☑ Project Overview:

This is a **Face Recognition Attendance System** with **anti-spoofing** using a **GUI made with Tkinter**, leveraging a webcam for real-time face detection, spoof prevention (via sharpness + face presence check), and user registration + login/logout tracking with logs.

📁 Files Used:

1. `main.py`: Main application file with GUI logic and core functionality.
2. `util.py`: Utility functions (provided in your message, like `get_button()`, `recognize()`).
3. `face_test.py`: Contains the anti-spoofing `test()` function.
4. `requirements.txt`: Lists dependencies (OpenCV, Pillow, face_recognition).
5. `db/`: Folder to store user face encodings as `.pickle` files.
6. `log.txt`: Text file to log check-ins and check-outs.
7. `resources/anti_spoof_models`: Folder placeholder (not used with ML in this version, simulated logic instead).

🧠 How It Works (Execution Flow):

🎬 When you run `main.py`:

1. **Main window opens** with:
   - Live webcam feed on the left.
   - 3 buttons on the right: `Login`, `Logout`, `Register New User`.
2. **Webcam starts** capturing frames every 20 ms.

👁‍🗨 Face Detection and Spoofing Logic

📌 Anti-Spoofing (`test()` in `face_test.py`):

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
laplacian_var = cv2.Laplacian(gray, cv2.CV_64F).var()
face_locations = face_recognition.face_locations(image)
```

- Converts to grayscale to calculate **image sharpness** using Laplacian variance.
- Uses `face_recognition.face_locations()` to check if any **face is present**.

```
if face_detected and laplacian_var > 100:
    return 1  # Real face
```

If a face exists and the frame is sharp (not blurry), it's considered **real**.

🧍 Login / Logout Flow

☑ When `Login` or `Logout` is clicked:

```
label = test(...)  # Anti-spoof check
if label != 1:
    show 'spoof detected'
```

1. **Spoofing is checked** using sharpness and face presence.
2. If **passed**, the face is **recognized** using:

```
name = util.recognize(...)
```

This compares the captured face with `.pickle` face encodings in the `db/`.

3. If matched:

```
messagebox.showinfo(...)  # "Welcome, Saket!"
log attendance in log.txt as: "Saket,2025-06-12 19:48:55.214,in"
```

4. If not matched or no face:
    - Shows appropriate error.

🧍 User Registration Flow

☑ When `Register New User` is clicked:

- A new window appears with:
    - Webcam still frame
    - Text input for name
    - `Accept` and `Try Again` buttons

1. After entering name, clicking `Accept`:

```
face_encoding = face_recognition.face_encodings(...)
pickle.dump(encoding, f"{name}.pickle")
```

2. The face is encoded and saved in `db/` as a `.pickle` file.
3. If no face detected, shows error.

📦 Code Walkthrough (Line-by-Line)

🔶 `main.py`

```
class App:
    def __init__(self):  # Initializes the GUI
        self.main_window = tk.Tk()
        self.login_button_main_window = util.get_button(...)
```

- Loads main window with GUI buttons and webcam feed.
- `db/` and `log.txt` are created if not exist.
- `spoof_model_path` is defined for future upgrades.

🖥️ `add_webcam()` + `process_webcam()`

- Sets up webcam stream and calls itself every 20ms for real-time display.

🔓 `handle_auth()`

- Shared between Login/Logout.
- First runs spoof test.
- Then calls `recognize()`.
- If known person: logs name + timestamp in `log.txt`.

📜 `recognize()` in `util.py`:

- Converts the frame to encodings.
- Matches against all saved `.pickle` encodings.
- Returns:
  - `name` if matched
  - `unknown_person` or `no_persons_found` otherwise

🆕 `register_new_user()`

- Opens new window.
- Captures current frame when opened.
- After entering name and clicking Accept:
  - Checks if a face is there.
  - Saves encoding using pickle.

💾 `log.txt` Sample Output:

```
Saket,2025-06-12 19:48:55.214453,in
Saket,2025-06-12 20:10:05.993812,out
```

🧰 `requirements.txt`:

```
opencv-python==4.9.0.80
Pillow==11.2.1
face_recognition==1.3.0
```

⚙️ Summary:

Feature	Status
Real-time webcam feed	☑ Yes
Face recognition	☑ Yes
Anti-spoofing (basic)	☑ Yes
GUI with Tkinter	☑ Yes
Attendance logging	☑ Yes
User registration	☑ Yes

☑ 1. `main.py`

```
import os
import datetime
import pickle
import tkinter as tk
import cv2
from PIL import Image, ImageTk
import face_recognition

import util
from face_test import test  # Anti-spoof function


class App:
    def __init__(self):
        self.main_window = tk.Tk()
        self.main_window.geometry("1200x520+350+100")
        self.main_window.title("Face Attendance System")

        # Buttons
        self.login_button_main_window = util.get_button(
            self.main_window, 'Login', 'green', self.login)
        self.login_button_main_window.place(x=750, y=200)

        self.logout_button_main_window = util.get_button(
            self.main_window, 'Logout', 'red', self.logout)
        self.logout_button_main_window.place(x=750, y=300)

        self.register_new_user_button_main_window = util.get_button(
            self.main_window, 'Register New User', 'gray',
            self.register_new_user, fg='black')
        self.register_new_user_button_main_window.place(x=750, y=400)

        # Webcam feed
        self.webcam_label = util.get_img_label(self.main_window)
        self.webcam_label.place(x=10, y=0, width=700, height=500)
        self.add_webcam(self.webcam_label)

        # Paths
        self.db_dir = './db'
        os.makedirs(self.db_dir, exist_ok=True)
        self.log_path = './log.txt'
        self.spoof_model_path = './resources/anti_spoof_models'

    def add_webcam(self, label):
        if 'cap' not in self.__dict__:
            self.cap = cv2.VideoCapture(0)
        self._label = label
        self.process_webcam()

    def process_webcam(self):
        ret, frame = self.cap.read()
        if not ret:
            return

        self.most_recent_capture_arr = frame
        img_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        self.most_recent_capture_pil = Image.fromarray(img_rgb)
        imgtk = ImageTk.PhotoImage(image=self.most_recent_capture_pil)
```

```
        self._label.imgtk = imgtk
        self._label.configure(image=imgtk)
        self._label.after(20, self.process_webcam)

    def handle_auth(self, is_login=True):
        label = test(
            image=self.most_recent_capture_arr,
            model_dir=self.spoof_model_path,
            device_id=0
        )

        if label != 1:
            util.msg_box('Spoof Detected!', 'You are not a real person!')
            return

        name = util.recognize(self.most_recent_capture_arr, self.db_dir)

        if name in ['unknown_person', 'no_persons_found']:
            util.msg_box('Unknown User', 'Please register first or try
again.')
        else:
            action = 'in' if is_login else 'out'
            message = f"Welcome, {name}!" if is_login else f"Goodbye,
{name}!"
            util.msg_box('Authentication Successful', message)

            with open(self.log_path, 'a') as f:
                f.write(f"{name},{datetime.datetime.now()},{action}\n")

    def login(self):
        self.handle_auth(is_login=True)

    def logout(self):
        self.handle_auth(is_login=False)

    def register_new_user(self):
        self.register_new_user_window = tk.Toplevel(self.main_window)
        self.register_new_user_window.geometry("1200x520+370+120")
        self.register_new_user_window.title("Register New User")

        self.accept_button_register_new_user_window = util.get_button(
            self.register_new_user_window, 'Accept', 'green',
            self.accept_register_new_user)
        self.accept_button_register_new_user_window.place(x=750, y=300)

        self.try_again_button_register_new_user_window = util.get_button(
            self.register_new_user_window, 'Try Again', 'red',
            self.try_again_register_new_user)
        self.try_again_button_register_new_user_window.place(x=750, y=400)

        self.capture_label =
util.get_img_label(self.register_new_user_window)
        self.capture_label.place(x=10, y=0, width=700, height=500)
        self.add_img_to_label(self.capture_label)

        self.entry_text_register_new_user = util.get_entry_text(
            self.register_new_user_window)
        self.entry_text_register_new_user.place(x=750, y=150)

        self.text_label_register_new_user = util.get_text_label(
            self.register_new_user_window, 'Please,\ninput username:')
```

