Rīgas Tālmācības Vidusskola

Dokumentācija

Piederumu piemērošanas lietotne

Darbu pilda:

Filips Emanuels Vaicis

Latvija, Rīga

2024

**Problēmas izpēte un analīze**

**Problēmas apraksts**

Aksesuāru kompānijai “Adidas” pašlaik nav aplikācija, kas ļautu potenciāliem pircējiem uzmērīt dažādus aksesuārus attālināti. Katrai interesējošai pusei jādodas uz vistuvāko “Adidas” veikalu un jāpielaiko preces uz vietas. “Adidas” plāno izlaist jaunu aplikāciju, kas radīs potenciāli lielāku peļņu.

**Izpētes metodes izvēle un pamatojums**

1. Lietotāja aptaujas: Tika veikta aptauja, kurā tika jautāts, cik noderīga būtu šāda programma, un kādas funkcijas viņi vēlētos redzēt programmā.
2. Konkurentu pārskats: Tika veikts pārskats par citu uzņēmumu priekšrocībām un trūkumiem, kā arī salīdzinājums ar Adidas pakalpojumiem.
3. Ekspertu viedokļi: Sadarbošanās ar mārketinga ekspertiem, lai iegūtu vērtīgas atsauksmes par esošajām problēmām un nepieciešamajām funkcijām.
4. Datubāzes analīze: Tika veikta esošo datu analīze par dažādām attēlu rediģēšanas programmām un to izmantošanas paradumiem.

**Izpētes procesa apraksts**

1. Problēmas identificēšana: Tika konstatēta problēma- potenciāliem pircējiem fiziski jādodas uz veikaliem, lai pielaikotu kompānijas piederumus.
2. Izpētes metožu izvēle: Tika izvēlētas metodes, kas piemērotas problēmas risināšanai un konstatēšanai.
3. Datu apkopošana un analīze: Tika apkopoti un analizēti dati, kas iegūti no ekspertu viedokļiem, lietotāju aptaujām, konkurentu izpētes un datubāzes analīzes.
4. Problēmas risinājuma identifikācija: Balstoties uz apkopoto informāciju, tika izvērtēti un konstatēti iespējamie risinājumi, kā arī nepieciešamās funkcijas un prasības.

**Izpētes datu apkopojums**

1. Lietotāja aptaujas: Aptaujā tika iegūti dati par nepieciešamajām funckijām un programmas nepieciešamību.
2. Konkurentu pārskats: Šajā posmā tika identificēti konkurentu trūkumi, kā arī priekšrocības. Tika veikts pārskats kā Adidas salīdzinās ar konkurentiem tehnoloģiju jomā.
3. Ekspertu viedokļi: Eksperta viedokļi sniedza noderīgu ieskatu esošajās problēmās, kā arī iespējamos risinājumus, pamatojoties uz viņu pieredzi un zināšanām mārketingā.
4. Datubāzes analīze: Datubāzes analīzes rezultāti sniedza informāciju par ērtu aplikācijas dizainu un lietotājam ērtu aplikācijas lietošanu.

Šie izpētes procesi ļāva rūpīgi izpētīt esošās problēmas un efektivākos risinājumus, kas atbilst lietotāju vajadzībām un prasībām.

**Programmatūras prasību specifikācija**

**Mērķauditorija**

1. Potenciālie “Adidas” produktu pircēji, kas vēlas pielāgot aksesuārus attālināti.
2. Lietotāji, kuri novērtē ērtu un modernu pieeju aksesuāru pielāgošanai.
3. Modeļi un modeļu kompānijas, kuriem interesē jaunas tehnoloģijas mode.

