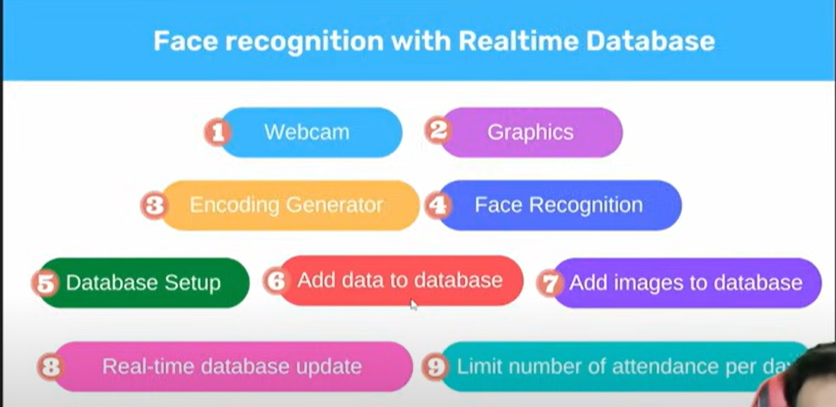
**SMART ATTENDANCE SYSTEM**

**OVERVIEW:**

This project is a combination of ML model and connecting it with the data base. The key concepts are going to be:

* Python
* ML (Basic)
* Open CV
* My SQL

So, Our approach for this project is given below:



Let’s do and understand the working of our project step by step.

**NEEDED COMPONENTS:**

* Any IDE (VS Code).
* Computer with web cam.
* Face recognition library.
* Images of the student.

Download VS code: [Link](https://code.visualstudio.com/download).

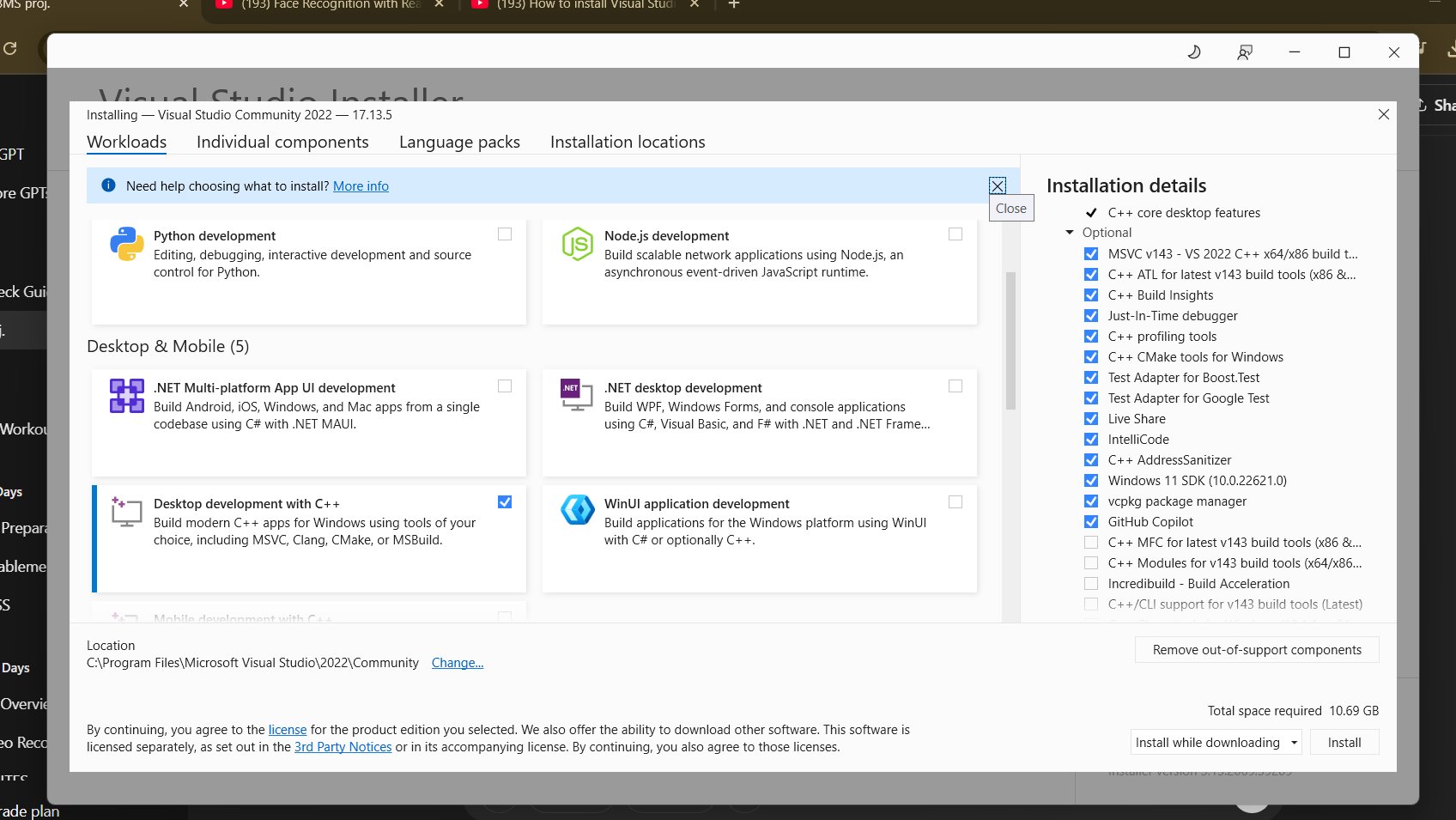
Download tutorial: [Tutorial link](https://youtu.be/bN6DE-4uFNo?si=rza3H1T-2PC8apoY).