```
        self.text_label_register_new_user.place(x=750, y=70)

    def try_again_register_new_user(self):
        self.register_new_user_window.destroy()

    def add_img_to_label(self, label):
        imgtk = ImageTk.PhotoImage(image=self.most_recent_capture_pil)
        label.imgtk = imgtk
        label.configure(image=imgtk)
        self.register_new_user_capture =
self.most_recent_capture_arr.copy()

    def accept_register_new_user(self):
        name = self.entry_text_register_new_user.get(1.0, "end-1c").strip()

        if not name:
            util.msg_box("Error", "Name cannot be empty!")
            return

        try:
            embeddings =
face_recognition.face_encodings(self.register_new_user_capture)[0]
        except IndexError:
            util.msg_box("Error", "No face detected! Try again.")
            return

        with open(os.path.join(self.db_dir, f'{name}.pickle'), 'wb') as
file:
            pickle.dump(embeddings, file)

        util.msg_box('Success!', f'User "{name}" registered successfully!')
        self.register_new_user_window.destroy()

    def start(self):
        self.main_window.mainloop()


if __name__ == "__main__":
    app = App()
    app.start()
```

☑ 2. `util.py`

```
import os
import pickle
import tkinter as tk
from tkinter import messagebox
import face_recognition


def get_button(window, text, color, command, fg='white'):
    return tk.Button(
        window,
        text=text,
        activebackground="black",
        activeforeground="white",
        fg=fg,
        bg=color,
        command=command,
        height=2,
        width=20,
        font=('Helvetica bold', 20)
    )


def get_img_label(window):
    label = tk.Label(window)
    label.grid(row=0, column=0)
    return label


def get_text_label(window, text):
    label = tk.Label(window, text=text)
    label.config(font=("sans-serif", 21), justify="left")
    return label


def get_entry_text(window):
    return tk.Text(window, height=2, width=15, font=("Arial", 32))


def msg_box(title, description):
    messagebox.showinfo(title, description)


def recognize(img, db_path):
    try:
        unknown_encodings = face_recognition.face_encodings(img)
    except Exception as e:
        print(f"Encoding error: {e}")
        return 'no_persons_found'

    if len(unknown_encodings) == 0:
        return 'no_persons_found'

    unknown_encoding = unknown_encodings[0]

    for filename in sorted(os.listdir(db_path)):
        if filename.endswith(".pickle"):
            path = os.path.join(db_path, filename)
```

```
            try:
                with open(path, 'rb') as file:
                    known_encoding = pickle.load(file)

                is_match = face_recognition.compare_faces([known_encoding],
unknown_encoding)[0]

                if is_match:
                    return filename.replace('.pickle', '')

            except Exception as e:
                print(f"Error loading {filename}: {e}")
                continue

    return 'unknown_person'
```

☑ 3. face_test.py

```
import cv2
import face_recognition


def test(image, model_dir=None, device_id=0):
    """
    Simulated anti-spoofing detector.
    Uses sharpness and presence of face to determine real vs spoof.

    Args:
        image (numpy.ndarray): BGR webcam frame
        model_dir (str): Optional path to models (not used)
        device_id (int): Optional GPU ID (not used)

    Returns:
        int: 1 = Real face, 0 = Spoof or unclear input
    """
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    laplacian_var = cv2.Laplacian(gray, cv2.CV_64F).var()
    face_locations = face_recognition.face_locations(image)
    face_detected = len(face_locations) > 0

    sharpness_threshold = 100
    if face_detected and laplacian_var > sharpness_threshold:
        return 1
    else:
        return 0
```

☑ 4. requirements.txt

```
opencv-python==4.9.0.80
Pillow==11.2.1
face_recognition==1.3.0
```

☑ Step-by-Step Flow After Running `main.py`

🟢 Step 1: Python Execution Begins