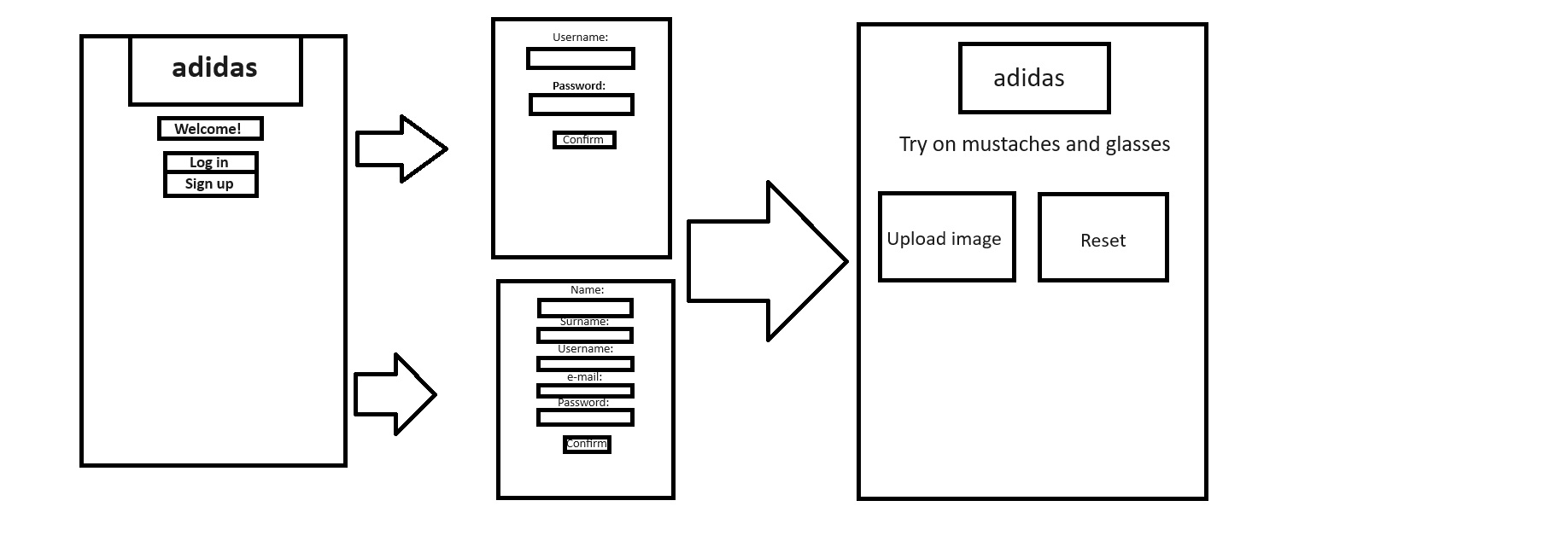
**Risinājuma raksturojums**

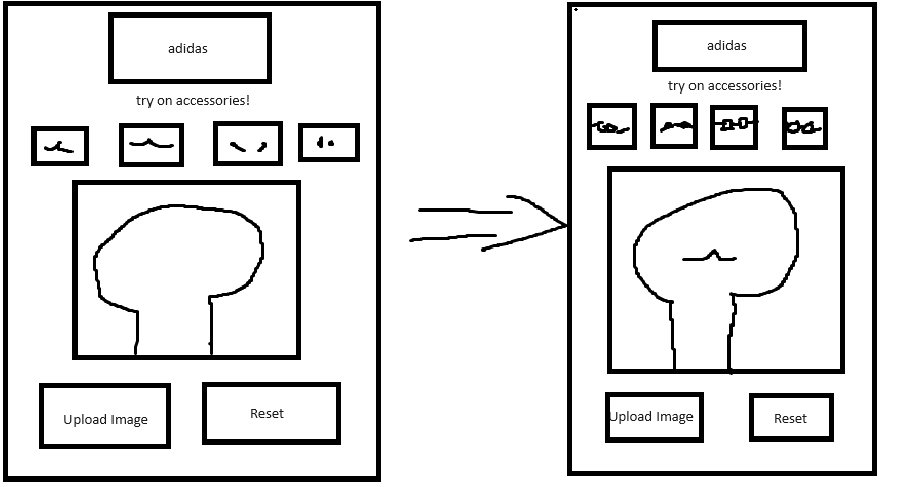
1. Aplikācija nodrošina iespēju potenciāliem pircējiem un interesentiem attālināti uzmērīt un pielāgot aksesuārus.
2. Lietotājiem tiks piedāvāta reģistrācija, lai izveidotu dziļāku saikni ar Adidas uzņēmumu.
3. Aplikācija piedāvās ērtu un draudzīgu interfeisu, kas nodrošinās, ka ikviens cilvēks var patīkami lietot aplikāciju.

**Programmatūras produkta un tā funkciju apraksts:**

1. Lietotāja reģistrācija un autorizācija: Lietotāji varēs reģistrēties un autorizēties, lai piekļūtu pie programmas funkcijām.
2. Attālināta mērīšana: Aplikācija nodrošinās iespēju attālināti izmērīt dažādus Adidas aksesuārus.
3. Personalizēta pielāgošana: Lietotne izmantos Python bibliotēkas -OpenCV un Tkinter, lai lietotāji varētu redzēt, kā viņu izvēlētie produkti uz tiem izskatās, augšupielādējot seju no frontālā leņķa.
4. Viegla saskarne: Lietotājiem pēc savu aksesuāru izvēles, tiks dota iespēja noņemt visus aksesuārus un sākt pielāgot piederumus no sākuma. Lietotāji jeb kurā laikā var augšupielādēt citu bildi, uz kuras pielāgot piederumus.

**Programmatūras produkta skice:**





**Programmatūras izstrādes plans**

Lai piemērotu efektīvāko izstrādes modeli, salīdzināšu Agile modeli ar citiem populāriem modeļiem, kā arī pamatošu izvēli par Agile moduļa izmantošanu.

Agile modelis:

* Elastība: Agile modelis piedāvā iespēju elastībai un pielāgoties vajadzībām un prasībām, kas ir būtiski, ja “Adidas” vēlas izlaist jaunu funckiju.
* Iteratīvs: Agile modelis ietver iterācijas, kurās katrā ciklā tiek izstrādāts un testēts neliels produkta gabals, kas dod iespēju novērtēt un reaģēt uz jaunām vajadzībām un prasībām.
* Klienta iesaiste: Agile modelis atbalsta aktīvu klienta iesaistīšanos un komunikāciju, kas ir svarīgi, lai nodrošinātu, ka izstrādātais produkts pilnībā atbilst klienta vēlmēm un prasībām.

Salīdzinājums ar citiem modeļiem:

1. Ūdenskrituma modelis: Salīdzinot agile modeli ar ūdenskrituma modeli, kurš ir lineārs un pieprasa visu prasību izpildi pirms nākamā posma uzsākšanas, agile modelis ir elastīgāks un ļauj strādāt iteratīvi, kas ir noderīgi, ja “Adidas” prasības mainās vai tiek precizētas projekta gaitā.
2. Spirālveida modelis: Salīdzinot agile modeli ar spirālveida modeli, kurš ir risku orientēts un ietver iterācijas, agile modelis ir līdzīgs, bet var būt mazāk strukturēts, kas atbilst “Adidas” projektam, kas varētu prasīt lielāku elastību.

Esmu izvēlējies agile modeli, kas nodrošinās “Adidas” projektam elastību, kas nepieciešama, lai veiksmīgi izstrādātu un uzturētu aplikāciju. Šis modelis ļaus ātri reaģēt uz jaunām prasībām, nodrošinot kvalitatīvu klienta iesaisti izstrādes procesā. Agile modelis dos “Adidas” aplikācijai iespēju regulāri piegādāt jaunas funkcijas un veikt nepieciešamos pielāgojumus.

