**Higher Level Overview of the application:**

This is a machine learning application for performing three stage biometric registration and verification , in simple words the functionality works in the following ways:

1. Register the User , by capturing their face , iris and lip features and ask them to input their name
2. After the user gives the data , the facial , lips and iris features along with their name will be stored in the sqlite databases.
3. Then if the User wants to verify if his face is stored , then the live captured face, iris and lip features of the image is compared to that of the stored image.
4. If they both are the same then the name of the User is shown , else it shows “Unknown User”.

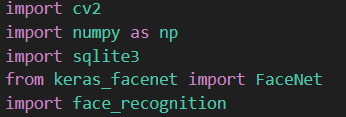
**Detailed Code Explanation of the Multibiometric verification:**

**Introduction:**

Here the objective is to make a multibiometric verification system that captures the users (face, iris , lip features) along with their name during registering and then verifies the stored user data with the live captured features of the user using facenet model for face recognition and face recognition library for iris and lip features detection and verification.

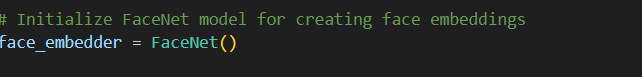
1. Face recognition code module using Facenet

**1.1.Libraries :**



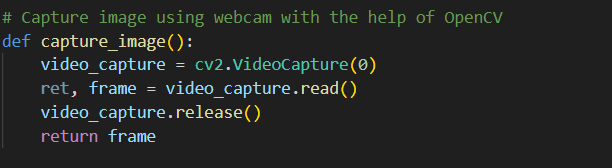
1. OpenCV(cv2) it is mainly imported to capture image through the webcam and process them . It is a open-source machine learning library.
2. Numpy ->This library is used to handle different numerical operations and manipulations of the face encodings further in the code.
3. Sqlite3 -> This is the database that is being used for storing the captured faces of the users along with their names.
4. face\_recognition This is a machine learning library that contains pre-trained deep learning models by using dlib under the hood , and helps in face detection , feature extraction and verification.
5. Keras\_facenet 🡪 This is a python library that gives access to facenet model which is a pretrained neural network model used for face recognition and verification.

**1.2.Initialize the Facenet model :**



Facenet model is initialized here and stored in a variable.

**1.3.Capture the image of the User through the Webcam:**



Inside the capture\_image() function :

a) Webcam is initialized by openCV

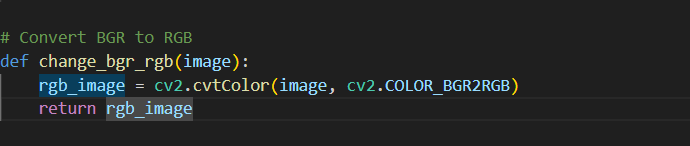
b) A single image on the frame is captured

c) Webcam is released

d) OpenCV windows are closed

e) Captured frame is returned

**1.4**. **convert the bgr image to rgb :**

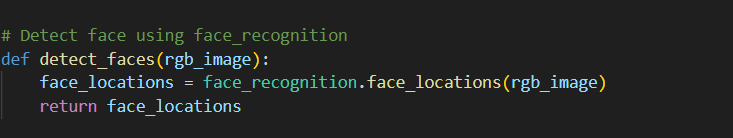
****

Inside the change\_bgr\_rgb() function:

a) Change the captured image from the BRG colour format to RGB .

b) Return the converted image

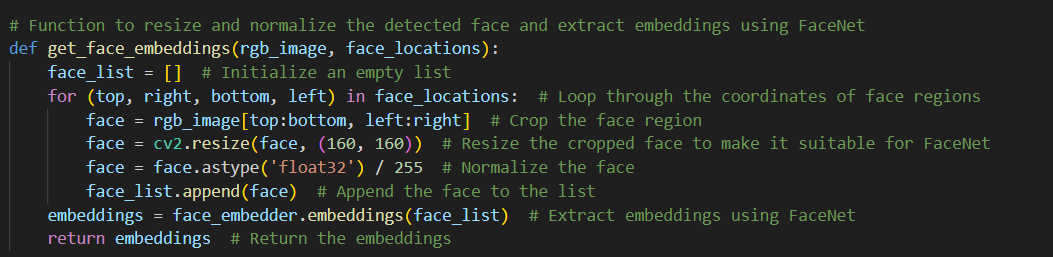
**1.5. Detect face using face\_recognition:**



Inside the detect\_faces() function:

1. The face\_recognition library is used to detect the face location in the given image
2. The result of above is returned.

**1.6.Resize and normalize the detected faces and extract the face embeddings using Facenet.**

****

Inside the get\_face\_embeddings() function:

a)The processed image and the face location are taken as the function parameters.

b)Loop through the coordinates of the face locations.

c)Crop the face region with the help of the coordinates

d)Resize the cropped face to make it suitable to be processed by the Facenet Model.

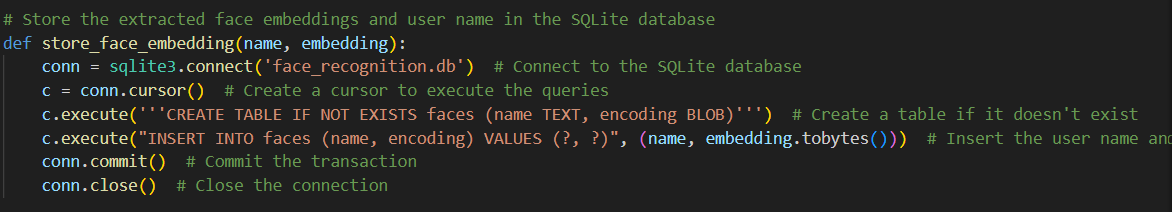
e)normalize the cropped face

f)append the face to the empty list

g)Feed the face data to the Facenet model to get the face embeddings

h)Return the embeddings.

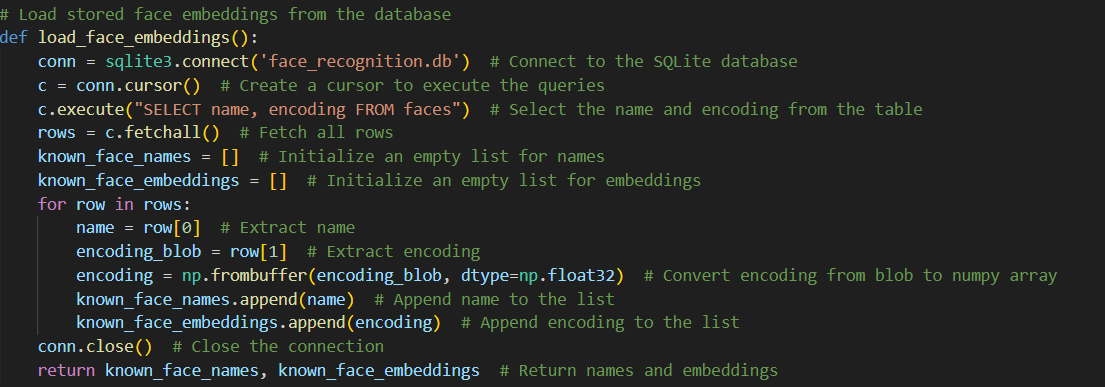
**1.7. Insert the face encodings of the captured faces along with names in SQLite database:**

****

**Inside the store\_face\_embedding() function:**

1. Connecting to the sqlite3 database
2. Create a cursor to access the database
3. Create a table called ‘Faces’ that will contain the user name and face embeddings .
4. If the table is already created , just insert the name and face embeddings and convert them to bytes.
5. Commit and close the connection .