We should also install Visual Studio. Because, the face recognition library (dlib and cmake are hard to download). These two libraries should be compiled in c++. So, Visual Studio helps in that case.

Download Visual Studio: [Link](https://visualstudio.microsoft.com/downloads/).

Download Tutorial: [Link](https://youtu.be/FIgfRebWoPM?si=wFPEe3Vbe2WAqhyk).

As we need **C++ compiler**, do a small change in the setup.



Now lets download the required packages like cmake, dlib and face recognition. Note that, don’t install cmake as pip install cmake. Because, **pip installs only python essentials**. Download cmake from website.

Cmake download website: [Link](https://cmake.org/download/).

Cmake installation tutorial: [Link](https://youtu.be/GJy_bw2Vg5c?si=nqAX7lulSw66WjgI).

**Note:** Python 3.13 which is recently released doesn’t support dlib library. Dlib plays important role in our project. So, check for the version and degrade the version, or try deleting the python and download till any pyhton 3.10 and delete the folder. Now, in terminal say pip install dlib.

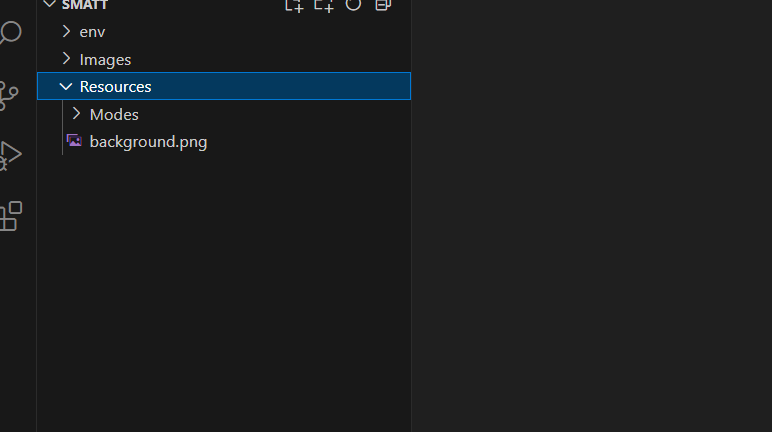
**Libraries required:**

* Cmake
* dlib
* face-recognition
* cvzone

I would prefer to create a **virtual environment**. It helps program files and packages isolate from other system files and many other advantages.

* python -m venv env (Create a folder named env)
* env\Scripts\activate (Instead of env you can give any name here and above)

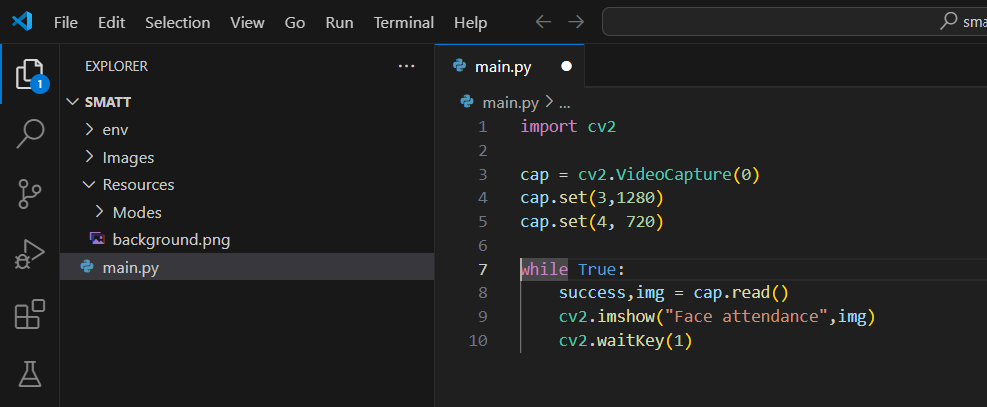
Now, this is the basic skeleton before starting to code:



Images folder has the images of the students. Resources folder has the GUI template of our project.