**Programmatūras izstrādes plāns**

1. Prasību izpēte:

* Identificēt un izpētīt potenciālos lietotājus, kā arī viņu prasības un vēlmes.
* Saprast sākotnējās funkcijas un prasības.

1. Projektēšana:

* Projektu atdala detalizētos uzdevumos.
* Izveidots budžets un projekta pārvaldības plāns.

1. Dizaina izstrāde:

* Sadarbojoties ar klientu, tiek iegūts programmas noformējums – kŗāsas, foni, rakstzīmju fonts un izmērs, utt..
* Izveidojam saskarnes karkasa diagrammu.

1. Programmēšana:

* Izstrādāju funckijas iterācijas, kas tiek pievienotas produktam, pamatojoties uz klienta prioritātēm.
* Sadarbojoties ar klientu, tiek veiktas izmaiņas pēc nepieciešamības.

1. Testēšana:

* Veikt funkciju testēšanu, lai identificētu un novērstu kļūdas.
* Izstrādāta programmatūras beta versija.

1. Dokumentēšana:

* Tiek izveidota lietotāja rokasgrāmata, lai nodrošinātu sapratni par produktu.

1. Uzturēšana:

* Pēc produkta izlaišanas tiks nodrošināta programmas uzturēšana un atblasts, veicot kļūdu labošanu un nodrošinot papildu funkcionalitāti.

**Atkļūdošanas un akcepttestēšanas pārskats**

**Programmatūras vienību izstrāde un vienībtestēšana**

* Katru izstrādāto programmatūras vienību testē.
* Atklātās kļūdas tiek labotas, pirms programmatūras vienības tiek apvienotas ar citām sistēmas daļām.

**Programmatūras vienību apvienošana un integrācijas testēšana**

* Izstrādātās programmatūras vienības apvieno un pārbauda to mijiedarbību, lai nodrošinātu pareizu darbību visā aplikācijā.
* Veic visas aplikācijas testu, lai pārliecinātos, ka jaunās programmatūras vienības nav salauzušas citas sistēmas daļas.

**Akcepttestēšana**

* Veic akcepttestēšanu, lai klients būtu apmierināts ar galīgo produktu, kā arī lai pārliecinātos, ka tas atbilst klienta prasībām un programmatūras prasību specifikācijai.
* Tiek pārbaudīti, vai visi izstrādātie risinājumi tiek veiksmīgi un pareizi izmantoti, un atbilst klienta prasībām.

**Lietotāja ceļvedis**

**Izvēršanas plans**

* Noteikt datuma plānošanu katram posmam, iekļaujot testēšanas un atsauksmju ievākšanas posmus.
* Izstrādāt komunikācijas plānu, kas informēs lietotājus par jauno programmatūru un tās funkcijām.

**Lietotāja ceļvedis**

* Satur informāciju par katru funkciju un to, kā to izmantot.
* Nodrošina apmācību par programmatūras izmantošanu, kas palīdzēs lietotājiem efektīvi izmantot programmatūru.
* Iekļauj visbiežāk uzdotos jautājumus, lai sniegtu ātru atbildi uz tiem.

**Uzturēšanas plāns**

* Noteikt atbalsta pakalpojumus, tostarp komunikācijas platformas un atbildes laiku.
* Nodrošināt regulāru programmatūras atjauninājumu, ievērojot lietotāju atsauksmes.
* Uzklausīt un lasīt gan lietotāju, gan kritiķu viedokļus un atsauksmes.

**Piemērotās licences pamatojums**

Ņemot vērā “Adidas” aplikācijas mērķi un uzdevumu, piemērotākā licence šaja gadījumā ir brīvas programmatūras licence. Salīdzinot ar citām licencēm, piemēram, komerciālās licences ierobežo programmatūras izplatīšanu, izmantošanu un modificēšanu, jo prasa samaksu par to. GPL un Apache licences atļauj programmatūras modificēšanu bez maksas, nodrošinot brīvību un elastīgumu. Proprietārās licences dod autoriem pilnīgu kontroli pār to, kā programmatūra tiek izmantota un modificēta. Brīvas programmatūras licence ļaus “Adidas” izstrādāt un izplatīt aplikāciju, iekļaujot brīvību lietotājiem, kas vēlas izmantot un pielāgot aplikāciju savām vajadzībām un prasībām. Šāda veida licence ļaus lietotājiem uzlabot aplikācijas funkcionalitāti, kas ir īpaši svarīgi, ņemot vērā aplikācijas potenciālo lietderību un pielāgojamību dažādiem klientiem un prasībām. Brīvās programmatūras licences bieži veicina kopienas iesaistīšanos un atbalstu. “Adidas” varētu iegūt labumu no šādas aktīvas kopienas, kas piedāvā palīdzību un padomus, uzlabojot aplikāciju, kā arī nodrošinot pastāvīgu atbalstu.

Brīvās programmatūras licences ļauj pielāgot aplikāciju pēc vajadzības, padarot to piemērotāku dažādām klientu grupām, kas vēlas veikt specifiskus uzdevumus. Tas nodrošina vairāk lietotājus, kā arī lielāku nozīmi aplikācijai.

Izvēloties brīvās programmatūras licenci demonstrē uzticību un paredzamību. “Adidas” var gūt vairāk iesaistītus klientus, kas respektēs kompāniju. Aplikācija būs daudz pārbaudāmāka, kas samazinās risku, ka problēmas paliks nepamanītas vai neatrisinātas.