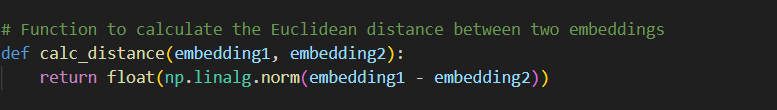
1.8. **Load the face encodings and name from the database it will be needed in the next step for verification :**

****

Inside the load\_face\_embeddings() function:

1. Connects to the sqlite3 database
2. Create a cursor to execute the commands
3. Sql statement to select all the name and face embeddings from the faces table
4. All the selected rows are fetched
5. Two empty lists are initialized
6. With a for loop iterate over each row that are fetched one by one
7. Store the name part in one variable
8. Store the encoding in another variable
9. Convert the encoding to numpy array from string
10. Finally append the stored name in the variable to the empty list
11. Append the stored and modified encoding to the empty list.
12. Finally close the connection to database
13. The face name and its embeddings are returned

**1.9.Calculate the eucledian distance between stored and captured face data**

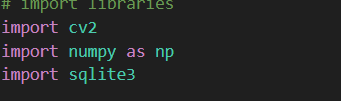


Inside the calc\_distance() function:

1. Euclidean distance between stored face embeddings and the captured face embeddings are done and the result show if it is the same or different person (less distance = same person , large distance = different person).

**2)Iris capture , storing and verification module**

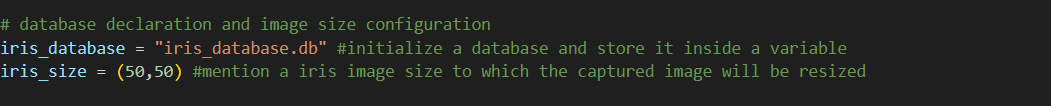
**2.1.Import libraries**



1. OpenCV(cv2) it is mainly imported to capture image through the webcam and process them . It is a open-source machine learning library.
2. Numpy ->This library is used to handle different numerical operations and manipulations of the face encodings further in the code.
3. Sqlite3 -> This is the database that is being used for storing the captured faces of the users along

with their names.

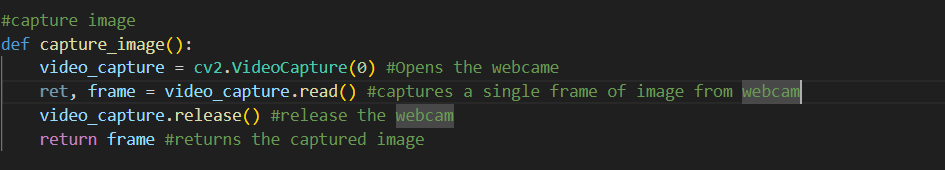
**2.2. Database and image size declaration.**



a)Database name is declared to store the captured iris features and stored in a variable

b) A fixed size is mentioned to which the iris images will be resized after capturing

**2.3.Capture Image**



Inside the capture\_image() function:

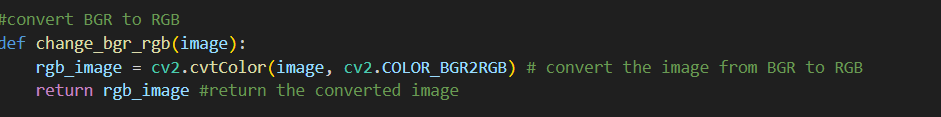
a)Open the webcam

b)Capture the image in a single frame

c)Release the webcam

d)Return the frame of the captured image

**2.4.Convert BGR to RGB**

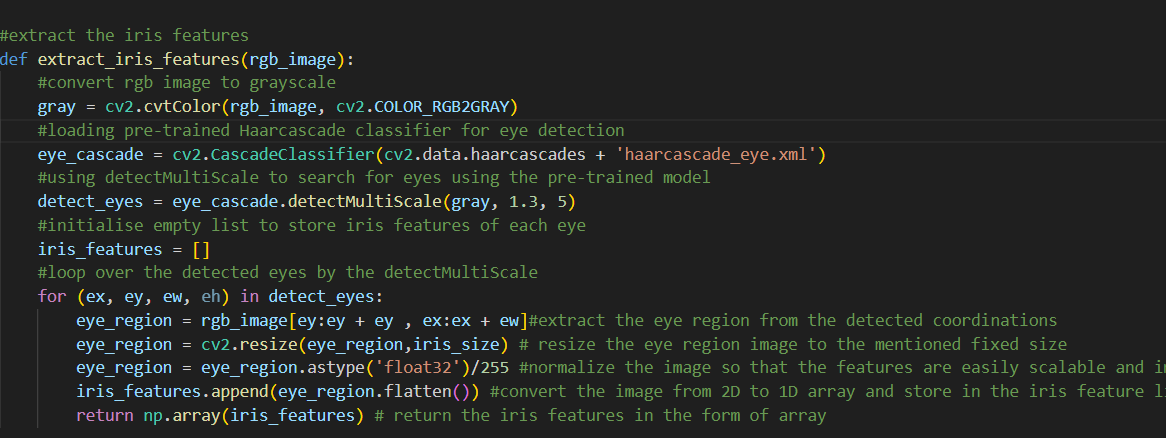


Inside the change\_bgr\_rgb(image) function:

a)Convert the captured image from the BGR to RGB format

b) Return the processed image

**2.5.Extract the irirs features from the processed image**



Inside the extract\_iris\_features(rgb\_image):

a)Convert the RGB image to grayscale format

b)Pre-trained Haarcascade classifier is loaded for eye detection.

c)After that detectMultiScale is used to search for eyes using the pre-trained model that was loaded.

d)Initialize an empty list to store the iris features of each detected eye.

e)Loop through the detected eyes and extract the eye-region from the detected coordinates