```
if __name__ == "__main__":
    app = App()
    app.start()
```

- The `App` class is instantiated.
- `__init__()` runs → sets up GUI, webcam, buttons, and paths.
- Then `start()` is called → launches the Tkinter main GUI loop (`mainloop()`).

🪟 Step 2: Main Window Appears

Your app opens a GUI window with:

Section	What's There
Left side	Live **webcam feed** (from `cv2.VideoCapture(0)`)
Right side	3 buttons: ☑ `Login` 🔴 `Logout` 🆕 `Register New User`

📸 Step 3: Webcam Starts Automatically

Function: `process_webcam()`

- Captures one frame from webcam every ~20ms (`after(20, self.process_webcam)`).
- Converts the frame to RGB for `PIL` display.
- Shows the webcam feed live in the left panel.

🟩 Step 4: You Click a Button — e.g., Login

```
self.login_button_main_window = util.get_button(..., self.login)
```

- Triggers: `self.login()` → internally calls `self.handle_auth(is_login=True)`

🔐 Step 5: Anti-Spoofing Check (Real vs Fake)

```
label = test(image=frame)
```

In `face_test.py`:

- Converts image to grayscale.
- Calculates **image sharpness** using Laplacian variance.
- Checks if a **face** exists in the frame.

If face is present and image is sharp → returns `1` (Real)
Otherwise → `0` (Spoofed, unclear, or no face)

❌ If `label != 1`:

Popup message: **"Spoof Detected! You are not a real person!"**
☑ Execution stops here.

🧠 What is Laplacian Variance?

Laplacian Variance is a **mathematical measure of an image's sharpness** — often used to detect **blur** in an image.

📸 Why is it useful?

In your face attendance system, you're using Laplacian variance to **detect if the captured webcam frame is clear (real face)** or **blurry (possibly spoof or unclear)**. It's a lightweight, fast anti-spoofing technique.

☑ Step-by-Step Explanation:

1. Laplacian Filter:

The **Laplacian** is a second-order derivative filter that highlights regions of **rapid intensity change** (edges).

```
laplacian = cv2.Laplacian(gray_image, cv2.CV_64F)
```

This gives you an image showing where edges (i.e. sharp details) exist.

---

2. Variance:

```
laplacian_var = laplacian.var()
```

- `variance` measures **spread or contrast** in edge strength.
- **Higher variance** → sharper image (more edge detail).
- **Lower variance** → blurry image (edges are smooth or missing).

🔢 Example Values:

Laplacian Variance	Interpretation
~0 to 50	Very blurry (probably spoof or motion)
100+	Sharp (likely real face)
500+	Very sharp (high-quality frame)

In your code:

```
if face_detected and laplacian_var > 100:
    return 1  # Real
else:
    return 0  # Spoof
```

🔍 Real Example in Code:

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
laplacian_var = cv2.Laplacian(gray, cv2.CV_64F).var()
```

🎯 Why It's Used for Spoof Detection:

- **Blurry images** are common with printed photos, screen displays, or movement.
- Real, live human faces in front of a webcam are **sharper**.
- So Laplacian variance becomes a quick "**liveness indicator**".

🧠 Step 6: Face Recognition

If real person, then:

```
name = util.recognize(frame, db_path="./db")
```

In `util.py`:

- Extracts face encodings from the image.
- Loads each `.pickle` file (user's face encodings) from `db/`.
- Compares captured face encoding with all known encodings using:
- `face_recognition.compare_faces(...)`
- If a match is found → returns the `username`
- If not → returns `'unknown_person'` or `'no_persons_found'`

☑ Step 7: Show Result (Success or Error)

Case A: If Match Found

- MessageBox: `"Welcome, Saket!"` (if login) or `"Goodbye, Saket!"` (if logout)
- Logs attendance to `log.txt`:
- `Saket,2025-06-12 20:13:44.295123,in`

Case B: If No Match

- MessageBox: `"Unknown User. Please register first or try again."`

🆕 Optional Step: Register New User

Clicking the "**Register New User**" button triggers:

```
self.register_new_user()
```

This:

- Opens a **new window**.
- Displays the current still webcam frame.
- Prompts for name.
- User types a name → clicks **Accept**

Then:

- Captures face from the frame.
- Converts it to face encoding.