Brīvās programmatūras licences izvēle var palielināt aplikācijas mērķauditoriju. Klienti, kas meklē brīvu un atvērtu risinājumu savām vajadzībām būs ieinteresēti šajā lietotnē nevis kādā slēgtā pirmkoda aplikācijā.

**Programmatūras kods**

Veidojot “Adidas” programmatūru tika ieverotas svarīgas programmēšanas prakses:

* Koda skaidrība: Kods tikts strukturēts un organizēts tā, lai būtu viegli saprotams lasītājiem.
* Koda uzturamība: Nepieciešamības gadījumā, kods ir viegli labojams un uzturams, samazinot nepieciešamību ilgām atkļūdošanas sesijām.
* Dokumentācijas saglabāšana: Programmatūras kods tiek pievienots dokumentācijas pielikumā, lai citi varētu izprast kodu.
* Koda komentēšana: Komentāri tiek ievietoti kodā, lai dotu papildus skaidrību par dažādām funkcijām un darbībām, kas varētu palīdzēt izprast kodu un veikt nepieciešamos labojumus vai paplašinājumus.

**Pielikums**

menu.py

import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageTk, ImageDraw

import numpy as np

import cv2

import imutils

import math

from databasefiles import dbhandling

print("Main program start")

# check if user has logged in

try:

    with open("databasefiles/user.txt", "r") as user\_f:

        user = user\_f.read()

        if user == "":

            print("Terminated; no user found")

            exit()

except:

    print("Terminated; unable to read user.txt")

    exit()

else:

    with open("databasefiles/user.txt", "w") as user\_f:

        user\_f.close()

    print("Logged in as", user)

eye\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade\_eye.xml")

smile\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade\_mouth.xml")

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade\_frontalface\_alt.xml")

app = tk.Tk()

# Load your background image (PIL)

bg\_image = Image.open(

    "assets/background.jpg"

)  # Change to your background image path (from PIL to Tkinter)

bg\_photo = ImageTk.PhotoImage(bg\_image)

#Creates a label for the image

bg\_label = tk.Label(app, image=bg\_photo)

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

resize\_delay = 100

resize\_timer = None

# Function for resizing the background

def resize\_background():

    global bg\_photo

    app\_width = app.winfo\_width()

    app\_height = app.winfo\_height()

    resized\_bg = bg\_image.resize((app\_width, app\_height), Image.LANCZOS)

    bg\_photo = ImageTk.PhotoImage(resized\_bg)

    bg\_label.config(image=bg\_photo)

#Prevents lagging when resizing the window

def delayed\_resize\_background(event):

    global resize\_timer

    if resize\_timer:

        app.after\_cancel(resize\_timer)

    resize\_timer = app.after(resize\_delay, resize\_background)

app.bind("<Configure>", delayed\_resize\_background) #Whenever window is resized, it calls in the function

resize\_background()  # Initially set the background image

app.minsize(600, 800)

app.title("Abibas")

app.geometry("800x850")

app.configure(bg="#f5f5f5")

button\_font = ("Helvetica", 20, "bold")

# Load the "Abibas.png" image

abibas\_image = Image.open(

    "assets/Abibas.png"

)

desired\_width = 200

desired\_height = 100

# Calculate new dimensions while preserving the aspect ratio

width, height = abibas\_image.size #Original size

aspect\_ratio = width / height #Aspect ratio calculation

if width > height:

    new\_width = desired\_width

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

else:

    new\_height = desired\_height

    new\_width = int(desired\_height \* aspect\_ratio)

# Resize the image to the calculated dimensions

abibas\_image = abibas\_image.resize((new\_width, new\_height), Image.LANCZOS)

# Convert the resized image to a PhotoImage (Tkinter format)

abibas\_photo = ImageTk.PhotoImage(abibas\_image)

# Create a label to display the resized Abibas image

abibas\_label = tk.Label(app, image=abibas\_photo, bg="#A0FF91")

abibas\_label.grid(row=0, column=0, columnspan=3, pady=(20, 10), sticky="n")

subtitle\_label = tk.Label(

    app,

    text="Try on Accessories!",

    font=("Circular Std Book", 30),

    bg="#CCFFCC",

    fg="#333",

)

subtitle\_label.grid(row=1, column=0, columnspan=3, sticky="n")

# Prevent the labels from disappearing when minimizing

app.grid\_rowconfigure(0, weight=0) #wont expand or resize

app.grid\_rowconfigure(1, weight=0) #wont expand or resize

app.grid\_columnconfigure(0, weight=1) #will expand or resize

app.grid\_columnconfigure(1, weight=1) #will expand or resize

app.grid\_columnconfigure(2, weight=1) #will expand or resize

frame = None

gray = None

faces = None

def chooseFile():

    global frame, gray, faces, original\_frame, file\_path

    file\_path = filedialog.askopenfilename(

        filetypes=[("Image files", "\*.png;\*.jpg;\*.jpeg;\*.gif")]

    )

    if file\_path:

        frame = cv2.imread(file\_path, 1) #Reads in color mode

        frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2BGRA) #Converts to Blue Green Red Alpha format

        gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) #Converts image to grayscale

        faces = face\_cascade.detectMultiScale(gray, 1.3, 10)

        original\_frame = frame.copy()

        display\_image(file\_path)

def display\_image(file\_path):

    max\_width = 400

    max\_height = 400

    image = Image.open(file\_path)

    # Calculate the aspect ratio to maintain proportions

    width, height = image.size

    new\_width = width

    new\_height = height

    if width > max\_width or height > max\_height:

        #The image is larger than the maximum size so we need to resize it proportionally

        ratio = min(max\_width / width, max\_height / height)

        new\_width = int(width \* ratio)

        new\_height = int(height \* ratio)

        image = image.resize((new\_width, new\_height), Image.LANCZOS)