**WEBCAM:**

Webcam is very important for our project. We use Opencv library for opening the webcam.



Videocapture function is used to access the video streams like webcam, external camera. VideoCapture(0) accesses the webcam.

Let’s see the cv2 functions and their role in a line:

|  |  |
| --- | --- |
| * cv2.VideoCapture() | * Opens webcam or video file for frame capture. |

|  |  |
| --- | --- |
| * cv2.imshow() | * Displays an image/frame in a window. |

|  |  |
| --- | --- |
| * cv2.waitKey() | * Waits for a key press for a specified time (in ms). |

|  |  |
| --- | --- |
| * cv2.destroyAllWindows() | * Closes all OpenCV windows. |

|  |  |
| --- | --- |
| * cv2.destroyWindow() | * Closes a specific OpenCV window. |

|  |  |
| --- | --- |
| * cv2.imread() | * Loads an image from file. |

|  |  |
| --- | --- |
| * cv2.imwrite() | * Saves an image to file. |

|  |  |
| --- | --- |
| * cv2.cvtColor() | * Converts image color space (e.g., BGR to RGB or grayscale). |

|  |  |
| --- | --- |
| * cv2.resize() | * Changes the size of an image. |

|  |  |
| --- | --- |
| * cv2.rectangle() | * Draws a rectangle on an image. |

|  |  |
| --- | --- |
| * cv2.putText() | * Adds text to an image. |

|  |  |
| --- | --- |
| * cv2.face.LBPHFaceRecognizer\_create() | * Creates a recognizer object (used in face recognition). |

|  |  |
| --- | --- |
| * cv2.CascadeClassifier() | * Loads a pre-trained model (like face detection XML). |

|  |  |
| --- | --- |
| * cv2.findContours() | * Finds contours in a binary image. |

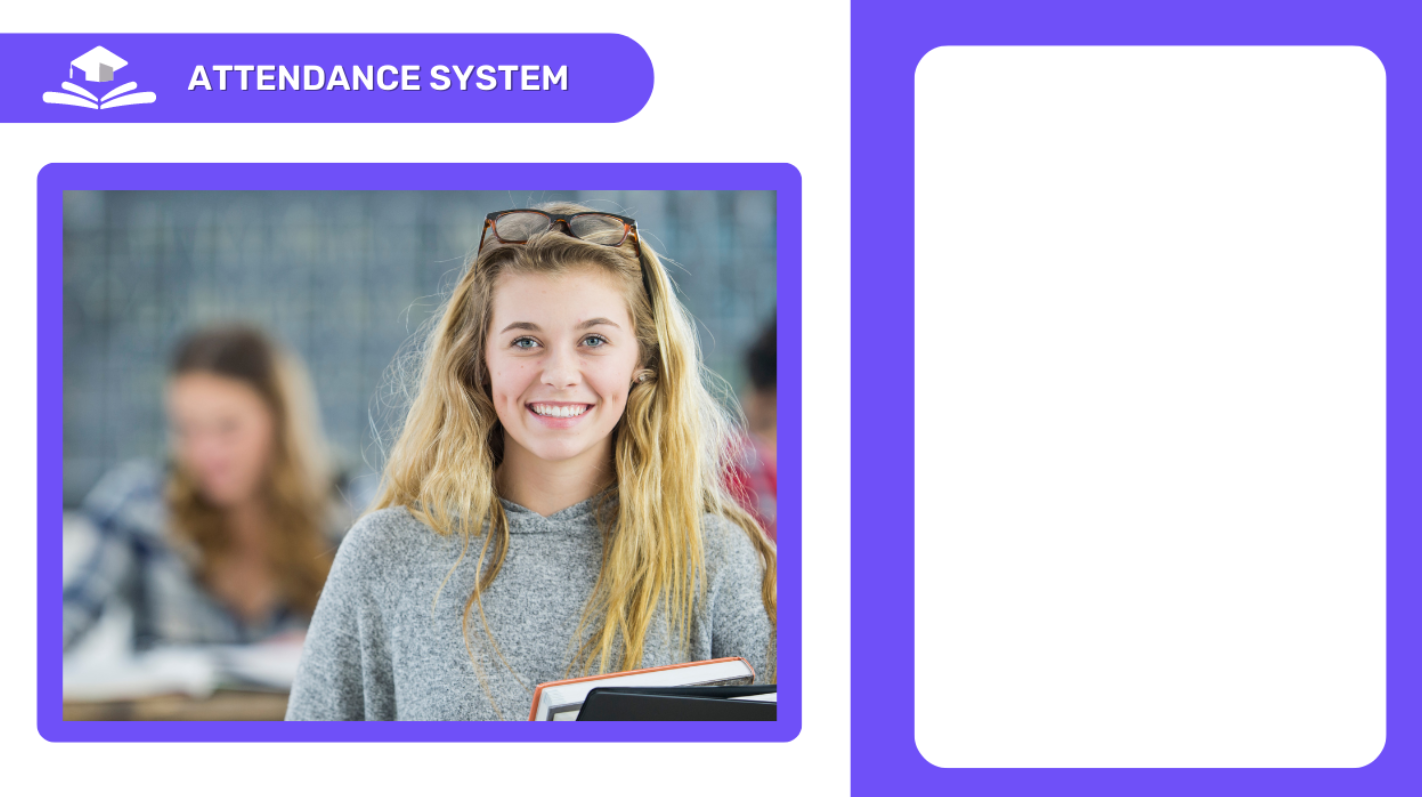
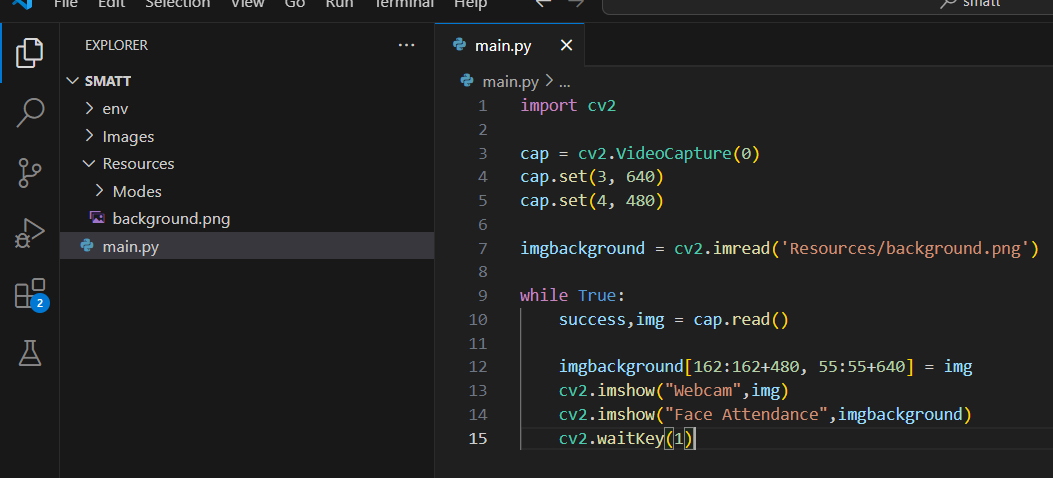
|  |  |
| --- | --- |
| * cv2.drawContours() | * Draws contours on an image. |