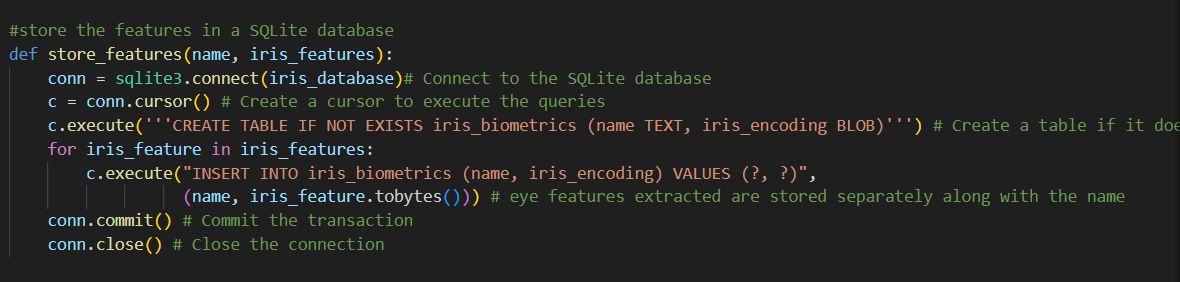
f)resize the detected eye-region to the fixed sixed declared at the beginning

g)normalize the image to make the features easily scalable and to improve the performance of the ML model.

h)convert the features from 2d to 1d array and store it in the iris feature list.

i)return the stored iris features in the form of a numpy array.

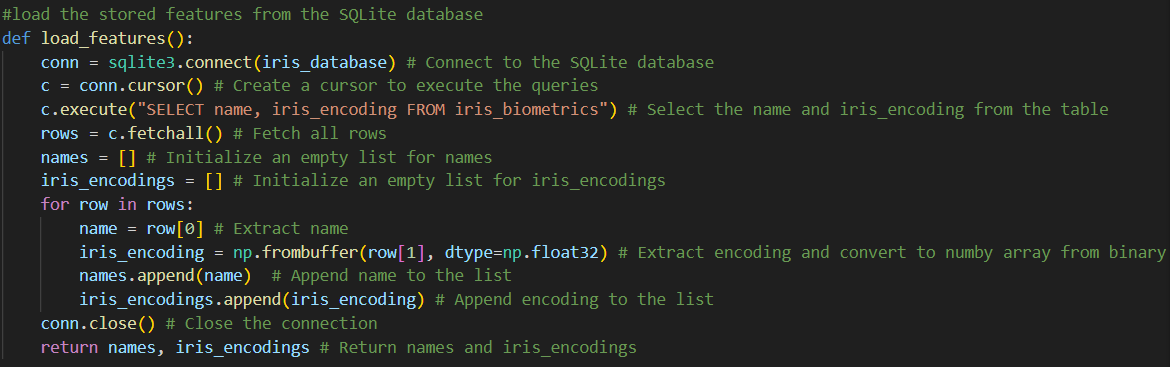
**2.6. Insert the iris encodings of the captured faces along with names in SQLite database:**

****

**Inside the store\_features() function:**

1. Connecting to the sqlite3 database
2. Create a cursor to access the database
3. Create a table called ‘iris\_biometrics’ that will contain the user name and iris encodings .
4. If the table is already created , just insert the name and iris encodings and convert them to bytes.
5. Commit and close the connection .

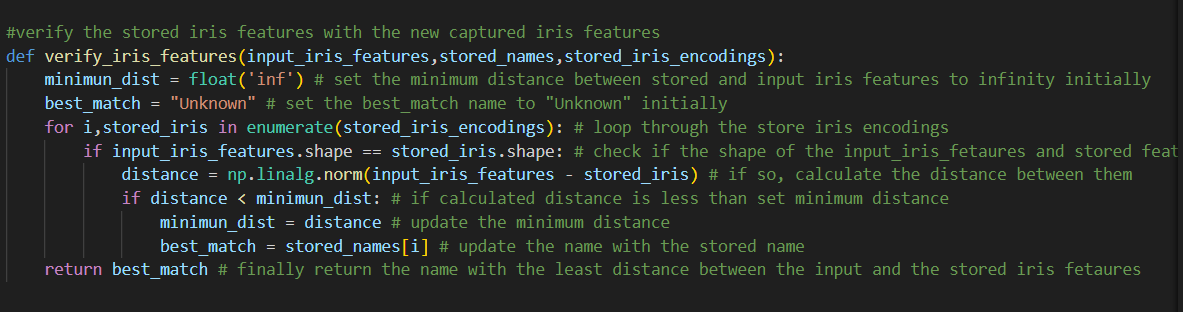
**2.7. Load the iris encodings and name from the database it will be needed in the next step for verification :**



Inside the load\_features():

1. Connects to the sqlite3 database
2. Create a cursor to execute the commands
3. Sql statement to select all the name and iris encodings from the faces table
4. All the selected rows are fetched
5. Two empty lists are initialized
6. With a for loop iterate over each row that are fetched one by one
7. Store the name part in one variable
8. Store the encoding in another variable
9. Convert the encoding to numpy array from string
10. Finally append the stored name in the variable to the empty list
11. Append the stored and modified encoding to the empty list.
12. Finally close the connection to database
13. The iris name and its encodings are returned

**2.8.Verify the stored iris features with the captured iris features:**



Inside the verify\_iris\_features():

a)Set the minimum distance between the stored iris features and the input iris features to infinity .

b)Initialize the best match to “Unknown”.

c)Loop through the stored iris encodings

d)check if the shape of the stored iris encodings and that of the input iris encodings match

e)if the match, the distance between them is calculated

f)if the new calculated distance is less than the initialized minimum distance

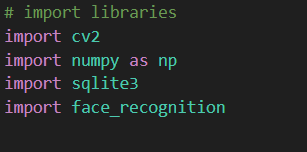
g)update the minimum distance as the newly calculated distance

h)update the name with the stored name

i)return the name with the least distance between stored iris and input iris features, that will be the verified user.

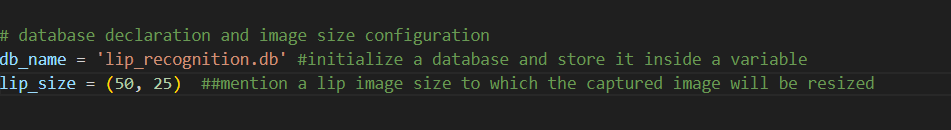
**3)Lips feature capture, storing and verification module.**

**3.1)Import Libraries**



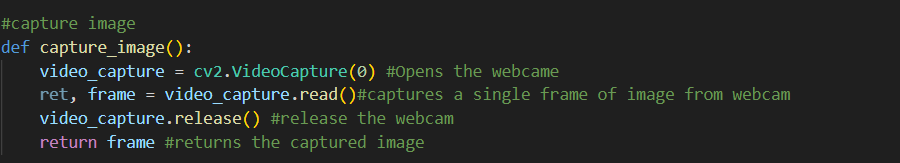
1. OpenCV(cv2) it is mainly imported to capture image through the webcam and process them . It is a open-source machine learning library.
2. Numpy ->This library is used to handle different numerical operations and manipulations of the face encodings further in the code.
3. Sqlite3 -> This is the database that is being used for storing the captured faces of the users along with their names.
4. face\_recognition This is a machine learning library that contains pre-trained deep learning models by using dlib under the hood , and helps in face detection , feature extraction and verification.