    # Rounded corners

    mask = Image.new("L", (new\_width, new\_height), 0)

    draw = ImageDraw.Draw(mask)

    rounded\_radius = 65

    draw.ellipse(

        (

            -rounded\_radius,

            -rounded\_radius,

            new\_width + rounded\_radius,

            new\_height + rounded\_radius,

        ),

        fill=255,

    )

    image.putalpha(mask)

    photo = ImageTk.PhotoImage(image)

    image\_label.config(image=photo)

    image\_label.image = photo

    put\_mustache1\_button.grid(row=6, column=0, padx=10, sticky="nsew")

    put\_mustache2\_button.grid(row=6, column=1, padx=10, sticky="nsew")

    put\_mustache3\_button.grid(row=6, column=2, padx=10, sticky="nsew")

    put\_mustache4\_button.grid(row=6, column=3, padx=10, sticky="nsew")

    app.update\_idletasks()# Update the window

    # Create a label for the main image display

    image\_label.grid(

        row=4, column=0, columnspan=3, pady=10, sticky="nsew"

    )

    #Hide buttons for now

    put\_glasses1\_button.grid\_forget()

    put\_glasses2\_button.grid\_forget()

    put\_glasses3\_button.grid\_forget()

    put\_glasses4\_button.grid\_forget()

    put\_glasses5\_button.grid\_forget()

def apply\_glasses(glasses\_path):

    global frame, gray, faces

    if frame is not None:

        for x, y, w, h in faces: #Get coordinates of bounding boxes

            roi\_gray = gray[y : y + h, x : x + w]#Region of interest in gray

            roi\_color = frame[y : y + h, x : x + w]# Region of interest in color

            eyes = eye\_cascade.detectMultiScale(roi\_gray, 1.3, 10) # (where, scaleFactor, minNighbors)

            for x, y, w, h in eyes:

                glasses = cv2.imread(glasses\_path, cv2.IMREAD\_UNCHANGED) #Loading image with alpha channel

                eye1 = (eyes[0][0], eyes[0][1]) #Coordinate of top left first eye

                eye2 = (eyes[1][0] + eyes[1][3], eyes[1][1])# Coordinate of top right second eye

                eyesWidth = math.dist(eye1, eye2)

                eyesHeight = (int(eyes[0][1]) + int(eyes[1][1])) / 2 #Average height of two detected eyes

                eyesMidPointX = int(eyesWidth) / 2

                glassesHeight = float((eyes[0][3]+eyes[1][3])/2)

                glasses = cv2.resize(

                    glasses, (int(eyesWidth) + 10, int(glassesHeight) + 5)

                )

                gh, gw, gc= glasses.shape

                PlacementX = int(eyesMidPointX) - (gh // 2) #Since the glasses will be displayed with top left coordinates, I will make it so its somewhere around the left eye

                #Making transparent parts of the image show up as transparent

                for i in range(0, gh):

                    for j in range(0, gw): #Iterates over every pixel of glasses

                        if glasses[i, j][3] != 0: #If the pixel is not transparent then add it to the image (a value of 0 in the alpha channel would mean its transparent)

                            roi\_color[int(eyesHeight) + i, int(PlacementX) + j] = glasses[i, j] #Apply every pixel of glasses image to the roi

            # Convert from BGR to RGB so PIL can use it

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

            # Create an image object from the modified image

            modified\_image\_pil = Image.fromarray(modified\_image)

            max\_width = 400

            max\_height = 400

            # Initialize new\_width and new\_height

            new\_width = modified\_image\_pil.width

            new\_height = modified\_image\_pil.height

            # Calculate the aspect ratio to maintain proportions

            if new\_width > max\_width or new\_height > max\_height:

                ratio = min(max\_width / new\_width, max\_height / new\_height)

                new\_width = int(new\_width \* ratio)

                new\_height = int(new\_height \* ratio)

                modified\_image\_pil = modified\_image\_pil.resize(

                    (new\_width, new\_height), Image.LANCZOS

                )

            # Create a mask for the slightly rounded corners

            mask = Image.new("L", modified\_image\_pil.size, 0)

            draw = ImageDraw.Draw(mask)

            rounded\_radius = 65

            draw.ellipse(

                (

                    -rounded\_radius,

                    -rounded\_radius,

                    new\_width + rounded\_radius,

                    new\_height + rounded\_radius,

                ),

                fill=255,

            )

            modified\_image\_pil.putalpha(mask)

            modified\_image\_pil.thumbnail((800, 800))

            modified\_photo = ImageTk.PhotoImage(modified\_image\_pil)# Converts to a Tkinter photo

            #Display the image

            image\_label.config(image=modified\_photo)

            image\_label.image = modified\_photo

            # Hide glasses buttons

            put\_glasses1\_button.grid\_forget()

            put\_glasses2\_button.grid\_forget()

            put\_glasses3\_button.grid\_forget()

            put\_glasses4\_button.grid\_forget()

            put\_glasses5\_button.grid\_forget()

def apply\_accessories(accesories\_path):

    global frame, gray, faces

    if frame is not None:

        for x, y, w, h in faces:

            roi\_gray = gray[y : y + h, x : x + h]

            roi\_color = frame[y : y + h, x : x + h]

            mouth = smile\_cascade.detectMultiScale(roi\_gray, 1.3, 30)

            for mx, my, mw, mh in mouth:

                accesories\_img = cv2.imread(accesories\_path, cv2.IMREAD\_UNCHANGED)

                accessoriesHeight = mh / 2

                accessories\_resized = cv2.resize(accesories\_img, (mw, int(accessoriesHeight)))

                h, w, c = accessories\_resized.shape

                accessoriesX = mouth[0][0]

                accessoriesY = int(mouth[0][1]) - int(w / 9) #Make the mustache above lips

                for i in range(0, h):

                    for j in range(0, w):