|  |  |
| --- | --- |
| * cv2.GaussianBlur() | * Applies Gaussian blur to reduce noise. |

|  |  |
| --- | --- |
| * cv2.threshold() | * Applies a binary threshold to an image. |

**GRAPHICS:**

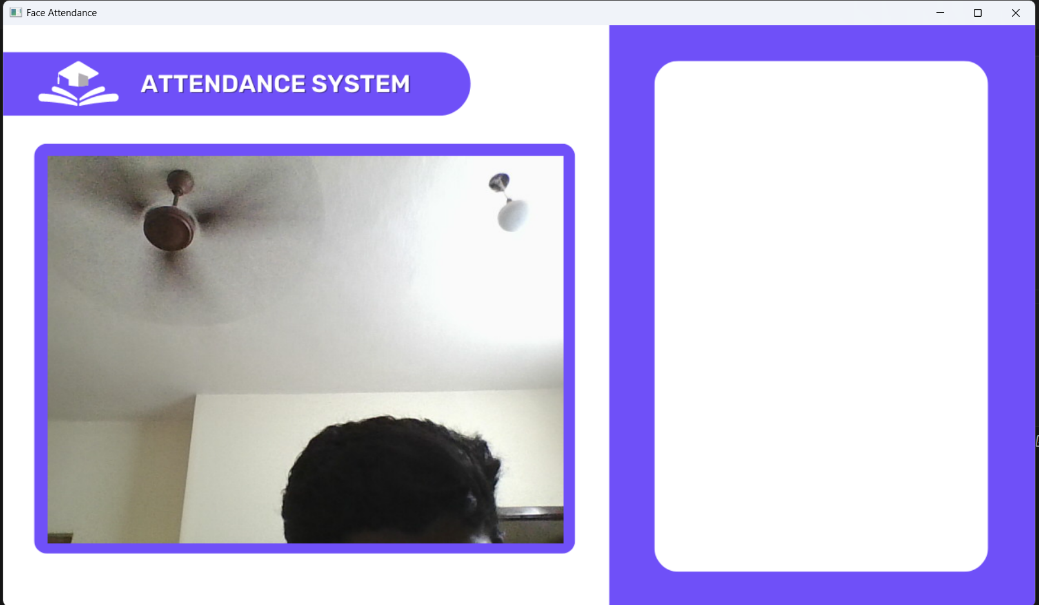
Now lets focus on the GUI part of our project. We kept all the GUI in the Resource folder. Main GUI is:



We need the webcam to be overlayed in the webcam area. So, lets fix the bound of the webcam and overlay.Till now we have opened the webcam and overlayed in the GUI.