- Saves it in `db/` folder as `{username}.pickle`.

Success popup: **"User registered successfully!"**

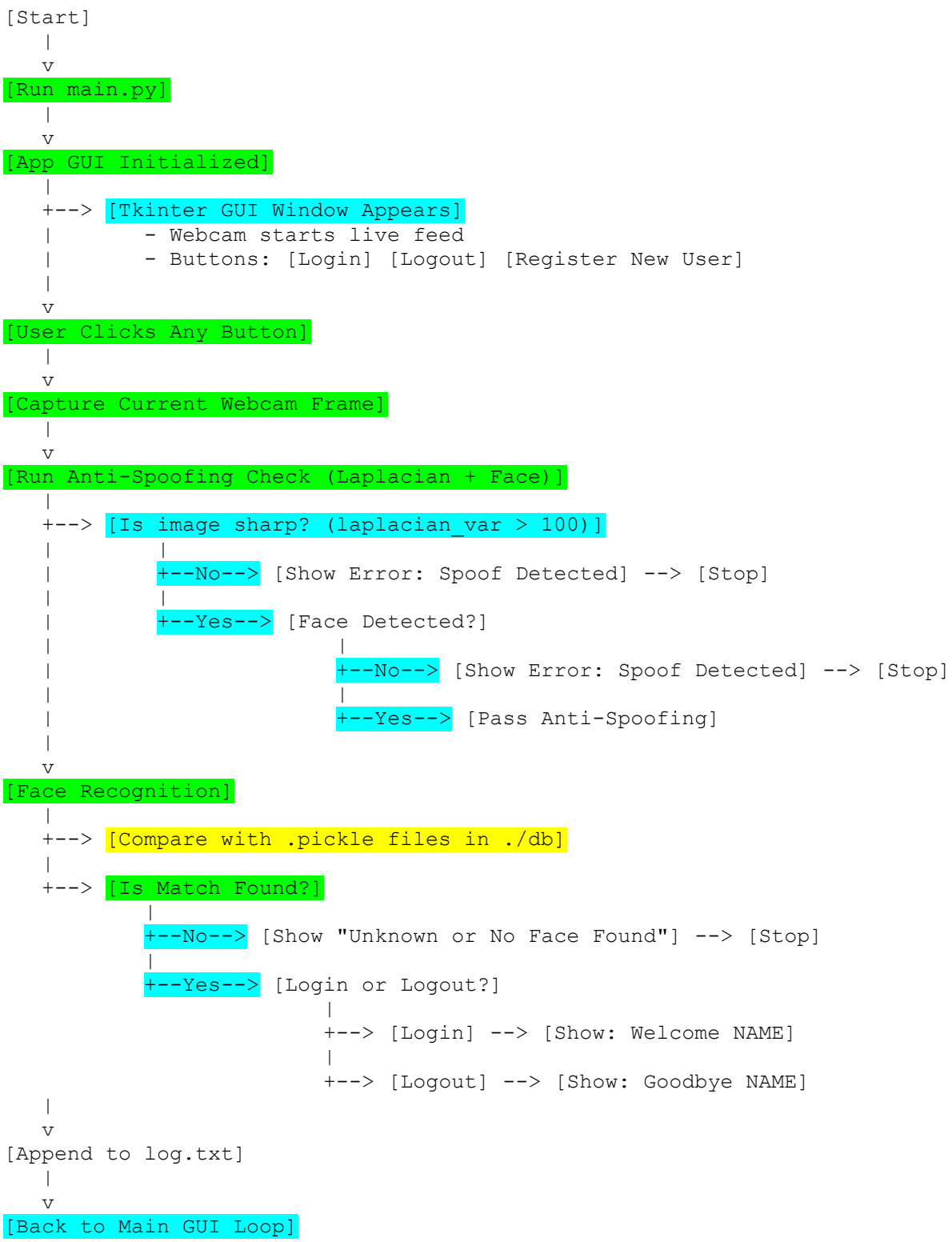
📁 Summary of Execution Flow

```
main.py
  ↓
App() → __init__()
  ↓
GUI launched with webcam and buttons
  ↓
User clicks Login/Logout/Register
  ↓
→ Anti-spoofing check
→ Face recognition
→ Logging or Registration
→ User notified with popup
```

