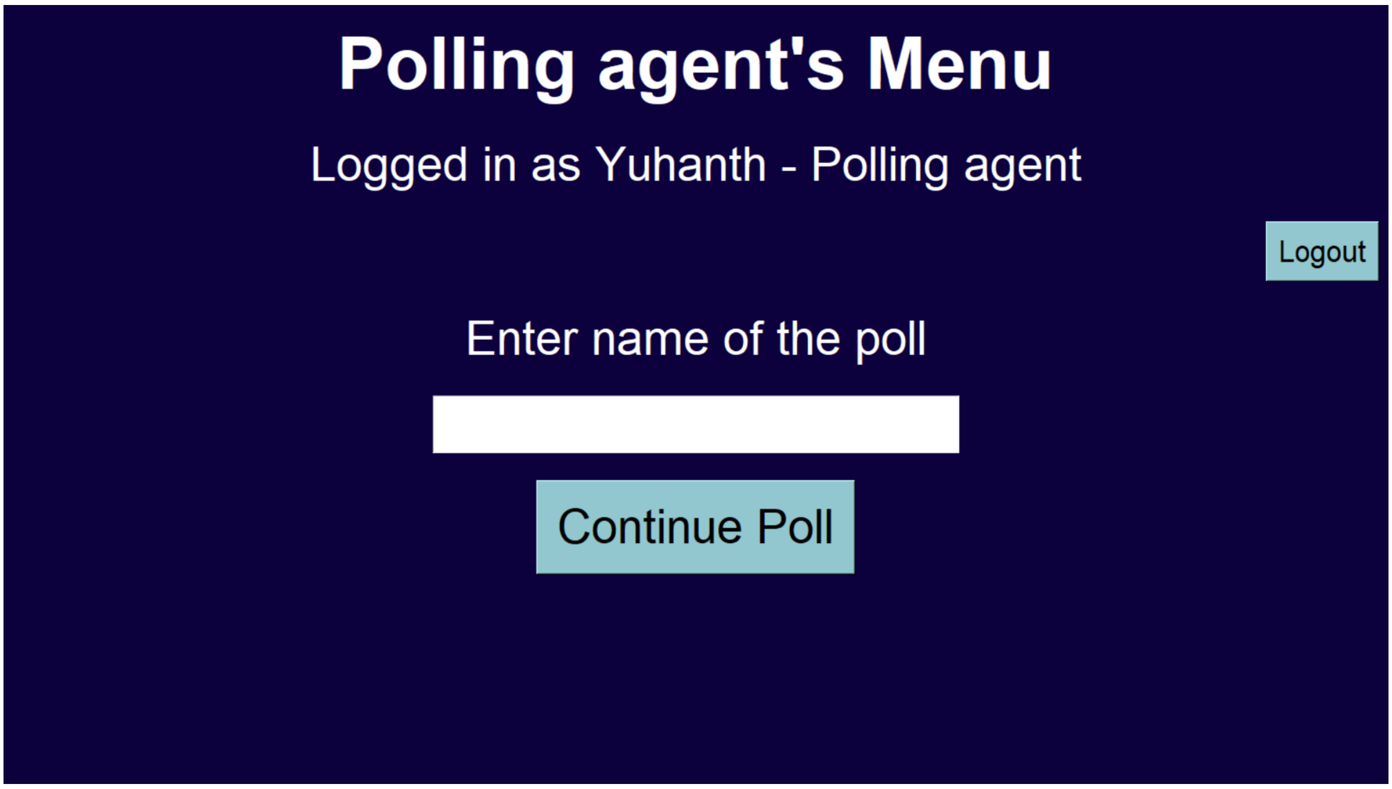
Back

Opt 1-3
Opt 2-2

Show Graph


Result Graph - Test poll

Back


Polling agent's Menu
Logged in as Yuhanth - Polling agent
Logout
Enter name of the poll
Continue Poll

Bibliography

- docs.python.org/3/library/tkinter.html
- pypi.org/project/cryptography/
- matplotlib.org
- docs.python.org/3/library/datetime.html
- www.geeksforgeeks.org