**3.2) Database and image size declaration.**



1. Database name is declared to store the captured lip features and stored in a variable
2. A fixed size is mentioned to which the lip images will be resized after capturing

**3.3) Capture Image**



Inside the capture\_image() function:

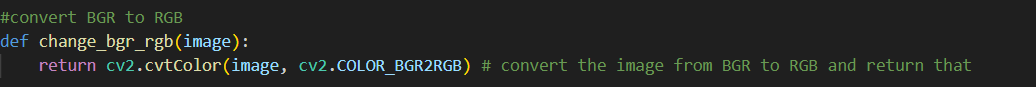
a)Open the webcam

b)Capture the image in a single frame

c)Release the webcam

d)Return the frame of the captured image

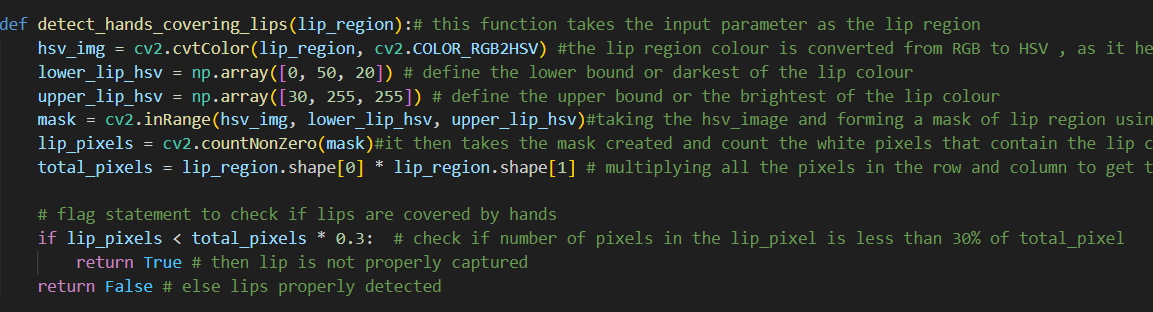
**3.4) Convert BGR to RGB**



Inside the change\_bgr\_rgb(image) function:

1. Convert the captured image from the BGR to RGB format
2. Return the processed image

**3.5)Detect hands or something else covering the lips**



Inside the detect\_hands\_covering\_lips(lip\_region) function:

a)The lip region image is converted from RGB to HSV format.

b)The lower bound or darkest region of the lip colour is defined

c)The upper bound or the lightest region of the lip colour is defined.

d)A mask image is formed by taking the hsv image and considering the lower and upper bound of the lip colour.

e)The mask that is created is then taken into consideration and the white pixels are counted as that detects the lip region.

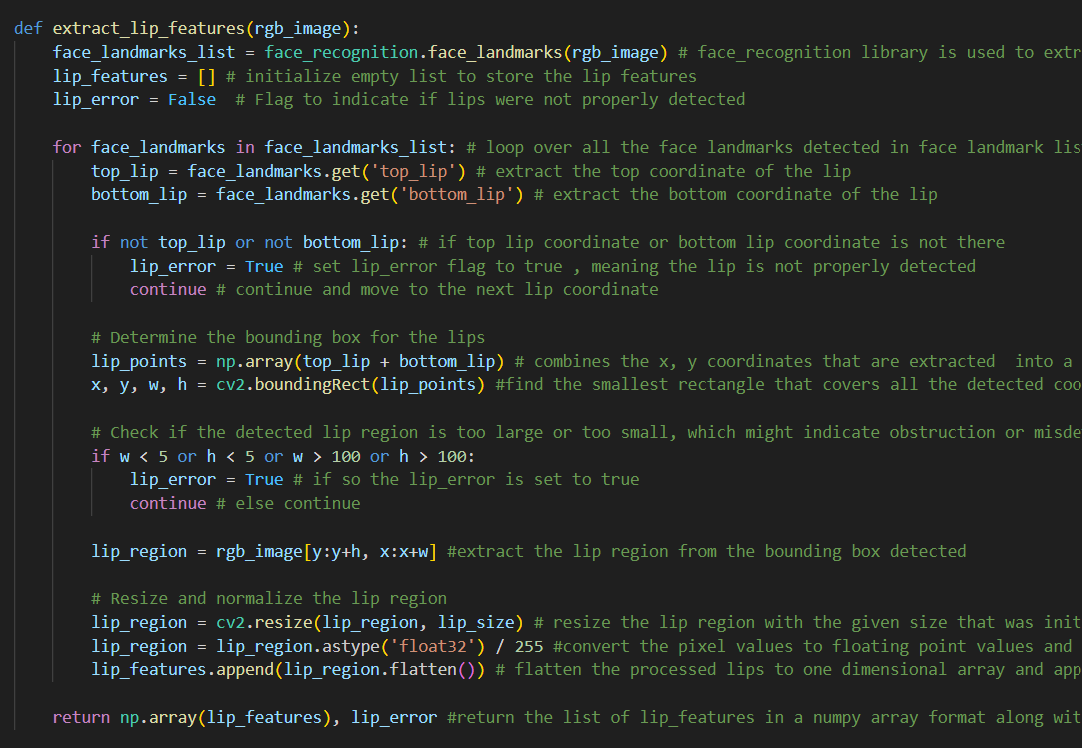
f)All the pixels in the row and column is then multiplied to get the total pixel in the detected lip region.

g)Check is done , where it is checked if the number if counted lip pixels is less than 30% of the total\_pixels.

h)If so, then the lip is not captured properly.

i)Else, lip is detected properly.