📦 Final Outputs/Changes:

- ☑️ Face encodings saved as `.pickle` in `db/`
- 📝 Login/logout actions appended in `log.txt`
- 📸 Webcam feed runs in real-time
- 🛡️ Spoofing blocked via blur + no-face detection

☑ Face Attendance System – Execution Flowchart



📝 Notes:

- **Laplacian Variance > 100** is the **anti-spoof threshold**.
- `recognize()` checks face against the `.pickle` files (known faces).
- All GUI buttons reuse the same webcam feed.
- If "Register New User" is clicked, a **new window** opens for input.

🛠️ Why CMake and Visual Studio C++ Build Tools Are Needed

Your project uses this line:

```
import face_recognition
```

Under the hood, this library depends on:

- **`dlib`** (a C++-based machine learning toolkit)
- Which in turn uses:
  - C++ compilers
  - CMake for building native extensions
  - Visual Studio build tools on Windows

☑ Usage of Each Tool

1. CMake

🔧 Purpose:
CMake is a build automation tool. It **configures** and **generates native build files** (e.g. `.sln`, `.vcxproj`) that are used to compile C++ code.

📌 In This Project:
When you install `face_recognition` (or `dlib`), CMake:

- Configures the environment
- Generates make/build files to compile the C++ extension
- Links the compiled binaries into Python

---

2. Visual Studio C++ Build Tools

🧱 **Purpose:**
These tools provide the **compiler (MSVC)** and standard C++ libraries necessary to **compile dlib** and its dependencies on Windows.

📌 **In This Project:**
Used during:

- The pip installation of `face_recognition` or `dlib`
- Compiling C++ extensions like `face_recognition_models`, which contain the face encoders, landmark detectors, etc.

🚫 Not Directly Used in Your Code

You're not writing or compiling C++ manually — but these tools are **essential for installing the libraries that require native compilation**.

When I said **"Not directly used in your code"**, I meant:
You **did not write any C++ code** or explicitly invoke **CMake** or **Visual Studio compilers** in your Python files like `main.py`, `util.py`, or `face_test.py`.

But they're still **used behind the scenes**, specifically when you **install the `face_recognition` library**, which depends on native (C++) code.

🔄 So, How are CMake and Visual Studio C++ Tools Used?

🧩 It happens during installation of the following package:

```
pip install face_recognition
```

Which depends on:

```
face_recognition → dlib → C++ code → needs compiling
```

🔧 Behind the scenes:

1. When you run:
2. `pip install face_recognition`
3. `pip` downloads the source code of `dlib` (a C++ library).
4. `setup.py` of `dlib` runs and:

- Calls **CMake** to configure the build
- Uses **Visual Studio C++ compiler** (`cl.exe`) to compile `.cpp` files
- Produces a `.pyd` file (a compiled Python module)

This compiled `.pyd` file is what your Python code imports when you write:

```
import face_recognition
```

📦 Example:

A part of `dlib`'s output when installing:

```
-- Building for: Visual Studio 16 2019
-- The C compiler identification is MSVC 19.29.30133.0
-- Configuring done
-- Generating done
-- Build files have been written to: C:/Users/.../build
```

📝 Summary Table:

Task	Who Triggers It	Tools Used
Install `face_recognition`	`pip` or `setup.py`	`CMake`, `Visual Studio C++ Compiler`
Compile C++ code (like dlib)	`pip install` step	MSVC (`cl.exe`), `nmake`, etc.
Use in Python	`import face_recognition`	No tools needed (already compiled)

🧠 Final Clarification:

You **don't manually run** CMake or Visual Studio in your project,
but your project **relies on their output** because `face_recognition` is built with them.