                        if accessories\_resized[i, j][3] != 0:

                            roi\_color[

                                int(accessoriesY) + i, int(accessoriesX) + j

                            ] = accessories\_resized[i, j]

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

            modified\_image\_pil = Image.fromarray(modified\_image)

            max\_width = 400

            max\_height = 400

            new\_width = modified\_image\_pil.width

            new\_height = modified\_image\_pil.height

            if new\_width > max\_width or new\_height > max\_height:

                ratio = min(max\_width / new\_width, max\_height / new\_height)

                new\_width = int(new\_width \* ratio)

                new\_height = int(new\_height \* ratio)

                modified\_image\_pil = modified\_image\_pil.resize(

                    (new\_width, new\_height), Image.LANCZOS

                )

            #Create a mask for the slightly rounded corners

            mask = Image.new("L", modified\_image\_pil.size, 0)

            draw = ImageDraw.Draw(mask)

            rounded\_radius = 65

            draw.ellipse(

                (

                    -rounded\_radius,

                    -rounded\_radius,

                    new\_width + rounded\_radius,

                    new\_height + rounded\_radius,

                ),

                fill=255,

            )

            modified\_image\_pil.putalpha(mask)

            modified\_image\_pil.thumbnail((800, 800))

            modified\_photo = ImageTk.PhotoImage(modified\_image\_pil)

            image\_label.config(image=modified\_photo)

            image\_label.image = modified\_photo

            # Hide mustache buttons

            put\_mustache1\_button.grid\_forget()

            put\_mustache2\_button.grid\_forget()

            put\_mustache3\_button.grid\_forget()

            put\_mustache4\_button.grid\_forget()

            # Show glasses buttons

            put\_glasses1\_button.grid(row=6, column=0, padx=10)

            put\_glasses2\_button.grid(row=6, column=1, padx=10)

            put\_glasses3\_button.grid(row=6, column=2, padx=10)

            put\_glasses4\_button.grid(row=6, column=3, padx=10)

            put\_glasses5\_button.grid(row=6, column=4, padx=10)

def reset\_frame():

    global frame, gray, faces, file\_path

    if original\_frame is not None:

        frame = original\_frame.copy()

        gray = cv2.cvtColor(frame, cv2.COLOR\_BGRA2GRAY)

        faces = face\_cascade.detectMultiScale(gray, 1.3, 10)

        display\_image(file\_path)

button\_width = int(

    app.winfo\_screenwidth() \* 0.05

)

button\_height = int(

    app.winfo\_screenwidth() \* 0.03

)

#create a frame for the upload and reset buttons

upload\_reset\_frame = tk.Frame(app, bg="#121212")

upload\_reset\_frame.grid(row=5, column=0, columnspan=3, pady=(0, 10), sticky="n")

upload\_button = tk.Button(

    upload\_reset\_frame,

    text="Upload Image",

    font=button\_font,

    compound=tk.CENTER,

    bg="#007bff",

    fg="#121212",

    width=10,

    height=2,

    command=chooseFile,

    borderwidth=5,

    relief="ridge",

    padx=20,

    pady=10,

)

upload\_button.grid(row=0, column=0, padx=(10, 5))

reset\_button = tk.Button(

    upload\_reset\_frame,

    text="Reset",

    font=button\_font,

    bg="#dc3545",

    compound=tk.CENTER,

    fg="#121212",

    width=10,

    height=2,

    command=reset\_frame,

    borderwidth=5,

    relief="ridge",

    padx=20,

    pady=10,

)

reset\_button.grid(row=0, column=1, padx=(5, 10))

# Create a frame for the accessories buttons

button\_frame = tk.Frame(app, bg="#121212", width=button\_width, height=button\_height)

button\_frame.grid(row=3, column=0, columnspan=3, pady=5)

mustache\_image1 = Image.open("assets/mustachep.jpg")

mustache\_image1 = mustache\_image1.resize((button\_width, button\_height), Image.LANCZOS)

mustache\_photo1 = ImageTk.PhotoImage(mustache\_image1)

put\_mustache1\_button = tk.Button(

    button\_frame,

    image=mustache\_photo1,

    compound=tk.CENTER,

    bg="#121212",

    fg="#121212",

    width=button\_width,

    height=button\_height,

    command=lambda: apply\_accessories("assets/mustache.png"),

    relief="ridge",

    borderwidth=5,

)

mustache\_image2 = Image.open("assets/mustachep2.jpg")

mustache\_image2 = mustache\_image2.resize((button\_width, button\_height), Image.LANCZOS)

mustache\_photo2 = ImageTk.PhotoImage(mustache\_image2)

put\_mustache2\_button = tk.Button(

    button\_frame,

    image=mustache\_photo2,

    compound=tk.CENTER,

    bg="#121212",

    fg="#fff",

    width=button\_width,

    height=button\_height,

    command=lambda: apply\_accessories("assets/mustache2.png"),

    relief="ridge",

    borderwidth=5,

)

mustache\_image3 = Image.open("assets/piercing.jpg")

mustache\_image3 = mustache\_image3.resize((button\_width, button\_height), Image.LANCZOS)

mustache\_photo3 = ImageTk.PhotoImage(mustache\_image3)

put\_mustache3\_button = tk.Button(

    button\_frame,

    image=mustache\_photo3,

    compound=tk.CENTER,

    bg="#121212",

    fg="#fff",

    width=button\_width,

    height=button\_height,

    command=lambda: apply\_accessories("assets/piercing.png"),

    relief="ridge",

    borderwidth=5,

)