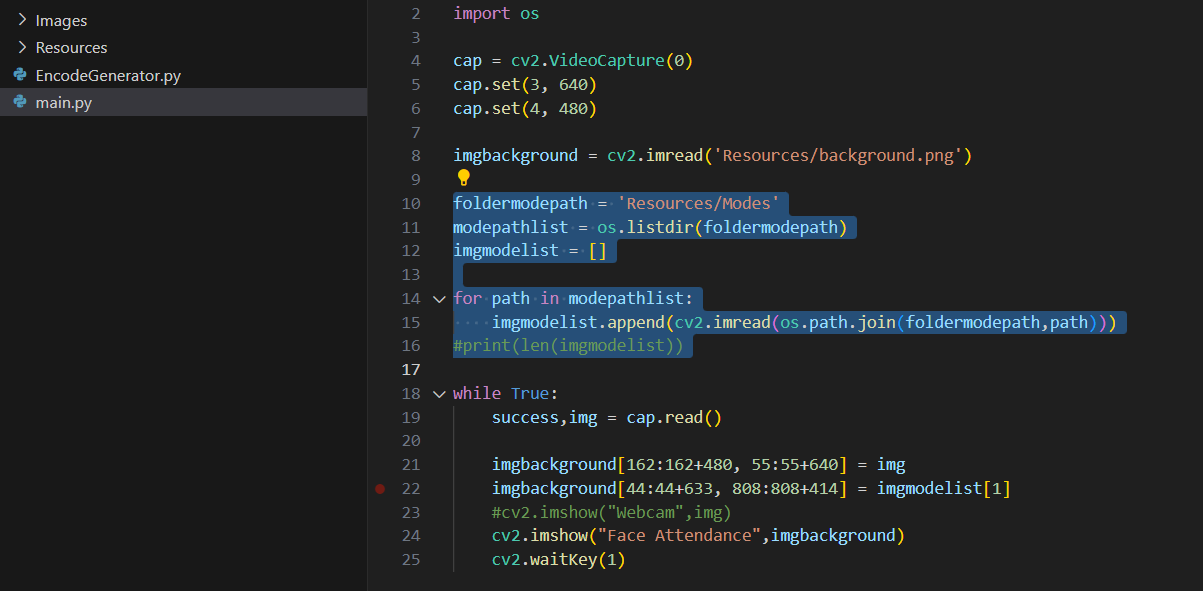
Line 12 pastes the webcam frame (img) onto the imgbackground image, starting from position [162, 55].

* You're placing the webcam feed **on top of the background** like a live window.



As you can see, the above code overlayed camera and GUI.

Now, the second step is to add and overlay the four modes on the left side of the GUI. We just need a couple lines of code to do that.



Those highlighted and line no.22 helps to get our desired outcome.

EXPLAINATION:

foldermodepath = 'Resources/Modes'

**Specifies the path to the Modes folder**, which likely contains different UI images/modes like "Attendance Taken", "Active", “Already marked”, etc.

modepathlist = os.listdir(foldermodepath)

**Lists all files inside the Resources/Modes folder**.

* For example, if the folder has 3 images: mode1.png, mode2.png, mode3.png,  
  then modepathlist = ['mode1.png', 'mode2.png', 'mode3.png']

**listdir** is one of the function of os which lists or showcases all the files inside the given path or folder.

imgmodelist = []

**Initializes an empty list** to later store all the mode images after reading them with cv2.imread().

for path in modepathlist:

imgmodelist.append(cv2.imread(os.path.join(foldermodepath, path)))

**Note:** This loop plays an important role in the code. Till now we have given the path and extracted the images from the path and we had created a array variable to add all the mode images.

**Loop through all filenames** in the modepathlist (like 'mode1.png', 'mode2.png', etc.)

* os.path.join(foldermodepath, path) combines folder path + filename, e.g. 'Resources/Modes/mode1.png'
* cv2.imread(...) reads the image
* .append(...) adds that image to imgmodelist list

After the loop your array be like:

imgmodelist[0] → mode1.png (image data)