**3.6)Extract the lip features:**



Inside the extract\_lip\_features(rgb\_image):

a)Extract the face landmarks using the face\_recognition library.

b)Initialize the empty list to store the lip features.

c)Initialize a flag as False.

d)loop over the face landmarks in the list

e)Extract the top coordinate of the lip

f)Extract the bottom coordinate of the lip

g)If top lip coordinate or bottom coordinate is not there, set the flag as True indicating error in properly detecting the lips

h)continue and move to the next coordinate

i)determine the bounding box for the lips by combining the x and the y coordinates

j)find the smallest rectangle that covers the detected coordinates

j)check if the detected lip region is too small or too large , that can indicate obstruction

j)if so, then the lip error is set to true

k)else continue

l)extract the lip region from the bounding box that is detected

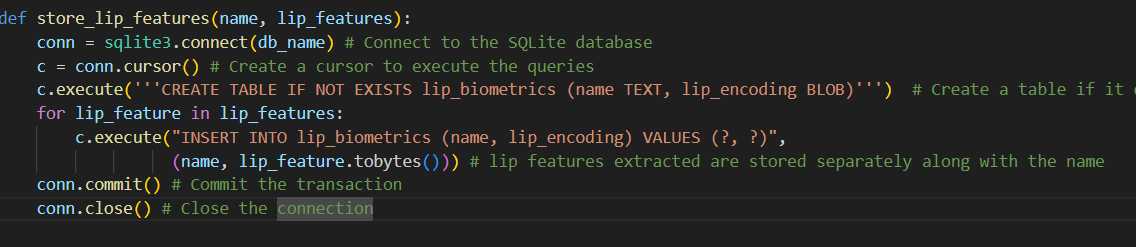
m)Resize the lip region with the initialized size

n)Convert the pixel values to floating point values

o)flatten the processed lip to one dimensional array and append it to the initialized empty list

p)Return the .lip features in the numpy format along with the flag value of detection.

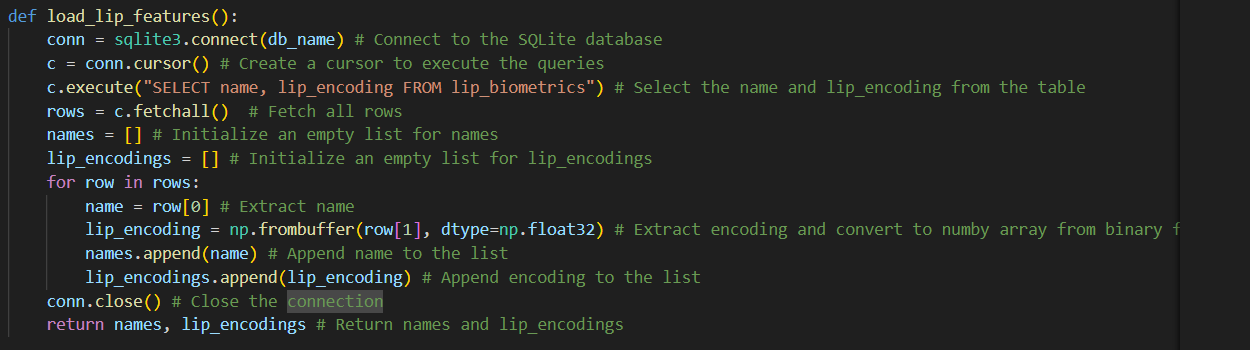
**3.7) Insert the lip encodings of the captured faces along with names in SQLite database:**

****

**Inside the store\_lip\_features() function:**

1. Connecting to the sqlite3 database
2. Create a cursor to access the database
3. Create a table called ‘lip\_biometrics’ that will contain the user name and lip encodings .
4. If the table is already created , just insert the name and lip encodings and convert them to bytes.
5. Commit and close the connection .

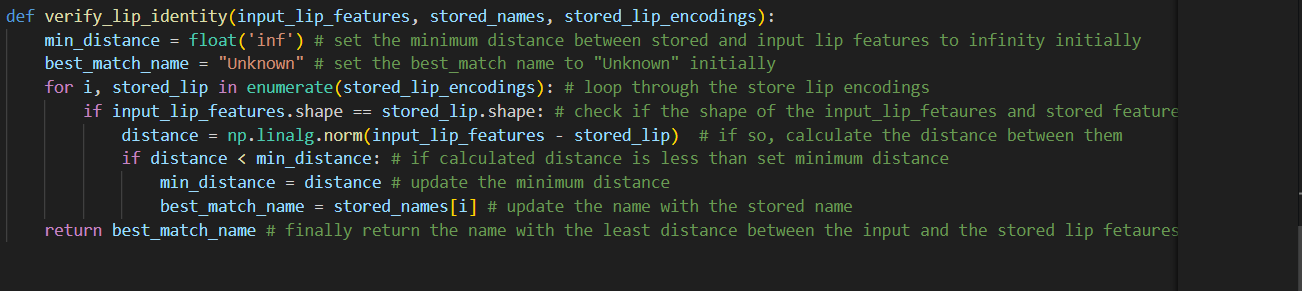
**3.8) Load the lip encodings and name from the database it will be needed in the next step for verification :**



Inside the load\_lip\_features():

1. Connects to the sqlite3 database
2. Create a cursor to execute the commands
3. Sql statement to select all the name and lip encodings from the faces table
4. All the selected rows are fetched
5. Two empty lists are initialized
6. With a for loop iterate over each row that are fetched one by one
7. Store the name part in one variable
8. Store the encoding in another variable
9. Convert the encoding to numpy array from string
10. Finally append the stored name in the variable to the empty list
11. Append the stored and modified encoding to the empty list.
12. Finally close the connection to database
13. The lip name and its encodings are returned

**3.9.Verify the stored lip features with the captured lip features:**



Inside the verify\_lip\_identity():

a)Set the minimum distance between the stored lip features and the input lip features to infinity .

b)Initialize the best match to “Unknown”.

c)Loop through the stored lip encodings

d)check if the shape of the stored lip encodings and that of the input lip encodings match

e)if the match, the distance between them is calculated

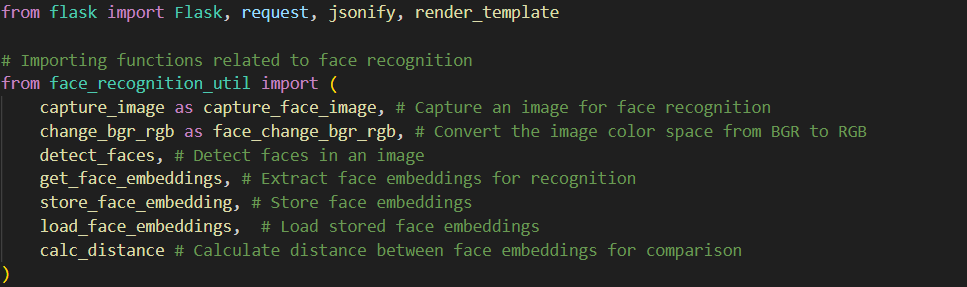
f)if the new calculated distance is less than the initialized minimum distance

g)update the minimum distance as the newly calculated distance

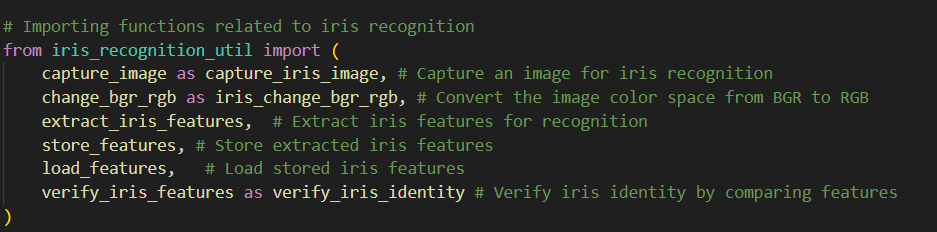
h)update the name with the stored name

i)return the name with the least distance between stored iris and input iris features, that will be the verified user.