mustache\_image4 = Image.open("assets/piercing1.jpg")

mustache\_image4 = mustache\_image4.resize((button\_width, button\_height), Image.LANCZOS)

mustache\_photo4 = ImageTk.PhotoImage(mustache\_image4)

put\_mustache4\_button = tk.Button(

    button\_frame,

    image=mustache\_photo4,

    compound=tk.CENTER,

    bg="#121212",

    fg="#fff",

    width=button\_width,

    height=button\_height,

    command=lambda: apply\_accessories("assets/piercing1.png"),

    relief="ridge",

    borderwidth=5,

)

glasses\_image1 = Image.open("assets/glassesp1.jpg")

glasses\_image1 = glasses\_image1.resize((button\_width, button\_height), Image.LANCZOS)

glasses\_photo1 = ImageTk.PhotoImage(glasses\_image1)

put\_glasses1\_button = tk.Button(

    button\_frame,

    image=glasses\_photo1,

    compound=tk.CENTER,

    bg="#121212",

    fg="#fff",

    width=button\_width,

    height=button\_height,

    command=lambda: apply\_glasses("assets/glasses1.png"),

    relief="ridge",

    borderwidth=5,

)

glasses\_image2 = Image.open("assets/glassesp2.jpg")

glasses\_image2 = glasses\_image2.resize((button\_width, button\_height), Image.LANCZOS)

glasses\_photo2 = ImageTk.PhotoImage(glasses\_image2)

put\_glasses2\_button = tk.Button(

    button\_frame,

    image=glasses\_photo2,

    compound=tk.CENTER,

    bg="#121212",

    fg="#fff",

    width=button\_width,

    height=button\_height,

    command=lambda: apply\_glasses("assets/glasses2.png"),

    relief="ridge",

    borderwidth=5,

)

glasses\_image3 = Image.open("assets/glassesp3.jpg")

glasses\_image3 = glasses\_image3.resize((button\_width, button\_height), Image.LANCZOS)

glasses\_photo3 = ImageTk.PhotoImage(glasses\_image3)

put\_glasses3\_button = tk.Button(

    button\_frame,

    image=glasses\_photo3,

    compound=tk.CENTER,

    bg="#121212",

    fg="#fff",

    width=button\_width,

    height=button\_height,

    command=lambda: apply\_glasses("assets/glasses3.png"),

    relief="ridge",

    borderwidth=5,

)

glasses\_image4 = Image.open("assets/glassesp4.jpg")

glasses\_image4 = glasses\_image4.resize((button\_width, button\_height), Image.LANCZOS)

glasses\_photo4 = ImageTk.PhotoImage(glasses\_image4)

put\_glasses4\_button = tk.Button(

    button\_frame,

    image=glasses\_photo4,

    compound=tk.CENTER,

    bg="#121212",

    fg="#fff",

    width=button\_width,

    height=button\_height,

    command=lambda: apply\_glasses("assets/glasses4.png"),

    relief="ridge",

    borderwidth=5,

)

glasses\_image5 = Image.open("assets/glassesp5.jpg")

glasses\_image5 = glasses\_image5.resize((button\_width, button\_height), Image.LANCZOS)

glasses\_photo5 = ImageTk.PhotoImage(glasses\_image5)

put\_glasses5\_button = tk.Button(

    button\_frame,

    image=glasses\_photo5,

    compound=tk.CENTER,

    bg="#121212",

    fg="#fff",

    width=button\_width,

    height=button\_height,

    command=lambda: apply\_glasses("assets/glasses5.png"),

    relief="ridge",

    borderwidth=5,

)

# label for the main image display

image\_label = tk.Label(app)

image\_label.configure(bg="#121212")

app.mainloop()

dbhandling.py

# Importējam SQLite

import sqlite3 as db

import datetime

import bcrypt

from tkinter import \*

from tkinter import ttk

from PIL import Image, ImageTk

global conn

conn = db.connect(r"C:\Users\micro\Desktop\opencv\opencv\databasefiles\accounts.db")

def db\_setup():

    sql = """ CREATE TABLE "users" (

    "user\_id"   INTEGER NOT NULL UNIQUE,

    "name"  TEXT NOT NULL CHECK ("name" <> ''),

    "surname"   TEXT NOT NULL CHECK ("surname" <> ''),

    "username"  TEXT NOT NULL UNIQUE CHECK ("username" <> ''),

    "email" TEXT NOT NULL UNIQUE CHECK ("email" <> ''),

    "pwd\_hash"  TEXT NOT NULL,

    "reg\_date"  TEXT NOT NULL,

    PRIMARY KEY("user\_id")

    );

    """

    return conn.execute(sql)

def submit\_signup(name, surname, username, email, password):

    if not password.get():

        print("=== blank password ===")

    ## password hashing

    # converting password to array of bytes

    bytes = password.get().encode("utf-8")

    # Adding the salt to password

    salt = bcrypt.gensalt()

    # Hashing the password

    hashed = bcrypt.hashpw(bytes, salt)

    reg\_date = datetime.datetime.now()

    sqlite\_insert\_blob\_query = f""" INSERT INTO users

                                  (name, surname, username, email, pwd\_hash, reg\_date) VALUES (?, ?, ?, ?, ?, ?)"""