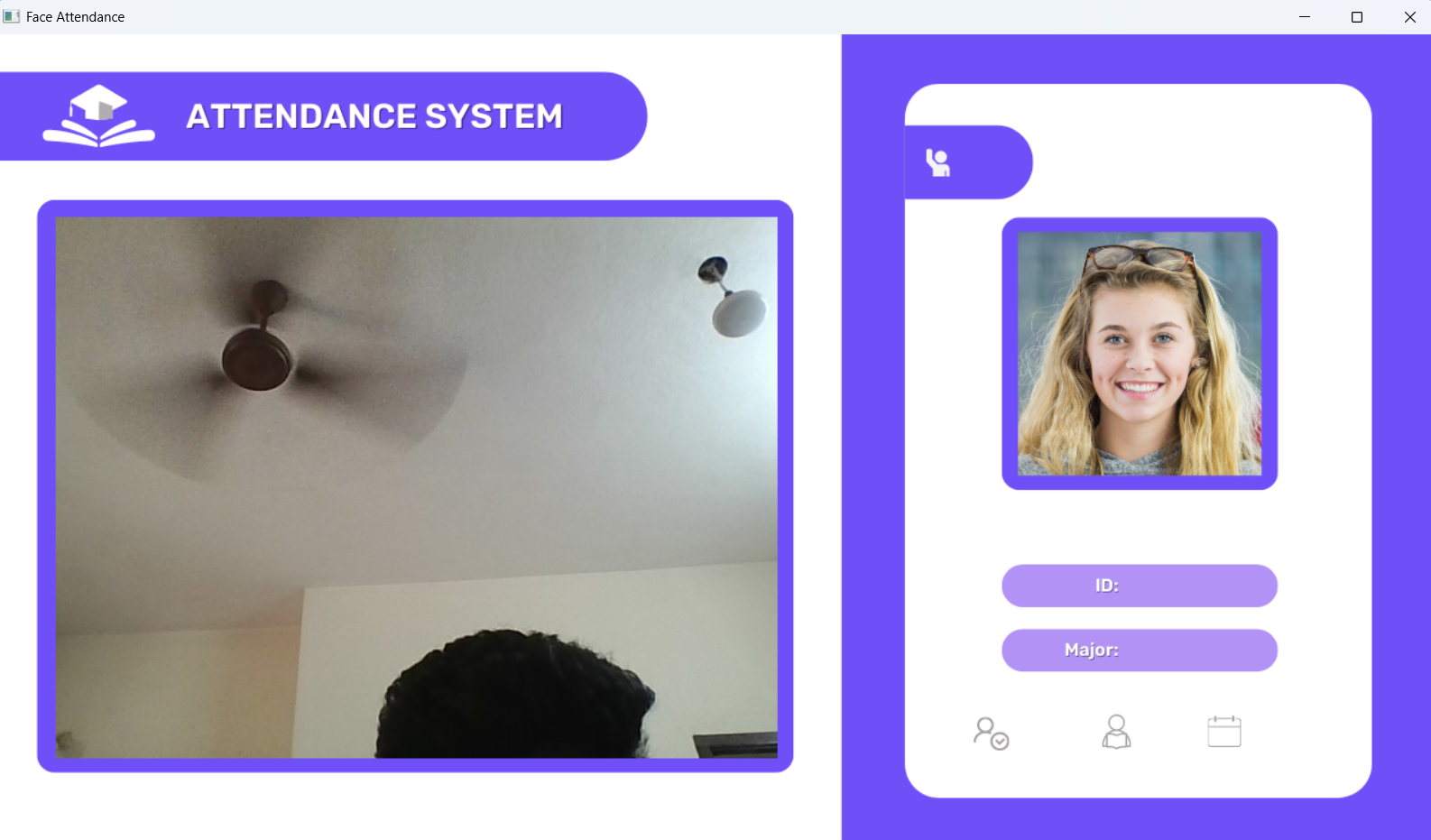
imgmodelist[1] → mode2.png

imgmodelist[2] → mode3.png

Now we should overlay with the main GUI. After overlaying, we can access every mode image by their index no. (starts from 0).