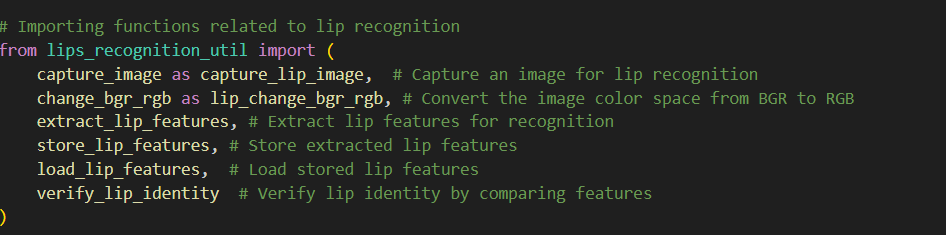
**4) Main Flask code to combine and make the functionalities work**



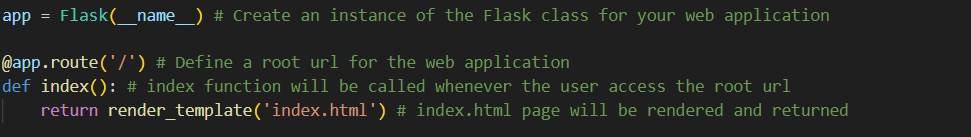
1. Import all the defined functions from the face\_recognition\_util script.



1. Import all the defined functions from the iris\_recognition\_util script.



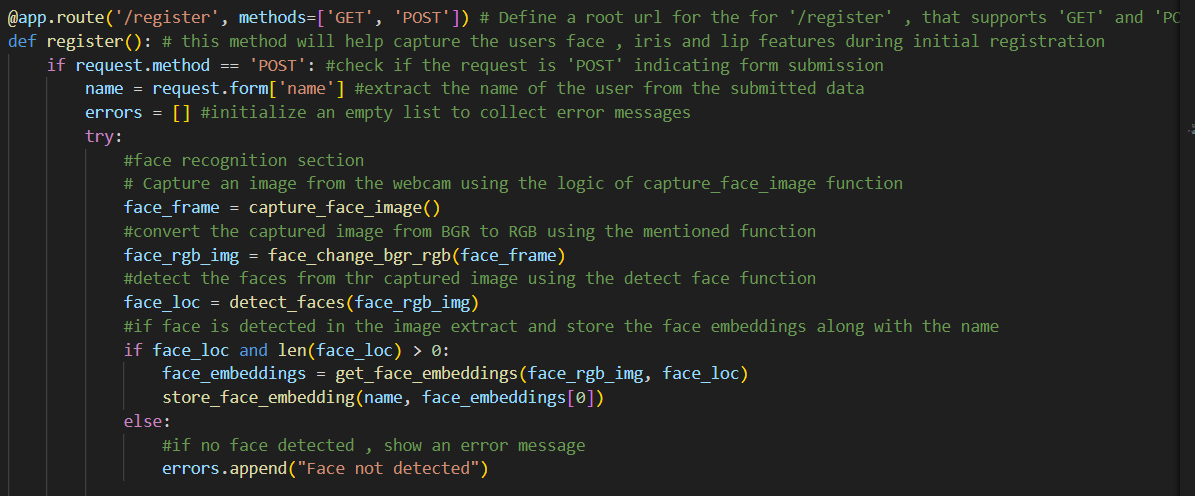
1. Import all the defined functions from the lips\_recognition\_util script.



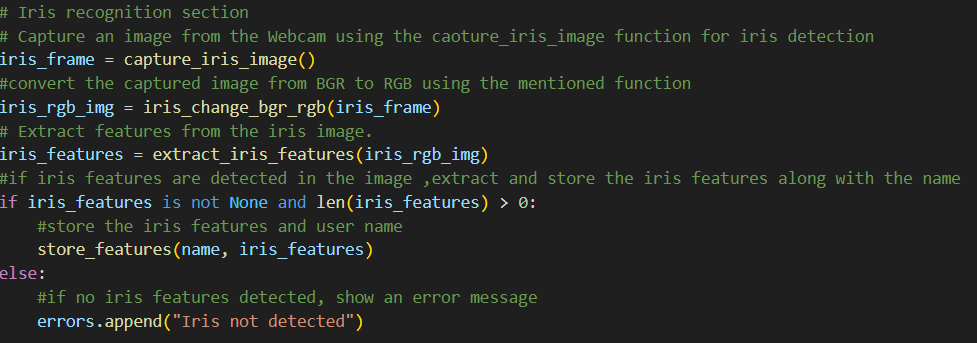
1. At first an instance of the Flask class is created.
2. Then, the root for the home page is defined.
3. Inside the function ,index(), when it is accessed it renders and returns the index.html page