    # Convert data into tuple format

    data\_tuple = (

        name.get(),

        surname.get(),

        username.get(),

        email.get(),

        hashed,

        reg\_date,

    )

    cur = conn.cursor()

    try:

        cur.execute(sqlite\_insert\_blob\_query, data\_tuple)

    except:

        print("=== Invalid registration data submitted ===")

    else:

        with open(r"C:\Users\micro\Desktop\opencv\opencv\databasefiles\user.txt", "w") as user\_f:

            user\_f.write(username.get())

            print("User:", username.get())

        cur.close()

        conn.commit()

        root.destroy()

    return

def signup():

    root.withdraw()

    signup\_window = Toplevel(root)

    signup\_window.title("Abibas")

    signup\_window.geometry("600x800")

    signup\_window.configure(background="#00001c")

    ttk.Label(

        signup\_window,

        text="Name:",

        anchor="center",

        font=("Helvetica", 16),

        background="#CCFFCC",

    ).pack()

    name = ttk.Entry(signup\_window, width=20, style="TEntry", font=("Helvetica", 12))

    name.anchor("center")

    name.pack()

    ttk.Label(signup\_window, text="", background="#00001c").pack()

    ttk.Label(

        signup\_window,

        text="Surname:",

        anchor="center",

        font=("Helvetica", 16),

        background="#CCFFCC",

    ).pack()

    surname = ttk.Entry(signup\_window, width=20, style="TEntry", font=("Helvetica", 12))

    surname.anchor("center")

    surname.pack()

    ttk.Label(signup\_window, text="", background="#00001c").pack()

    ttk.Label(

        signup\_window,

        text="Username:",

        anchor="center",

        font=("Helvetica", 16),

        background="#CCFFCC",

    ).pack()

    username = ttk.Entry(

        signup\_window, width=20, style="TEntry", font=("Helvetica", 12)

    )

    username.anchor("center")

    username.pack()

    ttk.Label(signup\_window, text="", background="#00001c").pack()

    ttk.Label(

        signup\_window,

        text="e-mail:",

        anchor="center",

        font=("Helvetica", 16),

        background="#CCFFCC",

    ).pack()

    email = ttk.Entry(signup\_window, width=20, style="TEntry", font=("Helvetica", 12))

    email.anchor("center")

    email.pack()

    ttk.Label(signup\_window, text="", background="#00001c").pack()

    ttk.Label(

        signup\_window,

        text="Password:",

        anchor="center",

        font=("Helvetica", 16),

        background="#CCFFCC",

    ).pack()

    password = ttk.Entry(

        signup\_window, width=20, style="TEntry", font=("Helvetica", 12), show="\*"

    )

    password.anchor("center")

    password.pack()

    ttk.Label(signup\_window, text="", background="#00001c").pack()

    confirm\_button = ttk.Button(

        signup\_window,

        text="Confirm",

        command=lambda: submit\_signup(name, surname, username, email, password),

        style="W.TButton",

    ).pack()

    return

def submit\_login(password, username, window):

    if not password.get():

        return print("=== blank password ===")

    userBytes = password.get().encode("utf-8")

    sql = f"""SELECT pwd\_hash FROM "users"

    WHERE "username" = "{username.get()}"; """

    cur = conn.execute(sql)

    dati = cur.fetchone()

    result = bcrypt.checkpw(userBytes, dati[0])

    conn.commit()

    if result == True:

        print("=== Authorized ===")

        with open(r"C:\Users\micro\Desktop\opencv\opencv\databasefiles\user.txt", "w") as user\_f:

            user\_f.write(username.get())

            print("User", username.get())

        root.destroy()

    else:

        print("=== Invalid login ===")

    return

def login():

    login\_window = Toplevel(root)

    login\_window.title("Abibas")

    login\_window.geometry("600x800")

    login\_window.configure(background="#00001c")

    root.withdraw()

    ttk.Label(

        login\_window,

        text="Username:",

        anchor="center",

        font=("Helvetica", 16),

        background="#CCFFCC",

    ).pack()

    username = ttk.Entry(login\_window, width=20, style="TEntry", font=("Helvetica", 12))

    username.anchor("center")

    username.pack()

    ttk.Label(login\_window, text="", background="#00001c").pack()

    ttk.Label(

        login\_window,

        text="Password:",

        anchor="center",

        font=("Helvetica", 16),

        background="#CCFFCC",

    ).pack()

    password = ttk.Entry(

        login\_window, width=20, style="TEntry", font=("Helvetica", 12), show="\*"

    )

    password.anchor("center")

    password.pack()

    ttk.Label(login\_window, text="", background="#00001c").pack()

    ttk.Button(

        login\_window,

        text="Confirm",

        command=lambda: submit\_login(password, username, login\_window),

        style="W.TButton",

    ).pack()

    return

# db\_setup(conn)

### Visuals

root = Tk()

root.title("Abibas")

root.geometry("600x800")

root.configure(background="#00001c")

image = Image.open(r"C:\Users\micro\Desktop\opencv\opencv\assets/abibas.png")

resize\_image = image.resize((200, 120))

abibas\_image = ImageTk.PhotoImage(resize\_image)

# create label and add resize image

abibas\_display = Label(image=abibas\_image, background="#A0FF91")

abibas\_display.image = abibas\_image

abibas\_display.pack()

entry\_style = ttk.Style()

entry\_style.theme\_use("alt")

entry\_style.configure("TEntry", fieldbackground="#CCFFCC")

btn\_style = ttk.Style()

btn\_style.theme\_use("alt")

btn\_style.configure("W.TButton", font=("Helvetica", 16), background="#007bff")

ttk.Label(root, text="", background="#00001c").pack()

subtitle\_label = ttk.Label(

    root, text="Welcome!", font=("Great Vibes", 30), background="#A0FF91"

)

subtitle\_label.pack()

ttk.Label(root, text="", background="#00001c").pack()

login\_btn = ttk.Button(root, text="Log In", command=login, style="W.TButton")

login\_btn.pack()

signup\_btn = ttk.Button(root, text="Sign Up", command=signup, style="W.TButton")

signup\_btn.pack()

conn.commit()

root.mainloop()

conn.close()