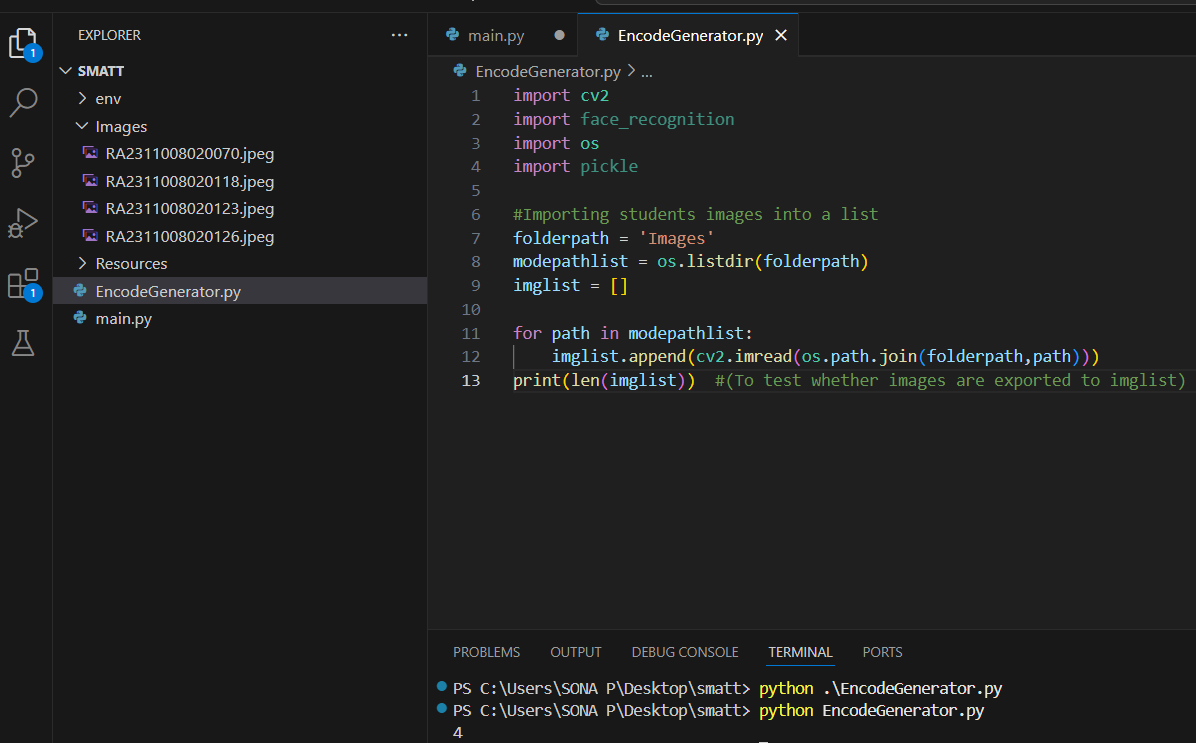
imgbackground[44:44+633, 808:808+414] = imgmodelist[1]

For now, lets leave it here, in future we should not give index in the code. The code should dynamically change. The output for the code till now is:

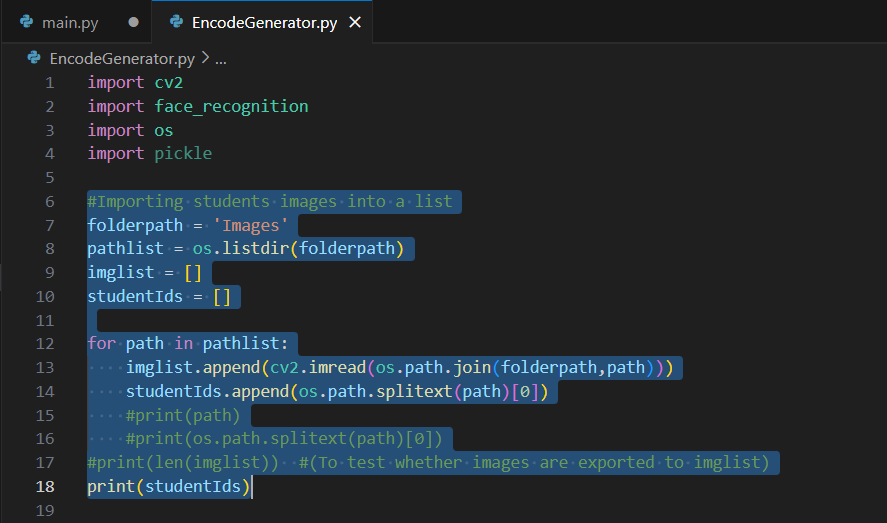


**ENCODING GENERATOR:**

We will the remaining GUI part later, now let’s move to the encoding generator. Create a python file. In this file we are going to use the photos from the dataset and perform some testing before we encode/ train them. The code is similar to the overlaying the modes.



In this EncodeGenerator.py file, we should import the required libraries before coding. Line 13 we are just testing whether all 4 images are imported to the imglist(list of the students images).



We just added line 13, because we need ID of the students also. So, we using os.splitext(path)[0]. Let’s take an example we have 1 photo(img1.jpeg) in Images folder and we are running this code the line 13 will split name of the file in list like [‘img1’, ‘.jpeg’]. As we need just the name/ ID of the student we are using index 0(first element) and that will be stored in studentIds array. Now we finally loaded the student’s images and ID’s in two separate arrays(imglist[], studentIds[]). Now let’s focus on encoding the student images.

WHAT IS ENCODING?