**Creating second route :**

**4.2) Face recognition section**

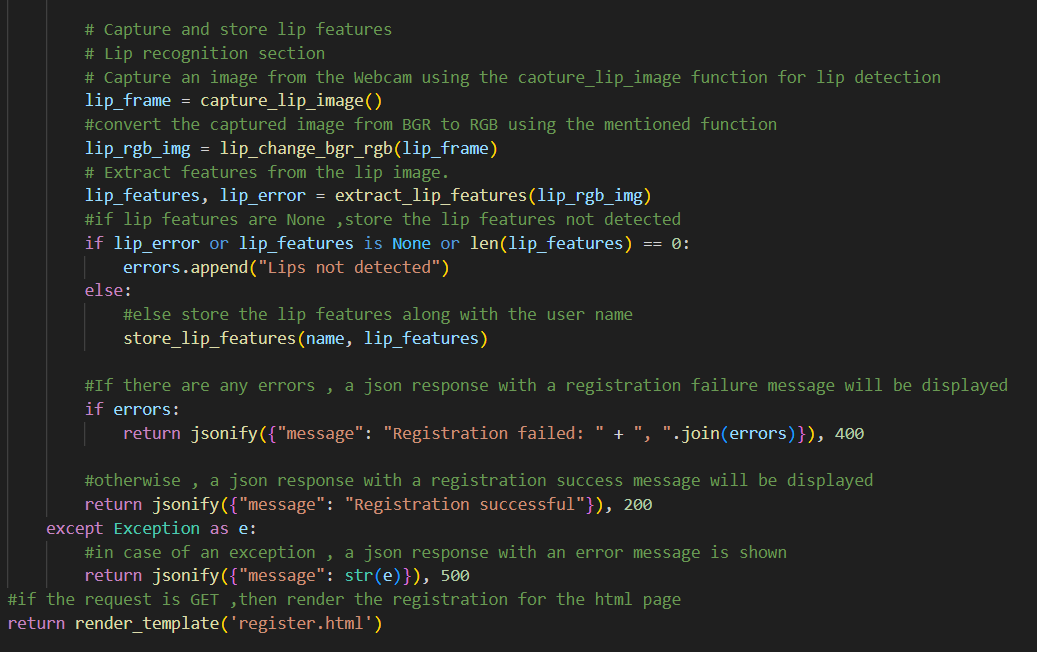


1. Another router ‘/register’ is defined which can handle both ‘GET’ and ‘POST’ requests.
2. Inside the register() function
3. A check is done for if the request method is ‘POST’ that means user submitted the form
4. If so, the name is retrieved from the form
5. Initialize an empty list to collect error messages
6. Capture an image from the webcam using the logic on capture\_face\_image func.
7. Convert the captured image from BGR to RGB using the given function.
8. Detect the faces from the captured image using the detect face function.
9. If the face is detected in the image then extract and store the face embeddings along with the name
10. Else show , no face detected with an error message.



**4.3)Iris recognition section:**

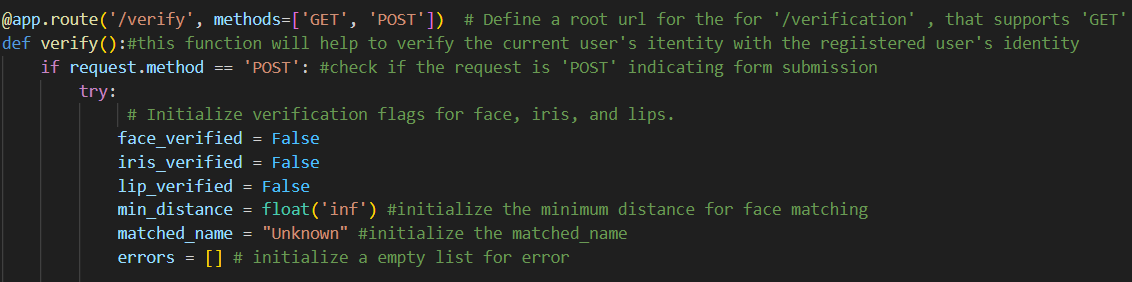
1. Capture an image from the webcam using capture\_iris\_image() function.
2. Convert the captured image from BGR to RGB using the mentioned() function.
3. Extract the features from the iris image.
4. If the iris features are detected in the image then extract and stored the iris features along with the name of the person
5. Then store the iris features and user name.
6. Else , no iris features are detected and show an error message.



**4.4)Lip recognition section:**

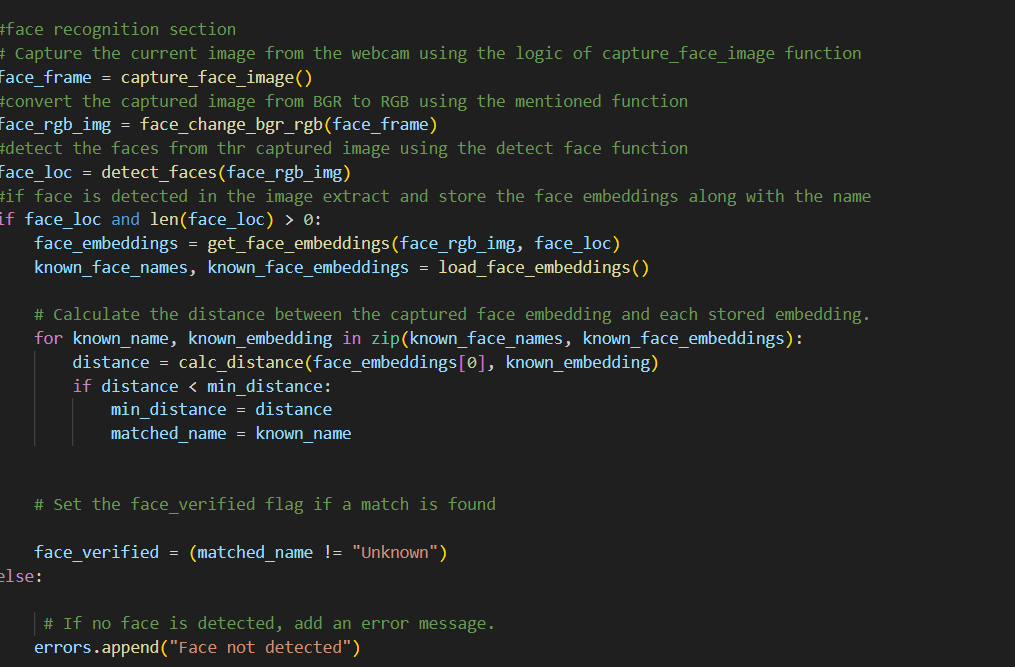
1. Capture an image from the webcam using capture\_lip\_image() function.
2. Convert the image from BGR to RGB using the functions mentioned .
3. Extract features from the lip image using the defined function.
4. If lip features are None then in the error message list store , “no lips detected”.
5. Else store lip features along with the user name
6. If there are any error , a json response with a registration failure message will be displayed.
7. Else, a json response with a registration success message will be displayed.
8. In case of exception , a json response with a error message will be shown.
9. Finally if the request is GET, then render the registration page from the register.html.

**4.5)Creating third route for verification:**



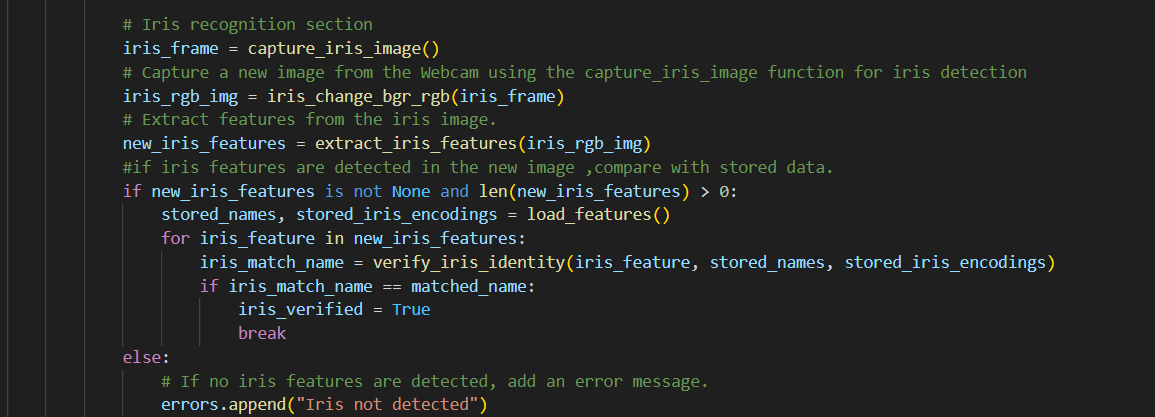
1. The third route is created here, called ‘verify’ and it can also handle both ‘GET’ and ‘POST’ requests.
2. Inside the verify() function, it is checked if the request is ‘POST’, which happens when user has started a verification process.
3. Initialize the three flags for face,iris and lips and set them to False ,initialize minimum\_distance for face matching to infinity and initialize the matched name to “Unknown” along with an empty list to store errors.

**4.6)Face verification section**



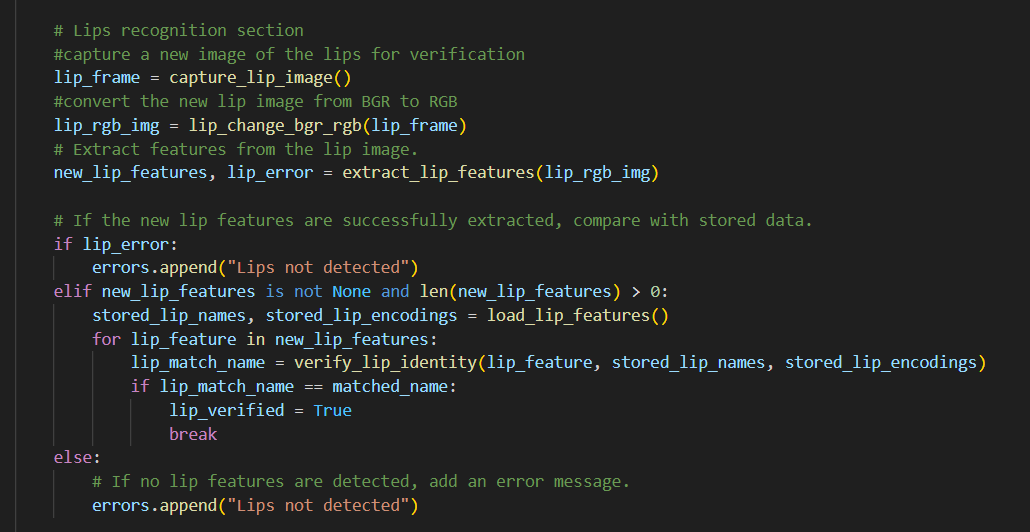
1. Capture the new face image from the webcam using the face capture logic
2. Convert the new captured face image from BGR to RGB using the mentioned function.
3. Detect the faces from the captured faces using the detect face function
4. If the face is detected in the newly captured and processed image then extract and store the face embeddings along with the name
5. Calculate the distance between the newly captured face embedding and each stored face embeddings.
6. If the distance between the two faces is smaller than the calculated minimum distance then update the minimum distance value to the current distance and the matched user name .
7. Set the face\_verified flag is the matching face is found.
8. If no matching face is found , then add an error message.

**4.7)Iris verification section**



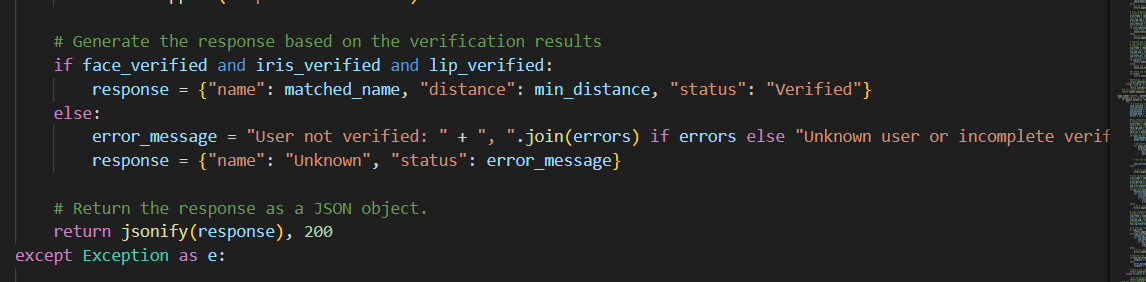
1. Capture the new image using webcam using the capture\_iris\_image function for iris detection
2. Convert the new image from BGR to RGB
3. Extract the features from the iris image using the function .
4. If iris features are detected in the new image then compare with the stored iris features and set the flag value of iris\_verified to true.
5. If no iris features are detected then add an error message.

**4.8)Lip verification section:**



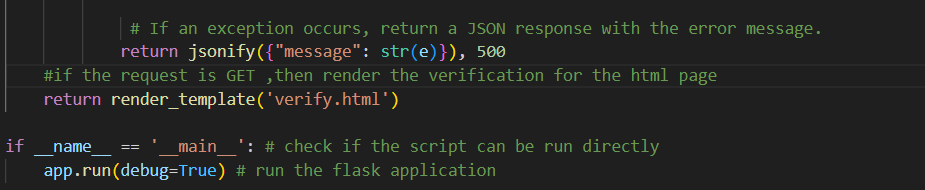
1. Capture the new image using the capture\_lip\_image function , that will help in lip\_detection
2. Convert the newly captured image from BGR to RGB
3. Extract the features from the lip image
4. If the new lip features are successfully extracted , compare the features extracted from the new lip image with that of the stored lips image.
5. If its matched then set the flag value to true.
6. Else , in the error list put, “Lips not detected”.

**4.9)Check to see if all three features match and set responses accordingly:**



1. Now check if the face\_verified, iris\_verified and lip\_verified is set to true , if so then , set the response as username,distance,status as verified.
2. Else, set a User not verified along with the reason.
3. Return the response in a JSON format .

**5) Return the responses according to the condition and run the app:**



1. If any exception occurs , return the error message in the JSON format.
2. Finally, if the request is a GET , then render the verification page from verify.html
3. This block is used to ensure , that the application starts only when run directly.
4. Starts the flask application server, with debugging enabled.

**6)Frontend part of the application:**

****

The entire code in this script is used for creating the front-end and the landing page for our simple facial recognition project.

****

This page contains the html script for the user registration frontend part.

****

This page contains the html script for the user verification frontend part.

**Summary:**

Hence, this project consists of a facial recognition system using a Flask Web application. Here, the user’s facial data, iris data,lip data and name is captured during registration and stored in the SQLite database and verification of faces is performed using Facenet model,iris and lip verification is done by using face\_recognition and openCV library features .The flask routes help user interactions during registration and verification by connecting the backend with the frontend.