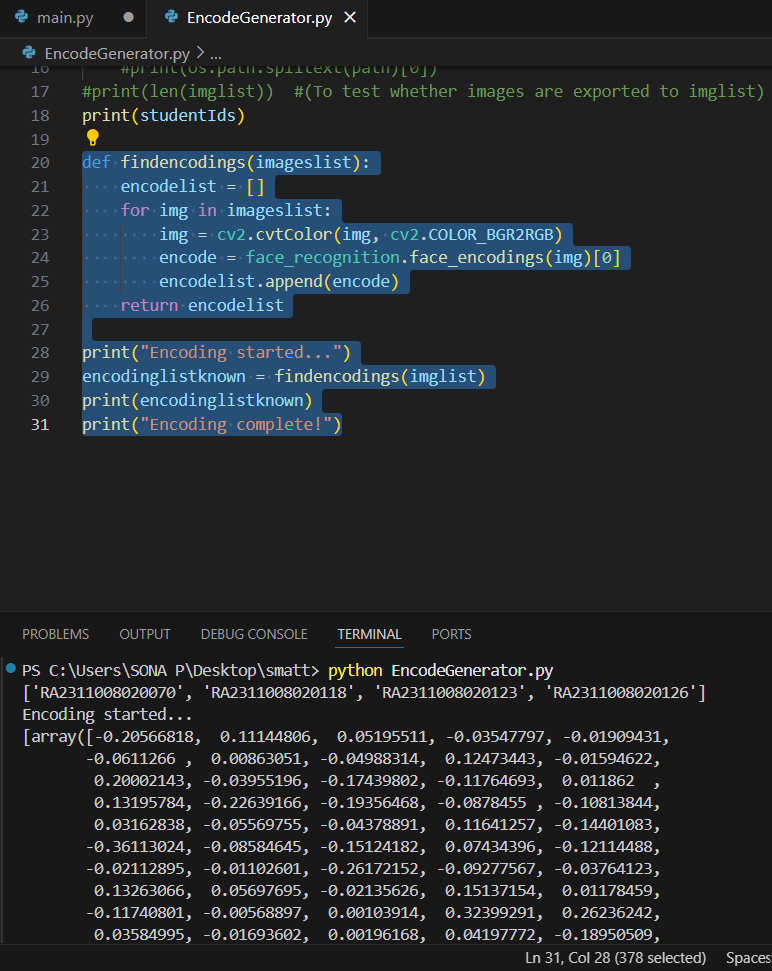
**Encoding = Converting a face image into a bunch of numbers** (vectors) that uniquely represent that face.

[-0.129, 0.044, 0.238, -0.109, ...] # **128 numbers per face**

These numbers capture:

* The **shape** of your eyes
* The **distance** between features
* The **jawline**
* The **skin tone pattern**, etc.
* You don’t compare images pixel by pixel.
* Instead, you compare **encodings** using math (distance between two encoding vectors).
* If the distance is small ➝ it's the **same person.**

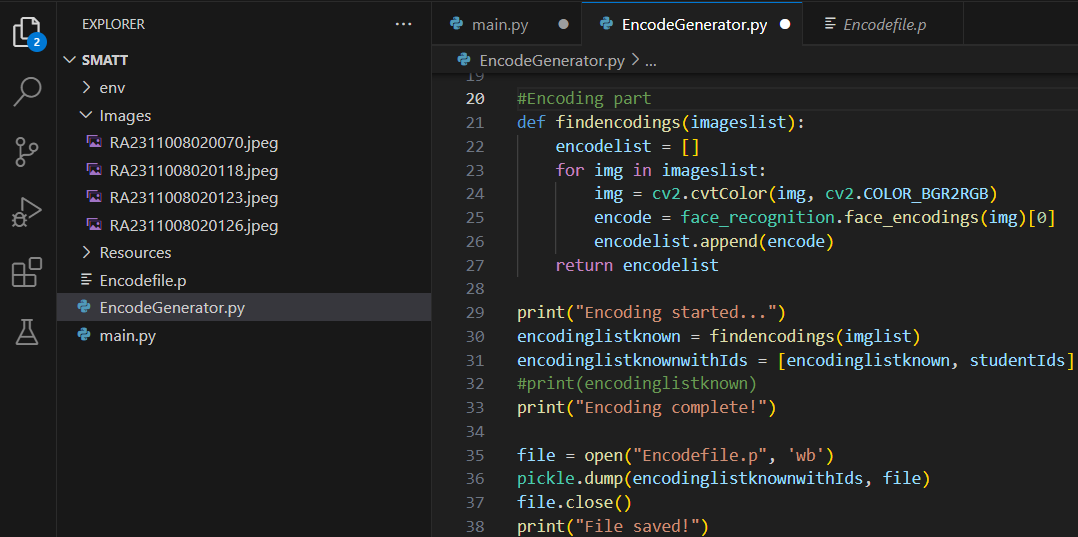
Now, lets create a function which loop through every images and encode each student image on EncodeGenerator.py.



STEPS FOR ENCODING:

1. Convert the color system using cv2.cvtColor(img, cv2.COLOR\_BGR2RGB).
2. Use face\_recognition library and use face\_encodings and feed the image.
3. Use the function on the array having the student’s image (imglist).
4. Save the encodings and StudentID’s for future prediction(pickle file).

When having multiple student images put it under for loop. The output gives 128 numbers per each face.



This is the code for full encoding part. As you see, Encodefile.p had been created.

